Microsoft

Microsoft WINDOWS SERVER 2003 PERFORMANCE GUIDE

Mark Friedman with the
Microsoft Windows Performance Team

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Congress Control Number 2005921847

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9   QWT   8 7 6 5 4 3

Distributed in Canada by H.B. Fenn and Company Ltd. A CIP catalogue record for this book is available from the British Library.

A CIP catalogue record for this book is available from the British Library.

Acquisitions Editor: Martin DelRe
Project Editor: Karen Szall
Copy Editor: Victoria Thulman
Technical Editor: Mitch Tulloch
Indexer: Tony Ross and Lee Ross

SubAsy Part No. X11-06988
Body Part No. X11-06989

Contents at a Glance

1 Performance Monitoring Overview 1

2 Performance Monitoring Tools 125

3 Measuring Server Performance 235

4 Performance Monitoring Procedures 313

5 Performance Troubleshooting 401

6 Advanced Performance Topics 537

Contents

About the Author xi

Acknowledgments xiii

Introduction xv

Document Conventions xv

Resource Kit Companion CD xvi

1 Performance Monitoring Overview 1

- Introducing Performance Monitoring 2
  - Learning About Performance Monitoring 3
  - Proactive Performance Monitoring 4
  - Diagnosing Performance Problems 5
  - Scalability 8
- Performance Monitoring Concepts 12
  - Definitions 13
  - Bottlenecks 22
  - Utilization Law 28
  - Queue Time and Utilization 29
  - Little's Law 33
  - Conclusions 36
- System Architecture 37
  - Using the Performance Monitor 37
  - Operating Systems 38
  - Processors 43
  - Memory and Paging 61
  - The I/O Subsystem 89
  - Network Interfaces 98
- Summary 123

2 Performance Monitoring Tools 125

- Summary of Monitoring Tools 126
  - Performance Statistics 126

Event Traces 127
Load Generating and Testing. 128
Administrative Controls 128
Required Security for Tool Usage 128
Performance Monitoring Statistics 133
Performance Objects. 134
Performance Counters 140
System Monitor 144
Viewing a Chart in Real Time. 145
Changing the Sampling Interval 146
Creating a Custom Monitoring Configuration 147
Saving Real-Time Data 148
Customizing How Data Is Viewed. 149
Tips for Working with System Monitor 151
Task Manager 154
Working with Task Manager 155
Monitoring Applications. 156
Monitoring Processes 157
Monitoring Performance 161
Monitoring the Network 163
Monitoring Users. 167
Automated Performance Monitoring. 168
Performance Logs and Alerts. 168
Counter Logs 169
Tips for Working with Performance Logs and Alerts. 178
Creating Performance Logs Using Logman. 178
Managing Performance Logs. 191
Using the Relog Tool 192
Using Typeperf Queries 200
Windows Performance Monitoring Architecture 207
Performance Library DLLs 207
Performance Counter Text String Files 208
Performance Data Helper Processing. 208
Disable Performance Counters 210
Remote Monitoring. 210
Event Tracing for Windows. 212
Event Tracing Overview 212

Using Log Manager to Create Trace Logs 217
Event Trace Reports 224
Alerts 225
Configuring Alerts 226
Configuring Alert Notification 229
Windows System Resource Manager. 233
Network Monitor 233

3 Measuring Server Performance. 235
Using Performance Measurements Effectively. 237
Identifying Bottlenecks 237
Management by Exception. 238
Key Performance Indicators 241
System and Application Availability 241
Processor Utilization 242
Monitoring Memory and Paging Rates 256
Monitoring Disk Operations. 279
Managing Network Traffic 292
Maintaining Server Applications 297
Terminal Services 310

4 Performance Monitoring Procedures 313
Understanding Which Counters to Log 314
Background Performance Monitoring 314
Management Reporting 315
Capacity Planning. 316
Daily Server Monitoring Procedures 316
Daily Counter Logs 317
Using Alerts Effectively 330
Daily Management Reporting 338
Historical Data for Capacity Planning 346
Automated Counter Log Processing 354
Using a SQL Server Repository 365
Using the System Monitor Console with SQL Server 366
How to Configure System Monitor to Log to SQL Server 367
Counter Log Database Schema 369
Querying the SQL Performance Database 373

Capacity Planning and Trending 379
Organizing Data for Capacity Planning 380
Forecasting Techniques 383
Counter Log Scenarios 388
Logging Local Counters 388
Monitoring Remote Servers in Real Time 392
Troubleshooting Counter Collection Problems 395
Missing Performance Counters 396
Restoring Corrupt Performance Counters 400

5 Performance Troubleshooting 401

Bottleneck Analysis 402
Baseline Data 402
Current Performance Levels 403
Resource Utilization and Queue Length 403
Decomposition 404
Analysis Procedures 404
Understanding the Problem 404
Analyzing the Logged Performance Data 404
Analyzing Performance Data Interactively 405
Fine-Grained Analysis Tools 405
What to Check Next in the Enterprise 406
Processor Troubleshooting 406
Resource Utilization and Queue Length 407
Decomposition 409
Identifying a Runaway Process by Using Task Manager 413
Identifying a Runaway Process by Using a Counter Log 416
Memory Troubleshooting 430
Counters to Evaluate When Troubleshooting Memory Performance 431
What to Check Next When Troubleshooting Memory Performance 434
Excessive Paging 435
Virtual Memory Shortages 448
32-Bit Virtual Memory Addressing Limits 456
Disk Troubleshooting 464
Disk Performance Expectations 466
Diagnosing Disk Performance Problems 489

Network Troubleshooting . . . 509
Counters to Log When Troubleshooting Network Performance . . . 509
Counters to Evaluate When Troubleshooting Network Performance . . . 511
LAN Performance . . . 514
WAN Performance . . . 526

6 Advanced Performance Topics . . . 537
Processor Performance . . . 538
Instruction Execution Throughput . . . 539
Time-Slicing Revisited . . . 546
Multiprocessors . . . 548
Memory Performance . . . 608
Extended Virtual Addressing in 32-Bit Machines . . . 608
64-Bit Virtual Memory . . . 621
Forecasting Memory Requirements . . . 622
The System Monitor Automation Interface . . . 628
Adding the System Monitor ActiveX Control to a Web Page . . . 629
Customizing the System Monitor ActiveX Control . . . 630
Configuring the System Monitor ActiveX Control Display Type . . . 632
Configuring the System Monitor ActiveX Control Sampling Rate . . . 634
Manually Retrieving Performance Data . . . 634
Configuring the System Monitor ActiveX Control's Appearance . . . 636
Configuring the System Monitor ActiveX Control Color Schemes . . . 637
Configuring the System Monitor ActiveX Control Font Styles . . . 638
Adding Performance Counters to the System Monitor ActiveX Control . . . 639
Configuring System Monitor ActiveX Control Performance Counters . . . 640
Removing Performance Counters from the System Monitor ActiveX Control . . . 641
Using Counter Paths to Track Individual Performance Counters . . . 642
Creating a Web Page for Monitoring Performance . . . 643
Drag-and-Drop Support . . . 645

Glossary . . . 649

Index . . . 681

About the Author

Mark Friedman is the president and CEO of Demand Technology Software (DTS) in Naples, Florida, and is responsible for the development of Microsoft® Windows Server™ performance monitoring and capacity planning tools for large-scale enterprises. Mark founded DTS in 1996, and the company became a Microsoft ISV partner in 1997. He has been programming on the Windows platform since 1990.

Mark founded OnDemand Software in 1994 and was chairman of the board of directors of that company when it was sold to Seagate Technology in the spring of 1996. OnDemand developed and marketed utility software, including the award-winning WinInstall software distribution package. Between 1987–1991, he was a director of product development at Landmark Systems and responsible for the design and development of TMON/MVS, a leading mainframe performance monitoring product.

Mark is a recognized expert in computer performance, disk and tape performance, and storage management. He is the author of over 100 technical articles and papers on these subjects. He co-authored (with Dr. Odysseas Pentakalos) the book *Windows 2000 Performance Guide* (O'Reilly, 2002). Mark's training seminars, lectures, and published work are highly regarded for their technical quality and depth, and he is esteemed for his ability to communicate complex technical topics in plain, concise terms.

He holds a master's degree in computer science from Temple University.

Thank you to those who contributed to the *Microsoft Windows Server 2003 Performance Guide* and *Microsoft Windows Server 2003 Troubleshooting Guide*.

Technical Writing Lead: David Stern

Writers: Mark Friedman, Tony Northrup, David Stern, Brit Weston

Editors: Carolyn Eller, Paula Younkin, Julia Ziobro

Project Manager: Cliff Hall

Production Lead: Jim Bevan

Art Production: Chris Blanton, David Hose, Jon Billow

Technical Contributors: Jee Fung Pang, Iain Frew, Neel Jain, Inder Sethi, Brad Waters, Bruce Worthington Ahmed Talat, Tom Hawthorn, Adam Berkan, Oscar Omar Garza Santos, Rick Vicik, Kathy Sestrap, David Stern, Jon Wojan, Ben Christenbury, Steve Patrick, Greg Cottingham, Rick Anderson, Khalil Nasser, Darrell Gorter, Andrew Ritz, Jeremy Cahill, Rob Haydt, Jonathan V. Smith, Matt Holle, Jamie Schwartz, Keith Hageman, Terence Hosken, Karan Mehra, Tony Donno, Joseph Davies, Greg Marshall, Jonathan Schwartz, Chittur Subbaraman, Clark Nicholson, Bob Fruth, Lara Sosnosky, Charles Anthe, Tim Lytle, Adam Edwards, Simon Muzio, Mike Hillberg, Vic Heller, Prakash Rao, Ilan Caron, Shy Cohen, Ashwin Palekar, Matt Desai, Mahmood Dhalla, Joseph Dadzie, David Cross, Jiandong Ruan, Stephane St-Michel, Kamen Moutafov, KC Lemson, Jim Cavalaris, Jeff Westhead, Glenn Pittaway, Stephen Hui, Davide Massarenti, David Kruse, Chris Evans, Brian Granowitz, David Lee, Neta Amit, Avi Shmueli, Jim Thatcher, Pung Xu, Steve Olsson, Ran Kalach, Brian Dewey, V Raman, Paul Mayfield, David Eitelbach, Jaroslav Dunajsky, Alan Warwick, Pradeep Madhavarapu, Kahren Tevosyan, Huei Wang, Ido Ben-Shachar, Florin Teodorescu, Michael Hills, Fred Bhesania, Randy Aull, Sachin Seth, Chris Stackhouse, David Fields, Stuart Sechrest, Landy Wang, Duane Thomas, Lisa Cipriano, Kristin Thomas, Stewart Cox, Joseph Davies, Pilar Ackerman, Cheryl Jenkins

From the Microsoft Press editorial team, the following individuals contributed to the *Microsoft Windows Server 2003 Performance Guide*:

Product Planner: Martin DelRe

Project Editor: Karen Szall

Technical Reviewer: Mitch Tulloch

Copy Editor: Victoria Thulman

Production Leads: Dan Latimer and Elizabeth Hansford

Indexers: Tony Ross and Lee Ross

Art Production: Joel Panchot and William Teel

Introduction

Welcome to *Microsoft® Windows Server™ 2003 Performance Guide*.

The *Microsoft Windows 2003 Server Resource Kit* consists of seven volumes and a single compact disc (CD) containing tools, additional reference materials, and an electronic version of the books (eBooks).

The *Microsoft Windows Server 2003 Performance Guide* is your technical resource for optimizing the performance of computers and networks running on the Microsoft Windows Server 2003 operating system. Windows Server 2003 provides a comprehensive set of features that helps you automate the management of most workloads and configurations. It also provides a powerful set of performance monitoring tools and performance-oriented settings that you can use to fine-tune system performance. Use this guide to gain a basic understanding of performance concepts and strategies so that you can optimize the speed, reliability, and efficiency of your Windows Server 2003 operating system.

Document Conventions

Reader alerts are used throughout the book to point out useful details.

Reader Alert	Meaning
Tip	A helpful bit of inside information on specific tasks or functions
Note	Alerts you to supplementary information
Important	Provides information that is essential to the completion of a task
Caution	Important information about possible data loss, breaches of security, or other serious problems
Warning	Information essential to completing a task, or notification of potential harm

The following style conventions are used in documenting command-line tasks throughout this guide.

Element	Meaning
Bold font	Characters that you type exactly as shown, including commands and parameters. User interface elements also appear in boldface type.
Italic font	Variables for which you supply a specific value. For example, *Filename.ext* can refer to any valid file name.
`Monospace font`	Code samples.
`%SystemRoot%`	Environment variables.

Resource Kit Companion CD

The companion CD includes a variety of tools and resources to help you work more efficiently with Microsoft Windows® clients and servers.

Note The tools on the CD are designed to be used on Windows Server 2003 or Windows XP (or as specified in the documentation of the tool).

The Resource Kit companion CD includes the following:

- *The Microsoft Windows Server 2003 Resource Kit* tools—a collection of tools and other resources that help you to efficiently harness the power of Windows Server 2003. Use these tools to manage Microsoft Active Directory® directory services, administer security features, work with the registry, automate recurring jobs, and perform many other tasks. Use the Tools Help documentation to discover and learn how to use these administrative tools.
- Windows Server 2003 Technical Reference—documentation that provides comprehensive information about the technologies included in the Windows Server 2003 operating system, including Active Directory and Group Policy, as well as core operating system, high availability and scalability, networking, storage, and security technologies.
- Electronic version (eBook) of this guide as well as an eBook of each of the other volumes in the *Microsoft Windows Server 2003 Resource Kit.*
- EBooks of *Microsoft Encyclopedia of Networking*, Second Edition, *Microsoft Encyclopedia of Security, Internet Information Services (IIS) 6 Resource Kit*, and *Microsoft Scripting Self-Paced Learning Guide.*
- Sample chapters from the *Assessing Network Security* and *Microsoft Windows Server 2003 PKI and Certificate Security* books.

- VBScript Essentials Videos—videos from the *Microsoft Windows Administrator's Automation Toolkit.*
- A link to the eLearning site where you can access free eLearning clinics and hand-on labs.
- An online book survey that gives you the opportunity to comment on your Resource Kit experience as well as influence future Resource Kit publications.

Resource Kit Support Policy

Microsoft does not support the tools supplied on the *Microsoft Windows Server 2003 Resource Kit* CD. Microsoft does not guarantee the performance of the tools, or any bug fixes for these tools. However, Microsoft Press provides a way for customers who purchase *Microsoft Windows Server 2003 Resource Kit* to report any problems with the software and receive feedback for such issues. To report any issues or problems, send an e-mail message to *rkinput@microsoft.com*. This e-mail address is only for issues related to *Microsoft Windows Server 2003 Resource Kit* and any of the volumes within the Resource Kit. Microsoft Press also provides corrections for books and companion CDs through the World Wide Web at http://www.microsoft.com/learning/support/. To connect directly to the Microsoft Knowledge Base and enter a query regarding a question or issue you have, go to http://support.microsoft.com. For issues related to the Microsoft Windows Server 2003 operating system, please refer to the support information included with your product.

Chapter 1

Performance Monitoring Overview

In this chapter:

Introducing Performance Monitoring 2
Performance Monitoring Concepts 12
System Architecture. 37
Summary 123

Comprehensive measurement data on the operation and performance of computers running the Microsoft Windows Server 2003 operating system makes these systems easy to manage. Windows Server 2003 provides powerful and comprehensive features that manage the performance of most workloads and configurations automatically. The presence of these advanced features means you usually do not need to intervene manually to try and coax better performance out of your Windows Server 2003 configuration. Nevertheless, the automatic management facilities might not be optimal in every case. For those situations in which the performance of applications is slow or otherwise less than optimal, you can use a variety of tunable, performance-oriented settings to gain better performance. How to use the most important settings that are available to fine-tune your Windows Server 2003 machines is one of the main areas of discussion in Chapter 6, "Advanced Performance Topics," in this book.

Caution There are many performance settings and tuning parameters that you can use in Windows Server 2003. The wrong values for these performance and tuning settings can easily do your system more harm than good. Changing the system's default settings should be attempted only after a thorough study has convinced you that the changes contemplated are likely to make things better. How to conduct a study to determine this is one of the important topics discussed throughout this section of the book.

When your performance is not optimal, Windows Server 2003 provides a rich set of tools for monitoring the performance of a computer system and its key components. These components include the hardware (for example, the processor, disks, and memory); the network; the operating system and its various services; the major Server subsystems (for example, security, file, print, Web publishing, database, and messaging); and the specific application processes that are executing. Rest assured that whatever role your Windows Server 2003 machine is configured to play in your environment, your machine will be capable of providing ample information about its operating characteristics and performance. This chapter shows you how to use this information effectively to solve a wide variety of problems, from troubleshooting a performance problem to planning for the capacity required to support a major new application running on your Windows Server 2003 farm.

This chapter introduces the major facilities for performance monitoring in Windows Server 2003. It begins by describing the empirical approach that experienced computer performance analysts should adopt to address performance problems. It then reviews the key concepts that apply whenever you are diagnosing computer performance. Because computer performance deals with measurements and their relationship to each other, some of the discussion concerns these mathematical relations.

The central topic in this chapter is an introduction to computer system architecture from a primarily performance point of view. This introduction focuses on how Windows Server 2003 works so that you will be able to use the performance statistics it generates more effectively. It also introduces the most important performance counters that are available for the major system components on your Windows Server 2003 computers. It discusses what these measurements mean and explains how they are derived. A more comprehensive discussion of the most important performance statistics is provided in Chapter 3, "Measuring Server Performance," in this book.

Introducing Performance Monitoring

Configuring and tuning computer systems for optimal performance are perennial concerns among system administrators. Users of Windows Server 2003 applications who rely on computer systems technology to get important work done are naturally also concerned about good performance. When computer performance is erratic or the response time of critical applications is slow, these consumers are forced to work less efficiently. For example, customers visiting a .NET e-commerce Web site facing elongated response times might become dissatisfied and decide to shop elsewhere.

The ability to figure out *why* a particular system configuration is running slowly is a desirable skill that is partly science and partly art. Whatever level of skill or artistry you possess, gathering the performance data is a necessary first step to diagnosing and resolving a wide range of problems. Determining which data to collect among all the performance statistics that can be gathered on a Windows Server 2003 machine is itself a daunting task. Knowing which tool to choose among the different tools supplied with Windows Server 2003 to gather the performance data you need is another important skill to learn. Finally, learning to understand and interpret the performance data that you gather is another valuable area of expertise you must cultivate. This section of this book is designed to help you with all these aspects of performance monitoring. Although reading this chapter and the related chapters in this book will not immediately transform you into a performance wizard, it will provide the background necessary for you to acquire that knowledge and skill.

Learning About Performance Monitoring

This introductory chapter looks at the basics of performance monitoring by explaining how a Windows Server 2003 computer system works. It provides an overview of the performance data that is available and what it can be used for. If you are experienced with performance monitoring, you might want to skim through this chapter and move to a more challenging one. Figure 1-1 provides a basic roadmap for the chapters of this book and can help you decide where to start your reading.

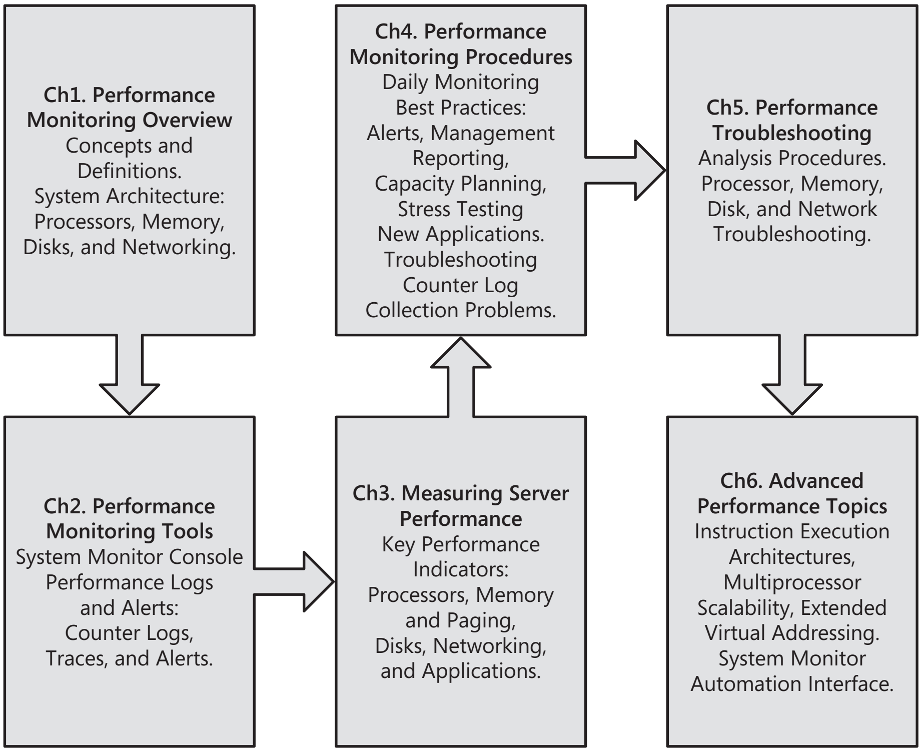

Ch1. Performance Monitoring Overview
Concepts and Definitions.
System Architecture: Processors, Memory, Disks, and Networking.
Ch2. Performance Monitoring Tools
System Monitor Console
Performance Logs and Alerts:
Counter Logs, Traces, and Alerts.
Ch3. Measuring Server Performance
Key Performance Indicators:
Processors, Memory and Paging, Disks, Networking, and Applications.
Ch4. Performance Monitoring Procedures
Daily Monitoring
Best Practices:
Alerts, Management Reporting, Capacity Planning, Stress Testing New Applications.
Troubleshooting Counter Log Collection Problems.
Ch5. Performance Troubleshooting
Analysis Procedures.
Processor, Memory, Disk, and Network Troubleshooting.
Ch6. Advanced Performance Topics
Instruction Execution Architectures, Multiprocessor Scalability, Extended Virtual Addressing.
System Monitor Automation Interface.

Figure 1-1 Chapter roadmap

Proactive Performance Monitoring

Experienced drivers are careful to monitor their vehicles' gas gauges on a regular basis so that they know how much gas is left in the tank and can stop to refuel before the tank runs dry. That, of course, is the idea behind the gas gauge in the first place—to monitor your fuel supply so that you take action to avert a problem before it occurs. No one would want a gas gauge that waited until after the car had stopped to announce, "By the way, you are out of gas."

Unfortunately, many system administrators wait until *after* a computer has started to experience problems to begin monitoring its performance. When you discover that the computer has, say, run out of disk space, it is already too late to take corrective action that would have averted the problem in the first place. Had you been monitoring performance on a regular basis as part of your routine systems management procedures, you would have known in advance that disk space was beginning to run low. You would have been able to take steps to prevent the problem from occurring. Instead of using performance monitoring to *react* to problems once they occur, use *proactive* measures to ensure that the systems you are responsible for remain capable of delivering acceptable levels of performance at all times.

Tip Don't neglect performance monitoring until *after* something bad happens. Use *proactive* measures to find and correct potential performance problems before they occur.

Understanding the capabilities of the hardware and software that Windows Server 2003 manages is crucial to this goal. The computer and network hardware that you can acquire today are extremely powerful, but they still have a finite processing capacity. If the applications you are responsible for push hard against the capacity limits of the equipment you have installed, critical performance problems are likely to occur. You need to understand how to identify capacity constraints and what you can do about them when you encounter them. For example, it might be possible to upgrade to even more powerful hardware or to configure two or more machines into a *cluster* that can spread the work from a single machine over multiple ones. But to relieve a capacity constraint, you must first be able to identify it by having access to performance monitoring data that you can understand and know how to analyze.

The overall approach championed throughout this book favors developing proactive procedures to deal with potential performance and capacity problems in advance. This approach focuses on *continuous* performance monitoring, with regular reporting procedures to identify potential problems *before* they flare up and begin to have an impact on daily operations. This book provides detailed guidelines that will allow you to implement proven methods. These techniques include:

- *Baselining* and other forms of workload characterization
- *Stress testing* new applications and hardware configurations before deploying them on a widespread scale
- Establishing and reporting *service level objectives* based on the reasonable service expectations of your applications
- *Management by exception* to focus attention on the most serious performance-related problems
- *Trending and forecasting* to ensure that service level objectives can be met in the future

These are all proven techniques that can successfully scale to an enterprise level. This book will describe step-by-step procedures that you can implement—what data to collect, what alert thresholds to set, what statistical measures to report, and so on, so that you can establish an effective program of regular performance monitoring for your installation. Of course, as you grow more confident in your ability to analyze the performance statistics you gather, you will want to modify the sample procedures described here. Once you understand better what you are doing, you will be able to tailor these procedures to suit your environment better.

Diagnosing Performance Problems

Even with effective proactive monitoring procedures in place across your network of Windows Server 2003 machines, you can still expect occasional flare-ups that will call for immediate and effective troubleshooting. It is likely that no amount of proactive monitoring will eliminate the need for all performance troubleshooting. This book also emphasizes the practical tools, tips, and techniques that you will use to diagnose and solve common performance problems.

Wherever possible, this book tries to give you clear-cut advice and simple procedures to follow so that you can quickly diagnose and resolve many common performance problems. However, a simple cookbook approach to performance monitoring will take you only so far. Because the Windows Server 2003 systems, applications, and configurations you manage can be quite complex, the proactive performance procedures that you establish are subject to at least some of that complexity. (For more information about these performance procedures, see Chapter 4, “Performance Monitoring Procedures,” in this book.) Some of the following factors can complicate the performance monitoring procedures you implement:

- **Complex and expensive hardware configurations** Windows Server 2003 supports a wide range of environments, from simple 32-bit machines with a single processor, to more complex 64-bit machines with up to 512 GB of RAM and attached peripherals, to symmetric multiprocessing (SMP) architectures supporting up to 64 processors, and even specialized Non-Uniform Memory Access (NUMA) architecture machines. These advanced topics are discussed in detail in Chapter 6, "Advanced Performance Topics."
- **The number of systems that must be managed** Developing automated performance monitoring procedures that can scale across multiple machines and across multiple locations is inherently challenging. For complex environments, you might need even more powerful tools than those discussed in this book, including the Microsoft Operations Manager (MOM), a Microsoft product that provides comprehensive event management, proactive monitoring and alerting, and reporting and trend analysis for Windows Server System-based networks. For more information about MOM, see http://www.microsoft.com/mom/.
- **The complexity of the applications that run on Windows Server 2003** Some of the complex application environments your Windows Server 2003 machines must support include multi-user Terminal Services configurations, the .NET Framework of application run-time services, Microsoft Internet Information Services (IIS) Web server, the Microsoft SQL Server database management system, and the Microsoft Exchange Server messaging and collaboration server application. Each of these applications might require specialized procedures to gather and analyze performance data that is specific to these environments. In addition, these servers' applications can be clustered so that application processing is distributed across multiple server machines. Many of the application subsystems have specific configuration and tuning options that can have an impact on performance levels. In many instances, the application-specific knowledge to solve a specific SQL Server or Exchange performance problem is beyond the scope of the book. Where possible, other useful books, resources, and white papers that deal with Microsoft server application performance are referenced.

Each of these factors can add complexity to any performance problem diagnosis task. The best solutions to problems of this type are likely to be dependent on highly specific aspects of your configuration and workload. In addition to providing simple recipes for resolving common performance issues, this book also attempts to supply you with the basic knowledge and skills that will allow you to deal with more complex problems. As you gain confidence in the effectiveness of the methods and analytic techniques that are described here, you will learn to identify and resolve more difficult and more complex performance problems.

Overhead Considerations

One of the challenges of performance monitoring in the Windows Server 2003 environment is that the system configuration, the hardware, and the application software can be quite complex, as discussed. The challenge in complex environments is to collect the right amount of performance monitoring data so that you can diagnose and solve problems when they occur.

> **Caution** You must always be careful to ensure that the performance data you gather does not put so great a burden on the machine you are monitoring that you actually contribute to the performance problem you are trying to fix. You must also be careful to avoid collecting so much performance data that it greatly complicates the job of analyzing it.

These and other related considerations are aspects of the problem of performance monitoring *overhead*. By design, the performance monitoring procedures recommended here gather data that has a high degree of usefulness and a low impact on the performance of the underlying system. Nevertheless, it is not always possible for performance monitoring procedures to be both efficient *and* effective at diagnosing specific problems. The performance data you need to solve a problem might be voluminous as well as costly to gather and analyze. There are often difficult tradeoffs decisions that need to be made. You will always need to carefully assess the tradeoffs, and exercise good judgment about which data to collect and at what cost.

> **Important** In a crisis, overhead considerations pale beside the urgent need to troubleshoot a problem that is occurring. The normal rules about limiting the impact of performance monitoring do not apply in a crisis.

In Chapter 2, "Performance Monitoring Tools," in this book, the architecture of the main performance monitoring interfaces that Windows Server 2003 uses is discussed. The mechanisms built into Windows Server 2003 to capture performance statistics, gather them from different system components, and return them to various performance monitoring applications like the built-in Performance Monitor application will be described in detail. Once you understand how performance monitoring in Windows Server 2003 works, you should be able to make an informed decision about what costly performance data to gather and when it is justified to do so.

Crisis Mode Interventions

You are probably familiar with the crisis mode you and your Information Technology (IT) organization are plunged into when an urgent performance problem arises. When a Windows Server 2003 machine responsible for some *mission-critical* application misbehaves, alarms of various kinds start to spread through the IT technical support group. In the initial stages, there are likely to be many agitated callers to your organization's Help Desk function that services the user community. If the crisis is prolonged, established escalation procedures start to increase the visibility of the key role your department plays in maintaining a stable systems environment. Many self-appointed "experts" in this or that aspect of computer performance eventually convene to discuss the situation. Your efforts to resolve the problem quickly are suddenly thrust into the spotlight. Senior managers who never seemed very interested in your job function before are now anxious to hear a detailed account of your activities to solve the current crisis.

During a crisis, it is important that cooler heads prevail. Instead of jumping to a conclusion about what caused the current problem, begin by gathering and analyzing focused data about the problem. Windows Server 2003 includes many tools that are specifically designed to gather data to solve particular performance problems. Chapter 5, "Performance Troubleshooting," documents the use of many special purpose tools that you might only need to use in a crisis.

Normally, the performance monitoring procedures recommended in Chapter 5, "Performance Monitoring Procedures," are designed to gather performance data without having a major impact on the performance of the underlying system. In crisis mode, however, normal overhead considerations do not apply. If costly data gathering will potentially yield crucial information about a critical performance problem, the normal overhead considerations usually do not apply.

Scalability

Computer equipment today is extremely powerful, yet performance problems have not disappeared. The computing resources you have in place are finite. They have definite limits on their processing capability. *Scalability* concerns how those finite limitations impact performance. Computer professionals worry about scalability because in their experience, many computer systems encounter performance problems as the number of users of those systems grows. When computer professionals are discussing application or hardware scalability, they are concerned with root computer performance and capacity planning issues.

Figure 1-2 shows two scalability curves. The left-hand y-axis represents a measure of performance workload *throughput*—it could be database transactions per second, disk I/Os per second, Web visitors per hour, or e-mail messages processed per minute. The horizontal x-axis shows the growth in the number of users of this application.

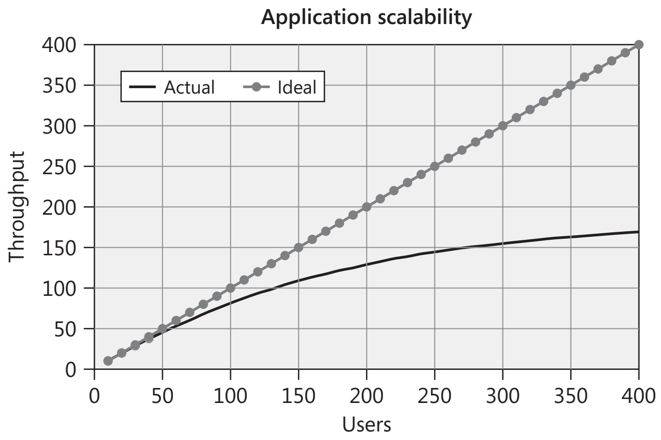


Figure 1-2 Ideal vs. actual application scalability

The "Ideal," or dotted, line is a straight line that shows performance increasing *linearly* as a function of the number of users. This is the ideal that computer engineers and designers strive for. As the number of concurrent users of an application grows, the user experience does not degrade because of elongated or erratic response times. The "Actual," or solid, line models the performance obstacles that an actual system encounters as the workload grows. Initially, the actual throughput curve diverges very little from the ideal case. But as the number of users grows, actual performance levels tend to be *nonlinear* with respect to the number of users. As more users are added and the system reaches its capacity limits, the throughput curve eventually plateaus, as illustrated in the figure.

Frequently, when computer applications are initially deployed, the number of users is quite small. Because the current system is not close to reaching its capacity limits, performance appears to scale linearly. But as users are added and usage of the application grows, performance problems are inevitably encountered. This is a core concern whenever you are planning for an application deployment that must accommodate a large number of users. Because computer hardware and software have finite processing limits, this nonlinear behavior—which is evidence of some form of performance degradation—can be expected at some point as the number of users grows. The focus of computer capacity planning, for instance, is to determine at what point, as the number of users grows, this performance degradation begins to interfere with the smooth operation of this application.

Inevitably, computer systems reach their capacity limits, and at that point, when more users are added, these systems no longer scale linearly. The focus of computer capacity planning for real-world workloads, of course, is to anticipate at what point serious performance degradation can be expected. After you understand the characteristics of your workload and the limitations of the computer environment in which it runs, you should be able to forecast the capacity limits of an application server.

In many instances, you can use stress-testing tools to simulate a growing workload until you encounter the capacity limits of your hardware. In simulated *benchmark* runs, using a stress-testing tool, from which the throughput curves in Figure 1-2 are drawn, the number of users is increased steadily until the telltale signs of nonlinear scalability appear. Stress testing your important applications to determine at what point serious performance degradation occurs is one effective approach to capacity planning. There are also analytic and modeling approaches to capacity planning that are effective. The mathematical relationships between key performance measurements, which are discussed in the next section of this chapter, form the basis for these analytic approaches.

For example, suppose you are able to measure the following:

- The current utilization of a potentially saturated resource like a processor, disk, or network adaptor
- The average individual user's resource demand that contributes to that resource's utilization
- The rate at which the number of application users is increasing

Using a simple formula called the Utilization Law, which is defined later in this chapter, you will be able to estimate the number of users necessary to drive the designated resource to its capacity limits, at which point that resource is bound to become a performance *bottleneck*. By both stress testing your application and using analytic modeling techniques, you can predict when the resource will reach its capacity limits. Once you understand the circumstances that could cause the resource to run out of capacity, you can formulate a strategy to cope with the problem in advance.

Caution Many published articles that discuss application scalability display graphs of performance levels that are reported as a function of an ever-increasing number of connected users, similar to Figure 1-2. These articles often compare two or more similar applications to show which has the better performance. They are apparently intended to provide capacity planning guidance, but unless the workload used in the tests matches your own, the results of these benchmarks might have little applicability to your own specific problems.

Experienced computer performance analysts understand that nonlinear scalability is to be expected when you reach the processing capacity at some bottlenecked resource. You can expect that computer performance will cease to scale linearly at some point as the number of users increases. As the system approaches its capacity limits, various performance statistics that measure the amount of work being performed tend to level off. Moreover, computer systems do not degrade gracefully. When a performance bottleneck develops, measures of application response time tend to increase very sharply. A slight increase in the amount of work that needs to be processed, which causes a very sharp increase in the response time of the application, is often evidence of a resource bottleneck. This nonlinear relationship between utilization and response time is also explored in the next section.

Being able to observe a capacity constraint that limits the performance of some real-world application as the load increases, as illustrated in Figure 1-2, is merely the starting point of computer performance analysis. Once you understand that a bottleneck is constraining the performance of the application, your analysis should proceed to identify the component of the application (or the hardware environment that the application runs in) that is the root cause of the constraint. This book provides guidance on how to perform a bottleneck analysis, a topic that is discussed in Chapter 5, "Performance Troubleshooting," but you should be prepared—this step might require considerable effort and skill.

After you find the bottleneck, you can then proceed to consider various steps that could relieve this capacity constraint on your system. This is also a step that might require considerable effort and skill. The alternatives you evaluate are likely to be very specific to the problem at hand. For example, if you determine that network capacity is a constraint on the performance of one of your important applications, you will need to consider practical approaches for reducing the application's network load, for example, compressing data before it is transmitted over the wire, or alternately, adding network bandwidth. You might also need to weigh both the potential cost and the benefits of the alternatives proposed before deciding on an effective course of action to remedy the problem. Some factors you might need to consider include:

- How long it will take to implement the change and bring some desperately needed relief to the situation
- How long the change will be effective, considering the current growth rate in the application's usage
- How to pay for the change, assuming there are additional costs involved in making the change (for example, additional hardware or software that must be procured)

Bottleneck analysis is a proven technique that can be applied to diagnose and resolve a wide variety of performance problems. Your success in using this technique depends on your ability to gather the relevant performance statistics you will need to understand where the bottleneck is. Effective performance monitoring procedures are a necessary first step. Understanding how to interpret the performance information you gathered is also quite important.

In benchmark runs, simulated users continue to be added to the system beyond the system's saturation point. Because these scalability articles report on the behavior of only simulated "users," they can safely ignore the impact on real customers and how these customers react to a computer system that has reached its capacity limits. In real life, system administrators must deal with dissatisfied customers who react harshly to erratic performance conditions. There might also be serious economic considerations associated with performance degradations. Workers who rely on computer systems to get their daily jobs done on time will lose productivity. Customers who rely on your applications might become so frustrated that they start to turn to your competitors for better service. When important business applications reach the limits of their scalability using current hardware and software, one of those crisis-mode interventions discussed earlier is likely to ensue.

Performance Monitoring Concepts

This section introduces the standard computer performance terminology that will be used in this book. Before you can apply the practices and procedures that are recommended in Chapter 4, "Performance Monitoring Procedures," it is a good idea to acquire some familiarity with these basic computer measurement concepts. By necessity, several mathematical formulas are introduced. These formulas are intended to illustrate the basic concepts used in computer performance analysis. Readers who are interested in a more formal mathematical presentation should consult any good computer science textbook on the subject.

Computers are electronic machines designed to perform calculations and other types of arithmetic and logical operations. The components of a computer system—its central processing unit (CPU) or processor, disks, network interface card, and so on—that actually perform the work are known generically as the computer's *resources*. Each resource has a finite *capacity* to perform designated types of work. *Customers* generate work requests for the server machine (or machines) to perform. In this book we are concerned primarily with Windows Server 2003 machines designed to service requests from multiple customers. In analyzing the performance of a particular computer system with a given workload, we need to measure the following:

- The *capacity* of those machines to perform this work
- The *rate* at which the machines are currently performing it
- The *time* it takes to complete specific tasks

The next section defines the terms that are commonly used to describe computer performance and capacity and describes how they are related to each other.

Definitions

Most computer performance problems can be analyzed in terms of resources, queues, service requests, and response time. This section defines these basic performance measurement concepts. It describes what they mean and how they are related.

Two of the key measures of computer capacity are bandwidth and throughput. *Bandwidth* is a measure of capacity, which is the rate at which work can be completed, whereas throughput measures the actual rate at which work requests are completed. *Scalability*, as discussed in the previous section, is often defined as the throughput of the machine or device as a function of the total number of users requesting service. How busy the various *resources* of a computer system get is known as their *utilization*. How much work each resource can process at its maximum level of utilization is defined as its *capacity*.

The key measures of the time it takes to perform specific tasks are queue time, service time, and response time. The term *latency* is often used in an engineering context to refer to either service time or response time. *Response time* will be used consistently here to refer to the sum of service time and queue time. In networks, another key measure is *round trip time*, which is the amount of time it takes to send a message and receive a confirmation message (called an *Acknowledgement*, or *ACK* for short) in reply.

When a work request arrives at a busy resource and cannot be serviced immediately, the request is queued. Queued requests are subject to a *queue time* delay before they are serviced. The number of requests that are delayed waiting for service is known as the *queue length*.

Note The way terms like response time, service time, and queue time are defined here is consistent with the way these same terms are defined and used in Queuing Theory, which is a formal, mathematical approach used widely in computer performance analysis.

Elements of a Queuing System

Figure 1-3 illustrates the elements of a simple *queuing system*. It depicts customer requests arriving at a server for processing. This example illustrates customer requests for service arriving intermittently. The customer requests are for different amounts of service. (The service request *arrival rate* and service time distributions are both *non-uniform*.) The server in the figure could be a processor, a disk, or a network interface card (NIC). If the device is free when the request arrives, it goes into service immediately. If the device is already busy servicing some previous request, the request is queued. *Service time* refers to the time spent at the device while the request is being processed. Queue time represents the time spent waiting in the queue until the server becomes available. Response time is the sum of both service time and queue time. How busy the server gets is its *utilization*.

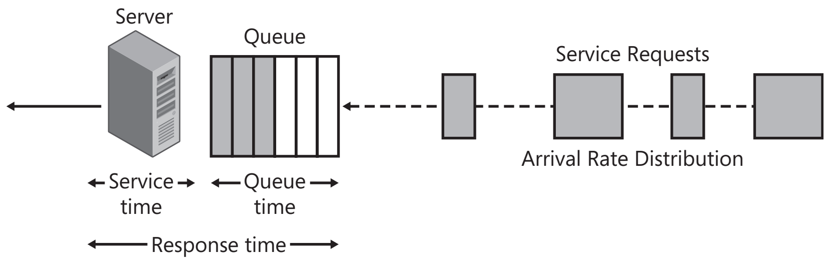


Figure 1-3 The elements of a queuing system

The computer resource and its queue of service requests depicted in Figure 1-3 leads to a set of mathematical formulas that can characterize the performance of this queuing system. Some of these basic formulas in queuing theory are described later. Of course, this model is too simple. Real computer systems are much more complicated. They have many resources, not only one, that are interconnected. At a minimum, you might want to depict some of these additional resources, including the processor, one or more disks, and the network interface cards. Conceptually, these additional components can be linked together in a network of queues. Computer scientists can successfully model the performance of complex computer systems using *queuing networks* such as the one depicted in Figure 1-4. When specified in sufficient detail, queuing networks, similar to the one illustrated, can model the performance of complex computer systems with great accuracy.

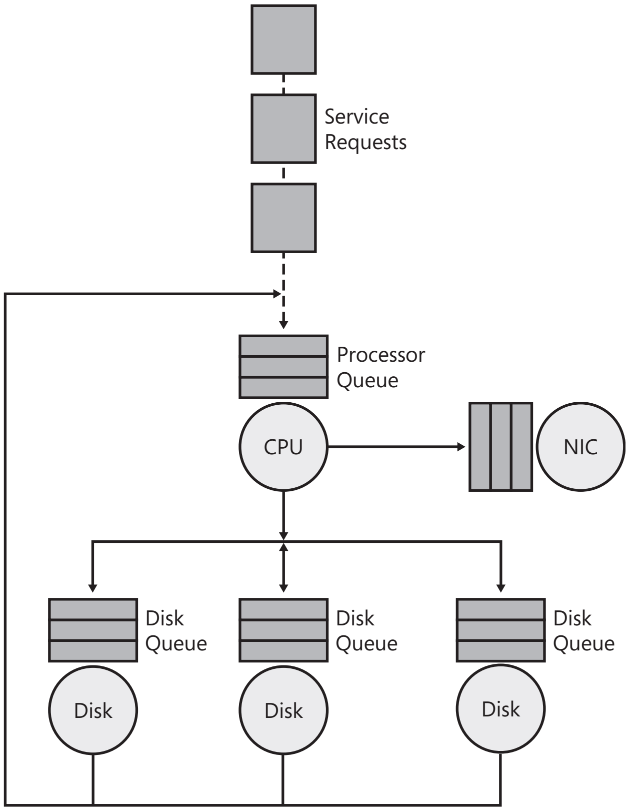


Figure 1-4 A network of queues

Note Not all the hardware resources necessary for a computer system to function are easily represented in a simple queuing model like the one depicted in Figure 1-4. The memory that a computer uses is one notable resource missing from this simple queuing model. Physical memory, or RAM, is not utilized in quite the same way as other resources like CPUs and disks. Cache buffering is another important element that might not be easy to characterize mathematically. Computer scientists use much more complex queuing models to represent a complex machine environment and all its critical resources accurately. For example, virtual memory overflowing to the paging file is usually represented indirectly as an additional disk I/O workload. If an element is important to performance, computer scientists usually find a way to represent it mathematically, but those more complex representations are beyond the scope of this chapter.

Bandwidth

Bandwidth measures the capacity of a link, bus, channel, interface, or the device itself to transfer data. Bandwidth is usually measured in either bits/second or bytes/second (where there are 8 bits in a data byte). For example, the bandwidth of a 10BaseT Ethernet connection is 10 *megabits per second* (*Mbps*), the bandwidth of an ultra SCSI disk is 160 *megabytes per second* (*MBps*), and the bandwidth of the PCI-X 64-bit 100 megahertz (MHz) bus is 800 MBps.

Bandwidth usually refers to the maximum theoretical data transfer rate of a device under ideal operating conditions. Therefore, it is an *upper-bound* on actual performance. You are seldom able to measure a device actually performing at its full rated bandwidth. Devices cannot reach their advertised performance level because *overhead* is often associated with servicing work requests. For example, you can expect operating system overhead, protocol message processing time, and a delay in disk positioning to absorb some of the available bandwidth for each request to read or write a disk. These overhead factors mean that the application can seldom use the full rated bandwidth of a disk for data transfer. As another example, various overheads associated with network communication protocols reduce the theoretical capacity of a 100 Mbps Fast Ethernet link to significantly less than 10 MBps. Consequently, discussing *effective bandwidth* or *effective capacity*—the amount of work that can be accomplished using the device under real-world conditions—is usually more realistic.

Throughput

Throughput measures the rate that work requests are completed, from the point of view of some observer. Examples of throughput measurements include the number of reads per second from the disk or file system, the number of instructions per second executed by the processor, HTTP requests processed by a Web server, and transactions per second that can be processed by a database engine.

Throughput and bandwidth are very similar. Bandwidth is often construed as the maximum *capacity* of the system to perform work, whereas throughput is the current *observed rate* at which that work is being performed.

Utilization

Utilization measures the fraction of time that a device is *busy* servicing requests, usually reported as a percent busy. Utilization of a device varies from 0 through 1, where 0 is idle and 1 (or 100 percent) represents utilization of the full bandwidth of the device. It is customary to report that the processor or CPU is 75 percent busy, or the disk is 40 percent busy. It is not possible for a single device to ever be greater than 100 percent busy.

Measures of resource utilization are common in Windows Server 2003. Later in this chapter, many of the specific resource utilization measurements that you are able to gather on your Windows Server 2003 machines will be described. You can easily find out how busy the processors, disks, and network adaptors are on your machines. You will also see how these utilization measurements are derived by the operating system, often using indirect measurement techniques that save on overhead. Knowing how certain resource utilization measurements are derived will help you understand how to interpret them.

Monitoring the utilization of various hardware components is an important element of any capacity planning exercise. If an application server is currently processing 60 transactions per second with a CPU utilization measured at 20 percent, the server apparently has considerable reserve capacity to process transactions at an even higher rate. On the other hand, a server processing 60 transactions per second running at a CPU utilization of 98 percent is operating at or near its maximum capacity.

In forecasting your future capacity requirements based on current performance levels, understanding the resource profile of workload requests is very important. If you are monitoring an IIS Web server, for example, and you measure processor utilization at 20 percent busy and the transaction rate at 50 HTTP GET Requests per second, it is easy to see how you might create the capacity forecast shown in Table 1-1.

Table 1-1 Forecasting Linear Growth in Processor Utilization as a Function of the Service Request Arrival Rate

HTTP GET Requests/Sec	% Processor Time
50	20%
100	40%
150	60%
200	80%
250	100%

The measurements you took and the analysis you performed enabled you to anticipate that having to process 250 HTTP GET Requests per second at this Web site would exhaust the current processor capacity. This conclusion should then lead you to start tracking the growth of your workload, with the idea of recommending additional processor capacity as the GET Request rate approaches 200 per second, for example.

You have just executed a simple capacity plan designed to cope with the scalability limitations of the current computer hardware environment for this workload. Unfortunately, computer capacity planning is rarely quite so simple. For example, Web transactions use other resources besides the CPU, and one of those other resources might reach its effective capacity limits long before the CPU becomes saturated.

And there are other complicating factors. One operating assumption in this simple forecast is that processing one HTTP GET Request every second in this environment requires 0.4 percent processor utilization, on average. This assumption is based on your empirical observation of the current system. Other implicit assumptions in this approach include:

- Processor utilization is a simple, linear function of the number of HTTP GET Requests being processed
- The service time distribution for processor at the processor per HTTP GET Request—the amount of processor utilization per request—remains constant

Unfortunately, these implicit assumptions might not hold true as the workload grows. Because of caching effects, for example, the amount of processor time per request might vary as the workload grows. If the caching is very effective, the amount of processor time per request could decrease. If the caching loses effectiveness as the workload grows, the average amount of processor time consumed per request might increase. You will need to continue to monitor this system as it grows to see which of these cases holds.

The component functioning as the constraining factor on throughput—in this case, the processor when 250 HTTP GET Requests per second are being processed—is designated as the *bottlenecked device*. If you improve performance at the bottlenecked device—by upgrading to a faster component, for example—you are usually able to extend the effective capacity of the computer system to perform more work.

Tip Measuring utilization is often very useful in detecting system bottlenecks. Bottlenecks are usually associated with processing constraints at some overloaded device. It is usually safe to assume that devices observed operating at or near their 100 percent utilization limits are bottlenecks, although things are not always that simple, as discussed later in this chapter.

It is not always easy to identify the bottlenecked device in a complex computer system or a network of computer systems. For example, 80 or 90 percent utilization is not necessarily the target threshold for all devices. Some computer equipment like disk drives perform more efficiently under heavier loads. These and other anomalies make the straight-line projections shown in Table 1-1 prone to error if *load-dependent servers* are involved.

Service Time

Service time measures how long it takes to process a specific customer work request. Engineers alternatively often speak of the length of time to process a request as the device's *latency*, which is another word for delay. For example, memory latency measures the amount of time the processor takes to fetch data or instructions from RAM or one of its internal memory caches. Other related measures of service time are the *turnaround time* for requests, usually ascribed to longer running tasks such as disk-to-tape backup runs. The *round trip time* is an important measure of network latency because when a request is sent to a destination across a communications link using the Transmission Control Protocol/Internet Protocol (TCP/IP), the sender must wait for a reply.

The service time of a file system request, for example, will vary based on whether the request is cached in memory or requires a physical disk operation. The service time will also vary according to whether it is a sequential read of the disk, a random read of the disk, or a write operation. The expected service time of the physical disk request also varies depending on the block size of the request. These workload dependencies demand that you measure disk service time directly instead of rely on projections that are based on some idealized model of disk performance.

The service time for a work request is generally assumed to be constant, a simple function of the device's speed or its capacity. Though this is largely true, under certain circumstances, device service times can vary as a function of utilization. Using intelligent scheduling algorithms, it is often possible for processors and disks to work more efficiently at higher utilization rates. You are able to observe noticeably better service times for these devices when they are more heavily utilized. Some aspects of these intelligent scheduling algorithms are described in greater detail later in this chapter.

The service time spent processing an ASP.NET Web-based application request can be broken down into numerous processing components—for example, time spent in the application program, time spent during processing by .NET Framework components, time spent in the operating system, and time spent in database processing. For each one of these subcomponents, the application service time can be further decomposed into time spent at various hardware components, for example, the CPU, the disk, and the network. *Decomposition* is an important technique used in computer performance analysis to relate a workload to its various hardware and software processing components. To decompose application service times into their component parts, you must understand how busy various hardware components are and, specifically, how workloads contribute to that utilization. This can be very challenging for many Windows Server 2003 transaction processing applications because of their complexity. You will need to gather detailed trace data to accurately map all the resources used by applications to the component parts of individual transactions.

Response Time

Response time is the sum of service time and queue time:

response time = service time + queue time

Mathematically, this formula is usually represented as follows:

$W = W_s + W_q$

where W is latency, W_s is the service time, and W_q is the queue time.

Response time includes both the device latency and any queuing delays that accrue while the request is queued waiting for the device. At heavily utilized devices, queue time is likely to represent a disproportionate amount of the observed response time. Queue time is discussed in greater detail in the next section.

Transaction response time also refers to the amount of time it takes to perform some unit of work, which can further be decomposed into the time spent using (and sometimes waiting to use) various components of a computer system. Because they best encapsulate the customer's experience interacting with an application hosted on a Windows Server 2003 machine, measures of application response time are among the most important measures in computer performance and capacity planning. Wherever possible, management reports detailing application response times are preferable to reports showing the utilization of computer resources or their service times.

Queue Time

When a work request arrives at a busy resource and cannot be serviced immediately, the request is queued. Requests are subject to a *queue time* delay once they begin to wait in a queue before being serviced.

Queue time arises in a multi-user computer system like Windows Server 2003 because important computer resources are *shared*. Shared resources include the processor, the disks, and network adaptors. This sharing of devices is orchestrated by the operating system using locking structures in a way that is largely transparent to the individual programs you are running. The operating system guarantees the integrity of shared resources like the processor, disks, and network interfaces by ensuring that contending applications can access them only *serially*, or one at a time. One of the major advantages of a multi-user operating system like Windows Server 2003 is that resources can be shared safely among multiple users.

When a work request to access a shared resource that is already busy servicing another request occurs, the operating system queues the request and *queue time* begins to accumulate. The one aspect of sharing resources that is not totally transparent to programs executing under Windows Server 2003 is the potential performance impact of resource sharing. Queuing delays occur because shared resources have multiple applications attempting to access these resources in parallel. Significant delays at a constrained resource are apt to become visible. If there is significant contention for a shared resource because two or more programs are attempting to use it at the same time, performance might suffer. When there is a performance problem on a local Windows workstation, only one user suffers. When there is a performance problem on a Windows Server 2003 application server, a multitude of computer users can be affected.

On a very heavily utilized component of a system, queue time can become a very significant source of delay. It is not uncommon for queue time delays to be longer than the amount of time actually spent receiving service at the device. No doubt, you can relate to many real-world experiences where queue time is significantly greater than service time. Consider the time you spend waiting in line in your car at a tollbooth. The amount of time it takes you to pay the toll is often insignificant compared to the time you spend waiting in line. The amount of time spent waiting in line to have your order taken and filled at a fast-food restaurant during the busy lunchtime period is often significantly longer than the time it takes to process your order. Similarly, queuing delays at an especially busy shared computer resource can be prolonged. It is important to monitor the queues at shared resources closely to identify periods when excessive queue time delays are occurring.

Important Measurements of either the queue time at a shared resource or the queue length are some of the most important indicators of performance you will encounter.

Queue time can be difficult to measure directly without adding excessive measurement overhead. Direct measurements of queue time are not necessary if both the service time and the queue depth (or queue length) can be measured reliably and accurately. If you know the queue length at a device and the average service time, the queue time delay can be estimated reliably, as follows:

queue time = *average queue length* × *average service time*

This simple formula reflects the fact that any queued request must wait for the request currently being serviced to complete.

Actually, this formula overestimates queue time slightly. On *average*, the queue time of the first request in the queue is only one half of the service time. Subsequent requests that arrive and find the device busy and at least one other request already in the queue are then forced to wait. Therefore, a better formula is:

queue time = ((queue length–1) × average service time) + (average service time÷2)

Of course, service time is not always so easy to measure either. However, service time can often be computed using the Utilization Law in cases where it cannot be measured directly but the device utilization and arrival rate of requests are known.

Not all computer resources are shared on Windows Server 2003, which means that these unshared devices have no queuing time delays. Input devices like the mouse and keyboard, for example, are managed by the operating system so that they are accessible by only one application at a time. These devices are *buffered* to match the speed of the people operating them because they are capable of generating interrupts faster than the application with the current input focus can process their requests. Instead of queuing these requests, however, the operating system device driver routines for the keyboard and mouse discard extraneous interrupts. The effect is that little or no queue time delay is associated with these devices.

Bottlenecks

One of the most effective methods used to tune performance is systematically to identify bottlenecked resources and then work to remove or relieve them. When the throughput of a particular system reaches its effective capacity limits, the system is said to be *bottlenecked*. The resource bottleneck is the component that is functioning at its capacity limit. The bottlenecked resource can also be understood as the resource with the fastest growing queue as the number of users increases.

Important Empirically, you can identify the bottlenecked resource that serves to constrain system scalability as the resource that saturates first or the one with the fastest growing queue. The goal of performance tuning is to create a balanced system with no single bottlenecked resource in evidence. A *balanced system* is one in which no resource saturates before any other as the load increases, and all resource queues grow at the same rate. In a balanced system, queue time delays are minimized across all resources, leading to performance that is optimal for a given configuration and workload.

Understanding that a computer system is operating at the capacity limit of one of its components is important to know. It means, for example, that no amount of tweaking the tuning parameters is going to overcome the capacity constraint and allow the system to perform more work. You need more capacity, and any other resolution short of providing some capacity relief is bound to fall short!

Once you identify a bottlenecked resource, you should follow a systematic approach to relieve that limit on performance and permit more work to get done. You might consider these approaches, for example:

- Optimizing the application so that it runs more efficiently (that is, utilizes less bandwidth) against the specific resource
- Upgrading the component of the system that is functioning at or near its effective bandwidth limits so that it runs faster
- Balancing the application across multiple resources by adding more processors, disks, network segments, and so on, and processing it in parallel

Possibly, none of these alternatives for relieving a capacity constraint will succeed in fixing the problem quick enough to satisfy your users. In these cases, it might be worthwhile to resort to tweaking this or that system or application tuning parameter to provide some short-term relief. The most important settings for influencing system and application performance in Windows Server 2003 are discussed in Chapter 6, "Advanced Performance Topics." There are also many run-time settings associated with applications such as Exchange or Internet Information Services (IIS) that can impact performance. Many application-oriented optimizations are documented in other Resource Kit publications or in white papers available at http://www.microsoft.com.

A number of highly effective performance optimizations are built into Windows Server 2003. These settings are automatic, but there might be additional adjustments that are worthwhile for you to consider making manually. These manual adjustments are discussed in Chapter 6, "Advanced Performance Topics." Some of the built-in performance optimizations employ intelligent scheduling algorithms. Keep in mind that scheduling algorithms have a better opportunity to improve performance only when enough queued requests are outstanding that it makes a difference which request the operating system schedules next. Consequently, the busier the resource is, the greater the impact on performance these performance optimizations have. The next section explains how these scheduling algorithms work.

Managing Queues for Optimal Performance

If multiple requests are waiting in a queue, the *queuing discipline* is what determines which request is serviced next. Most queues that humans occupy when they are waiting for service are governed by the principle of *fairness*. A fair method of ordering the queue is First Come, First Serve (FCFS). This is also known as a FIFO queue, which stands for First In, First Out. This principle governs how you find yourself waiting in a bank line to cash a check, for example. FIFO is considered fair because no request that arrives *after* another can be serviced before requests that arrived earlier are themselves satisfied. *Round robin* is another fair scheduling algorithm where customers take turns receiving service.

Unfair scheduling algorithms Fair scheduling policies do not always provide optimal performance. For performance reasons, the Windows Server 2003 operating system does not always use fair scheduling policies for the resource queues it is responsible for managing. Where appropriate, Windows Server 2003 uses *unfair scheduling* policies that can produce better results at a heavily loaded device. The unfair scheduling algorithms that are implemented make it possible for devices such as processors and disks to work more efficiently under heavier loads, for example.

Priority queuing with preemptive scheduling Certain work requests are regarded as higher priority than others. If both high priority and low priority requests are waiting in the queue, it makes sense for the operating system to schedule the higher priority work first. On Windows Server 2003, queued requests waiting for the processor are ordered by priority, with higher priority work taking precedence over lower priority work.

The priority queuing scheme used to manage the processor queue in Windows Server 2003 has at least one additional feature worth considering here. The processor hardware is also used to service high priority *interrupts* from devices. Devices such as disks interrupt the processor to signal that an I/O request that was initiated earlier is now complete. When a device interrupt occurs, the processor stops executing the current program thread and begins to immediately service the device that generated the higher priority interrupt. (The interrupted program thread is queued and rescheduled to resume execution after the interrupt is serviced.) Higher priority work that is scheduled to run immediately and interrupts a lower priority thread that is already running is called *preemptive scheduling*. Windows Server 2003 uses both priority queuing and preemptive scheduling to manage the system's processor queue. The priority queuing scheme used by the operating system to manage the processor queue is reviewed in more detail later in this chapter.

Priority queuing has a well-known side effect that becomes apparent when a resource is very heavily utilized. If there are enough higher priority work requests to saturate the processor, lower priority requests might get very little service. This is known as *starvation*. When a resource is saturated, priority queuing ensures that higher priority work receives preferred treatment, but lower priority work can suffer from starvation. Lower priority work could remain delayed in the queue, receiving little or no service for extended periods. The resource utilization measurements that are available on Windows Server 2003 for the processor allow you to assess whether the processor is saturated, what work is being performed at different priority levels, and whether low priority tasks are suffering from starvation.

Serving shorter requests first An important result of Queuing Theory is that when queued requests can be sorted according to the amount of service time that will be needed to complete the request, higher throughput is achieved if the shorter work requests are serviced first. This is the same sort of optimization that supermarkets use when they have shoppers sort themselves into two sets of queues based on the number of items in their shopping carts. In scheduling work at the processor, for example, this sorting needs to be done based on the *expected* service time of the request, because the actual duration of a service request is not known in advance. Windows Server 2003 implements a form of dynamic sorting that boosts the priority of processor service requests that are expected to be short and reduces the priority of requests that are expected to take longer. Another situation in which queued requests are ordered by the shortest service time first is when Serial ATA or SCSI disks are enabled for tagged command queuing.

For all the benefit these intelligent scheduling algorithms confer, it is important to realize that reordering the device queue can have a significant performance impact only when there is a long queue of work requests that can be rearranged. Computer systems on which these scheduling algorithms are most beneficial have a component that is saturated for an extended period of time, allowing lengthy lines of queued requests to build up which can then be sorted. Such a system is by definition out of capacity. Reducing the queue depth at the bottlenecked device by adding capacity, for example, is a better long-term solution. You should configure machines with sufficient capacity to service normal peak loads so that lengthy queues that can be sorted optimally are the exception, not the rule. While intelligent scheduling at a saturated device can provide some relief during periods of exceptional load, its effectiveness should never divert your attention from the underlying problem, which is a shortage of capacity at the resource where the queue is being manipulated so favorably.

Bottleneck Analysis

To make the best planning decisions, a traditional approach is to try and understand hardware *speeds and feeds*—how fast different pieces of equipment are capable of running. This approach, however, is much more difficult than it sounds. For example, it certainly sounds like a SCSI disk attached to a 20-MBps SCSI-2 adapter card would run much slower than one attached to an 80-MBps UltraSCSI-3 adapter card. UltraSCSI-3 sounds like it should beat an older SCSI-2 configuration every time. But the fact is that there might be little or no practical difference in the performance of the two configurations. One reason there might be no difference is because the disk might transfer data only at 20 MBps anyway, so the extra capacity of the UltraSCSI-3 bus is never being utilized.

Important A complex system can run only as fast as its slowest component. This is the principle that underlies the technique of bottleneck analysis.

The principle that a complex system will run only as fast as its slowest component forms the basis for a very useful analysis technique called *bottleneck analysis*. The slowest device in a configuration is often the weakest link. Find it and replace it with a faster component, and you have a good chance of improving performance. Replacing some component *other* than the bottleneck device with a faster component will not appreciably improve performance. This theory sounds good, of course, but you probably noticed that the rule does not tell you *how* to go about finding this component. Given the complexity of many modern computer networks, this seemingly simple task is actually quite complicated.

In both theory and practice, performance tuning is the process of locating the bottleneck in a configuration and removing it—somehow. The system's performance will improve until the next bottleneck manifests, which you can then identify and remove. Easing a bottleneck for an overloaded resource usually entails replacing it with a newer, faster version of the same component. For example, if network bandwidth is a constraint on performance, upgrade the configuration from 10 Mb Ethernet to 100 Mb Fast Ethernet. If the network actually is the bottleneck, performance should improve.

A system in which all the bottlenecks have been removed can be said to be a *balanced* system. All the components in a balanced system are at least capable of handling the flow of work from component to component without excessive delays building up at any one particular component. For a moment, think of the network of computing components where work flows from one component (the CPU) to another (the disk), back again, then to another (the network) and back again to the CPU, as depicted in Figure 1-5. When different workload processing components are evenly distributed across the hardware devoted to doing the processing, that system is balanced.

You can visualize a balanced system (and not one that is simply over-configured) as one in which workload components are evenly distributed across the processing resources. If there are delays, the work that is waiting to be processed is also evenly distributed in the system. Work that is evenly distributed around the system waiting to be processed is illustrated in Figure 1-5. Suppose you could crank up the rate at which requests arrive to be serviced. (Think of SQL Server requests to a Windows Server 2003 database, for example, or logon requests to an Active Directory authentication server.) If the system is balanced, you will observe that work waiting to be processed remains evenly distributed across system components, as shown in Figure 1-5.

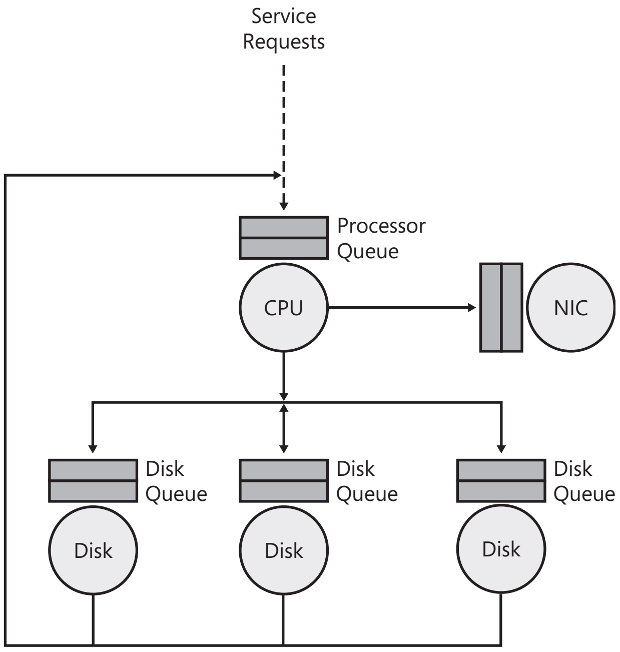


Figure 1-5 A balanced system

If instead you observe something like what is depicted in Figure 1-6, where many more requests are waiting behind just one of the disks, you have identified with some authority the component that is the bottleneck in the configuration. When work backs up behind a bottlenecked device, delays there can cascade, causing delays to build up elsewhere in the configuration. Because the manner in which work flows through the system might be complicated, a bottlenecked resource can impact processing at other components in unexpected ways. Empirically, it is sufficient to observe that work accumulates behind the bottlenecked device *at the fastest rate* as the workload rate increases. Replacing this component with a faster processing component should improve the rate that work that can flow through the entire system.

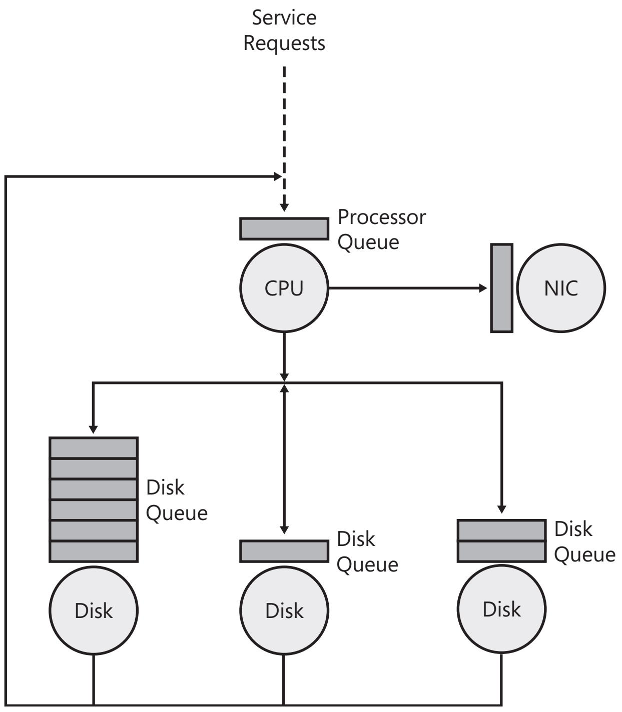


Figure 1-6 A bottlenecked system

Utilization Law

The utilization of a device is the product of the observed rate in which requests are processed and the service time of those requests, as follows:

utilization = *completion rate* × *service time*

This simple formula relating device utilization, the request completion rate (or throughput), and the service time is known as the Utilization Law. Service time is often difficult to measure, but the Utilization Law makes it possible to measure the throughput rate and the utilization of a disk, for example, and derive the disk service time. A disk that processes 30 input/output (I/O) operations per second with an average service time of 10 milliseconds is busy processing requests 30 × 0.010 sec = 300 milliseconds of utilization every second, or 30 percent busy.

Utilization, by definition, is limited to 0–100 percent device-busy. If it is not possible for a device to be more than 100 percent utilized, what happens when requests arrive at a device faster than they can be processed? The answer, of course, is that requests for service that arrive faster than they can be serviced must be queued. A related question—the relationship between queue time and utilization—is explored later in this chapter.

There is no substitute for direct measurements of device or application throughput, service time, and utilization. But the Utilization Law allows you to measure two of the three terms and then derive the remaining one. Many of the device utilization measurements reported in Windows Server 2003 are derived by taking advantage of the Utilization Law.

Queue Time and Utilization

If a request A arrives at an idle resource, request A is serviced immediately. If the resource is already busy servicing request B when request A arrives, request A is queued for service, forced to wait until the resource becomes free. It should be apparent that the busier a resource gets, the more likely a new request will encounter a busy device and be forced to wait in a queue. This relationship between utilization and queue time needs to be investigated further. The insights revealed by a branch of mathematics known as Queuing Theory can shed light on this interesting relationship.

Queuing Theory is a branch of applied mathematics that is widely used to model computer system performance. It can be used, for example, to predict how queue time might behave under load. Only very simple queuing models will be discussed here. These simple models relate the following:

- Server utilization (resources are termed *servers*)
- Rate at which work requests arrive
- Service time of those requests
- Amount of queue time that can be expected as the load on the resource varies

Caution These simple models do not represent reality that closely and are used mainly because they are easy to calculate. However, experience shows that these simple queuing models can be very useful in explaining how many of the computer performance measurements you will encounter behave—albeit up to a point.

You need to understand some of the important ways these simple models fail to reflect the reality of complex computer systems so that you are able to use these mathematical insights wisely. Simple queuing models, as depicted in Figure 1-3, are characterized by three elements: the arrival rate of requests, the service time of those requests, and the number of servers to service those requests. If those three components can be measured, simple formulas can be used to calculate other interesting metrics. Both the queue length and the amount of queue time that requests are delayed while waiting for service can be calculated using a simple formula known as Little's Law. Queue time and service time, of course, can then be added together to form response time, which is usually the information you are most interested in deriving.

Arrival Rate Distribution

To put the mathematics of simple Queuing Theory to work, it is necessary to know both the average rate that requests arrive and the distribution of arrivals around the average value. The arrival rate distribution describes whether requests are spaced out evenly (or uniformly) over the measurement interval or whether they tend to be bunched together, or bursty. When you lack precise measurement data on the arrival rate distribution, it is usually necessary to assume that the distribution is bursty (or random). A random arrival rate distribution is often a reasonable assumption, especially if many independent customers are generating the requests. A large population of users of an Internet e-business Web site, for example, is apt to generate a randomly distributed arrival rate. Similarly, a Microsoft Exchange Server servicing the e-mail requests of employees from a large organization is also likely to approximate a randomly distributed arrival rate.

But it is important to be careful. The independence assumption can also be a very poor one, especially when the number of customers is very small. Consider, for example, a disk device with only one customer, such as a back-up process or a virus scan. Instead of having random arrivals, a disk back-up process schedules requests to disk one after another in a serial fashion. A program execution thread from a back-up program issuing disk I/O requests will generally not release another I/O request until the previous one completes. When requests are scheduled in this fashion, it is possible for a single program to drive the disk to virtually 100 percent utilization levels without incurring any queuing delay. Most throughput-oriented disk processes routinely violate the independence assumption behind simple queuing models because they schedule requests to the disk serially. As a consequence, a simple queuing model of disk performance is likely to seriously overestimate the queue time at a disk device being accessed by a few customers that are scheduling their requests serially.

Service Time Distribution

It is also necessary to understand both the average service time for requests and the distribution of those service times around the average value. Again, lacking precise measurement data, it is simple to assume that the service time distribution is also random. It will also be useful to compare and contrast the case where the service time is relatively constant, or uniform.

The two simple cases illustrated in Figure 1-7 are denoted officially as M/M/1 and M/D/1 queuing models. The standard notation identifies the following:

arrival rate distribution/service time distribution/number of servers

where *M* is an *exponential*, or random distribution; *D* is a uniform distribution; and *1* is the number of servers (resources).

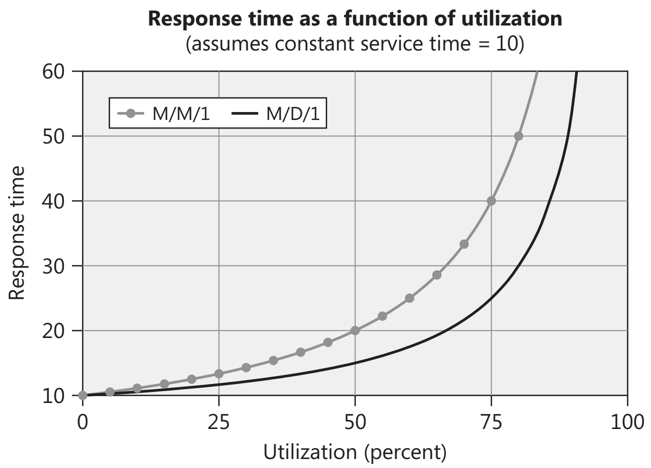


Figure 1-7 Graphing response time as a function of utilization

Both curves in Figure 1-7 show that the response time of a request increases sharply as the server utilization increases. Because these simple models assume that service time remains constant under load (not always a valid assumption), the increase in response time is a result solely of increases in the request queue time. When device utilization is relatively low, the response time curve remains reasonably flat. But by the time the device reaches 50 percent utilization, in the case of M/M/1, the average queue length is approximately 1. At 50 percent utilization in an M/M/1 model, the amount of queue time that requests encounter is equal to the service time. To put it another way, at approximately 50 percent busy, you can expect that queuing delays lead to response times that are *double* the amount of time spent actually servicing the request. Above 50 percent utilization, queue time increases even faster, and more and more queue time delays accumulate. This is an example of an exponential curve, where queue time (and response time) is a nonlinear function of utilization. As resources saturate, queue time comes to dominate the application response time that customers experience.

The case of M/D/1 shows queue time for a uniform service time distribution that is exactly 50 percent of an M/M/1 random service time distribution with the same average service time. Reducing the variability of the service time distribution works to reduce queue time delays. Many tuning strategies exploit this fact. If work requests can be scheduled in a way to create a more uniform service time distribution, queue time—and response time—are significantly reduced. That is why supermarkets, for example, separate customers into two or three sets of lines based on the number of items in their shopping carts. This smoothes out the service time distribution in the supermarket checkout line and reduces the overall average queue time for shoppers that are waiting their turn.

Queue Depth Limits

One other detail of the modeling results just discussed should be examined. Technically, these results are for *open* network queuing models that assume the average arrival rate of new requests remains constant no matter how many requests are backed up in any of the queues. Mathematically, open queuing models assume that the arrival rate of requests is sampled from an infinite population. This is an assumption that is made to keep the mathematics simple. The formula used to derive the queue time from utilization and service time for an M/M/1 model in Figure 1-7 is shown here:

$$W_q = (W_s \times u) / (1 - u)$$

In this formula, *u* is the device utilization, W_s is the service time, and W_q is the queue time.

The corresponding formula for an M/D/1 model where the service time distribution is uniform is this:

$$W_q = (W_s \times u) / (1 - u) / 2$$

The practical problem with this simplifying assumption is that it predicts the queue length growing at a faster rate than you are likely to observe in practice. Mathematically, both these formulas show extreme behavior as the utilization, *u*, approaches 100 percent. (As *u* approaches 1, the denominator in both formulas approaches 0.) This produces a right-hand tail to the queue length distribution that rises hyper exponentially to infinity as a resource saturates.

In practice, when a bottleneck is evident, the arrival rate of new requests slows down as more and more customers get stuck in the system, waiting for service. It takes a more complicated mathematical approach to model this reality. *Closed* network queuing models are designed to reflect this behavior. In a closed model, there is an upper limit on the customer population. Examples of such an upper limit in real life include the number of TCP sessions established that sets an upper limit on the number of network requests that can be queued for service. In dealing with the processor queue, there is an upper limit on the queue depth based on the number of processing threads in the system eligible to execute. This upper limit on the size of the population creates a practical limit on maximum queue depth. If 200 processing threads are defined, for example, it is impossible for the processor queue depth (representing threads that are queued for service) to exceed 199. The practical limit on the processor queue depth is even lower. Because many processing threads are typically idle, a more practical upper limit on the processor queue depth you are likely to observe is the number of processing threads that are eligible to run, that is, those threads not in a voluntary Wait state. The number of processing threads that are currently eligible to run, in fact, sets a practical upper limit on the processor queue depth. Once every thread that is eligible to run is stuck in the processor queue, jammed up behind a runaway thread in a high priority loop, for example, no new requests for processor service are generated.

Though closed network queuing models are much more capable of modeling reality than the simple open models, they lead to sets of simultaneous equations that need to be solved, and the mathematics is beyond the scope of this chapter. You can learn more about these equations and the techniques for solving them in any good computer science textbook on the subject. The point of this discussion is to provide some advice on when to use the simple equations presented here. The simple formulas for M/M/1 and M/D/1 open queuing models suffice for many capacity planning exercises that do not require a precise solution. However, they break down when there is a saturated resource. Up until the point when server utilization starts to approach 100 percent, the simple open models generally do a reasonably good job of predicting the behavior you can observe in practice.

Little's Law

A mathematical formula known as Little's Law relates response time and utilization. In its simplest form, Little's Law expresses an equivalence relation between response time (W), the arrival rate (λ), and the number of customer requests in the system (Q), which is also known as the queue length:

$$Q = \lambda \times W$$

Note that in this context, the queue length Q refers both to customer requests in service (Q_s) and waiting in a queue (Q_q) for processing. In Windows Server 2003, there are several opportunities to take advantage of Little's Law to estimate the response time of applications where only the arrival rate and queue length are known. For the record, Little's Law is a very general result that applies to a large class of queuing models. It allows you to estimate response time in a situation in which measurements for both the arrival rate and the queue length are available. Note that Little's Law itself provides no insight into how the response time (W) is broken down into the service time (W_s) and the queue time delay (W_q).

Unfortunately, defining suitable boundaries for transactions in Windows applications is very difficult, which is why there are not more direct measurements of response time available in Windows Server 2003. Using Little's Law, in at least one instance (discussed later in this book) it is possible to derive reliable estimates of response time from the available measurements.

The response time to service a request at a resource is usually a nonlinear function of its utilization. This nonlinear relation between response time and utilization that usually holds is known as Little's Law. Little's Law explains why linear scalability of applications is so difficult to achieve. It is a simple and powerful construct, with many applications to computer performance analysis. However, don't expect simple formulas like Little's Law to explain everything in computer performance. This chapter, for example, will highlight several common situations where intelligent scheduling algorithms actually reduce service time at some computer resources the busier the resource gets. You cannot apply simple concepts like Little's Law unreflexively to many of the more complicated situations you can expect to encounter.

Response Time Revisited

As the utilization of shared components increases, processing delays tend to be encountered more frequently. When the network is heavily utilized and Ethernet collisions occur, for example, network interface cards are forced to retransmit packets. As a result, the service time of individual network requests elongates. The fact that increasing the rate of requests often leads to processing delays at busy shared components is crucial. It means that you should expect that as the load on your server rises and bottlenecks in the configuration start to appear, the overall response time associated with processing requests will not hold steady. Not only will the response time for requests increase as utilization increases, but that response time will likely increase in a *nonlinear* relationship with respect to utilization. In other words, as the utilization of a device increases slightly from 80 percent to 90 percent busy, you might observe that the response time of requests doubles, for example.

Response time, then, encompasses both the service time at the device processing the request *and any other delays* encountered waiting for processing. Formally, response time is defined as

response time = service time + queue time

where queue time (W_q) represents the amount of time a request waits for service. In general, at low levels of utilization, there is minimal queuing, which allows device service time to dictate response time. As utilization increases, however, queue time increases nonlinearly and, soon, grows to dominate response time.

Note Although this discussion focuses on the behavior of a queue at a single resource, network queuing models allow you to step back and analyze the response time of customer transactions at a higher level. The customer transaction must first be decomposed into a series of service demands against a set of related resources—how many times, on average, each transaction uses each of the disks, for example. Then the response time of the high-level transaction can be modeled as the sum of the response times at each individual resource.

Conceptually, a client transaction can be represented using a workstation component, a network transmission component, and a server component. Each of these subcomponents can be further understood as having a processor component, a disk component, and a network component, and so on. To track down the source of a performance problem, you might need measurement data on *every* one of the resources involved in processing the request.

Queues are essentially data structures where requests for service are parked until they can be serviced. Examples of queues abound in Windows Server 2003, and measures showing the queue length or the amount of time requests wait in the queue for processing are some of the most important indicators of performance bottlenecks you are likely to find. The Windows Server 2003 queues that will be discussed here include the operating system's thread scheduling queue, logical and physical disk queues, the network interface queue, and the queue of Active Server Pages (ASP) and ASP.NET Web server requests. Little's Law shows how the rate of requests, the response time, and the queue length are related. This relationship allows us to calculate, for example, average response times for ASP.NET requests even though Windows Server 2003 does not report the average response time of these applications directly.

Preventing Bottlenecks

With regular performance monitoring procedures, you will be able to identify devices with high utilizations that lead to long queues in which requests are delayed. These devices are bottlenecks throttling system performance. Intuitively, what can you do about a bottleneck once you discover one? Several forms of preventive medicine can usually be prescribed:

1. Upgrade to a faster device, if available.
2. Balance the workload across multiple devices, if possible.
3. Reduce the load on the device by tuning the application, if possible.
4. Change scheduling parameters to favor cherished workloads, if possible.

These are hardly mutually exclusive alternatives. Depending on the situation, you might want to try more than one of them. Common sense should dictate which of these alternatives you try first. Which change will have the greatest impact on performance? Which configuration change is the least disruptive? Which is the easiest to implement? Which is possible to back out in case it makes matters worse? Which alternative involves the least additional cost? Sometimes, the choices are fairly obvious, but often these are not easy questions to answer.

Measuring Performance

Response time measurements are also important for another reason. For example, the response time for a Web server is the amount of time between the instant that a client selects a hyperlink and the requested page is returned and displayed on her monitor. Because it reflects the user's perspective, the overall response time is the performance measure of greatest interest to the users of a computer system. It is axiomatic that long delays cause user dissatisfaction with the users' computer systems, although this is usually not a straightforward relationship either. Human factors research, for instance, indicates that users might be bothered more by long, unexpected delays than by consistently long response times that they can become resigned to endure.

Conclusions

This section showed how the scalability concerns of computer performance analysts can be expressed in formal mathematical terms. Queuing systems represent computer workloads in the form of resources and their queues and customer requests for service. The total amount of time a customer waits for a request for service to complete is the response time of the system. Response time includes both time in service at the resource and the time delayed in the resource queue waiting for service. A complex computer system can be represented as a network of interconnected resources and queues.

The utilization of a resource is the product of the average service time of requests and the request rate. This relationship, which is known as the Utilization Law, allows you to calculate the service time of disk I/O requests from the measured utilization of the disk and the rate of I/O completions.

Queue time is often a significant factor delaying service requests. Consequently, minimizing queue time delays is an important performance and tuning technique. A formula known as Little's Law expresses the number of outstanding requests in the system, which includes queued requests, as the product of the arrival rate and response time. Queue time tends to increase exponentially with respect to utilization. The device with the fastest growing queue as the workload grows becomes the bottleneck that constrains performance and limits scalability. Identifying the bottlenecked device in a configuration that is performing poorly is another important technique in performance and tuning. Only configuration and tuning actions that improve service time or reduce queuing at the bottlenecked device can be effective at relieving a capacity constraint. Any other configuration or tuning change that you make will prove fruitless.

The formulas that were discussed in this section are summarized in Table 1-2.

Table 1-2 Formulas Used in Computer Performance Analysis

Formula	Derivation
Response time	response time = service time + queue time
Utilization Law	utilization = service time × arrival rate
Queue time as a function of queue length and service time	queue time = ((queue length–1) × average service time) + (average service time/2)
Queue time as a function of utilization and service time: (M/M/1)	queue time = (service time × utilization) / (1 – utilization)
Queue time as a function of utilization and service time: (M/D/1)	queue time = (service time × utilization) / (1 – utilization) / 2
Little's Law	queue length = arrival rate × response time

The Utilization Law and Little's Law are especially useful when you have measurement data for two of the terms shown in the equation and can use the equations to derive the third term. The simple open queuing model formulas are mainly useful conceptually to demonstrate how response time is related to utilization, modeling the behavior of a bottlenecked system where one or more resources are saturated. The primary method used in computer performance and tuning is bottleneck analysis, which uses measurement data to identify saturated resources and then works to eliminate them systematically.

System Architecture

To use the performance monitoring tools provided with Windows Server 2003 effectively requires a solid background in computer systems architecture. This is knowledge that often takes systems administrators years of experience to acquire. This section discusses the aspects of computer architecture that are particularly important in understanding how to use the Windows Server 2003 performance tools effectively. Among the topics addressed in this section are how Windows Server 2003 thread scheduling at the processor works, how the system manages virtual memory and paging, Windows Server 2003 networking services, and the Windows Server 2003 I/O Manager.

This section describes the basic architectural components of a Windows Server 2003 machine, emphasizing performance-related issues. Discussing these topics here is intended to introduce a complex subject, providing the basic background you need to pursue these more complex topics further, particularly as you begin to explore the performance monitoring tools available to you in Windows Server 2003. Aspects of the operating system that are most important in diagnosing and resolving performance problems are emphasized throughout.

The introductory material presented here is a prerequisite to the discussion in Chapter 3, "Measuring Server Performance," which describes the most important performance statistics that can be gathered. This introductory material also provides the background necessary to understand the information presented in Chapter 5, "Performance Troubleshooting."

Using the Performance Monitor

The discussion of each major component of the Windows Server 2003 operating system focuses on the measurements that are available to help you understand how your computer is performing.

Tip The best way to learn about computer performance is through direct observation. Look for sections marked as Tips that invite you to follow along with the discussion. Use the built-in Performance Monitor tool to examine the performance counters that are directly related to the topic you are reading about.

The Performance Monitor is probably the most important tool that you will use to diagnose performance problems on a Windows Server 2003 machine. Complete documentation about using this and other performance monitoring tools is available in Chapter 2, "Performance Monitoring Tools," in this book. To access the Performance Monitor, from the Run menu, type **perfmon**, or click **Performance** on the Administrative Tools menu.

Performance data in the Performance Monitor is organized into *objects*, which, in turn, contain a set of related *counters*. To open a selection menu so that you can access the current values of the counters, click the plus sign (+) on the button bar that allows you to add counters to a real-time display. The names of the counters to watch are referenced in each Tip section, which look like the Tip that follows. A simple *object-name\counter-name* convention is used here to identify the counter. For example, Processor\% Processor Time identifies a counter called % Processor Time that you will find under the Processor object. A complete guide to the syntax for Performance Monitor counters is also provided in Chapter 2, "Performance Monitoring Tools."

Tip You can observe the performance of your machine in action using the System Monitor console, which runs a desktop application called Performance Monitor. To access the System Monitor console, open the Administrative Tools menu and select Performance. You can watch the current values of many performance statistics using this tool.

Performance statistics are organized into objects and counters. To add counters to the current Performance Monitor Chart View, click the **Add counters** button, identified by a plus sign (+), to select the counter values you want to observe.

Complete documentation about using the System Monitor console is available in Chapter 2, "Performance Monitoring Tools."

Operating Systems

Windows Server 2003 is an operating system program especially designed to run enterprise and departmental-level server applications on a wide range of computers. A computer system includes hardware components such as a central processing unit (CPU, or processor, for short) that performs arithmetic and logical operations, a memory unit that contains active programs and data, at least one disk drive where computer data files are stored permanently, a network interface to allow the computer to communicate with other computers, a video display unit that provides visual feedback, and a keyboard and mouse to allow for human input. An operating system is a control program, a piece of software that allows you to utilize the computer system. An operating system is required to run your computer hardware configuration.

Like many other examples of operating systems software, Windows Server 2003 provides many important control functions, such as these:

- Supports Plug and Play services that allow it to recognize, install, and configure new peripherals such as printers, fax machines, scanners, and tape back-up devices.
- Secures the use of your computers against misappropriation by unauthorized users.
- Allows you to install and run many popular computer applications. In addition, it is capable of running multiple application programs at any one time, known as *multithreading*.
- Allows your programs to store and access data files on local disks. In addition, it allows you to set up and maintain file and print servers so that you can easily share data files and printers with other attached users.
- Allows you to set up and maintain Web servers so that other users can interact and communicate using Internet protocols and other Web services.

Moreover, Windows Server 2003 is designed to function as an application server. It can be configured to serve many roles, including a domain controller responsible for security and authentication services; a database server running Microsoft SQL Server, a database management system (DBMS); a messaging server running the Exchange messaging and collaboration program; and a Web server running the Internet Information Services (IIS) application.

Windows Server 2003 is designed to run on a wide range of computer hardware configurations, including advanced multiprocessor systems costing millions of dollars. Many of the advanced capabilities of Windows Server 2003 are discussed in Chapter 6, "Advanced Performance Topics." From a performance perspective, some of the most important aspects of the Windows Server 2003 operating system are the advanced hardware functions it supports.

Figure 1-8 shows the basic structure of the Windows Server 2003 operating system and its most important components. Operating system components run in a protected mode enforced by the hardware, known as *Privileged state* or *Kernel mode*. Kernel mode programs, such as device drivers, can access hardware components directly. Applications running in User mode cannot—they access hardware components indirectly by calling the operating system services that are responsible for managing hardware devices.

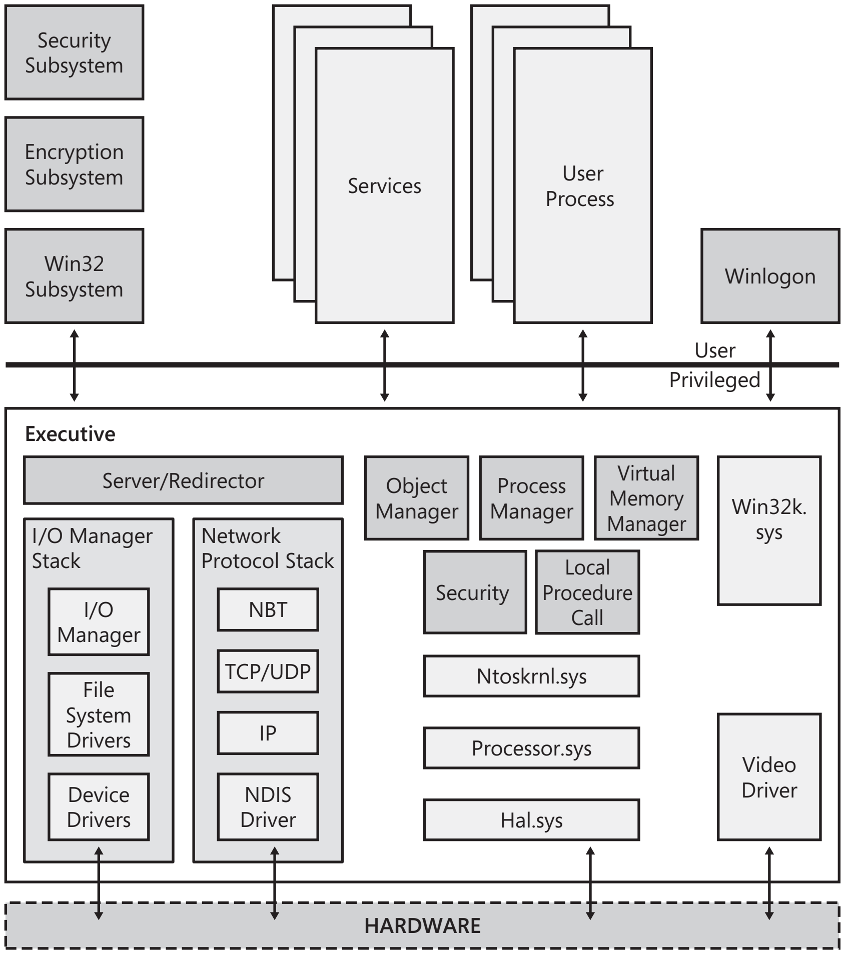


Figure 1-8 The main components of the Windows Server 2003 operating system

The full set of operating system services is known as the Executive, which reflects its role as a control program. Some of the major components of the Executive are discussed further later in this chapter. A more detailed account of the operating system structure and logic is provided in Microsoft Windows Server 2003 Resource Kit Troubleshooting Guide.

Components

Operating system components run in Kernel mode. These components include the kernel, the Executive, and the device drivers, as illustrated in Figure 1-8. Kernel mode is a method of execution that allows code to access all system memory and the full set of processor instructions. The differential access allowed to Kernel mode and User mode processes is enforced by the processor architecture. User mode components are limited to using the set of interfaces provided by the Kernel mode components to interact with system resources.

The kernel itself, contained in Ntoskrnl.exe, is the core of the Windows Server 2003 layered architecture. The kernel performs low-level operating system functions, including thread scheduling, dispatching interrupts, and dispatching exceptions. The kernel controls the operating system's access to the processor or processors. The kernel schedules different blocks of executing code, called threads, for the processors to keep them as busy as possible and to maximize efficiency. The kernel also synchronizes activities among Executive-level subcomponents, such as I/O Manager and Process Manager, and plays a role in troubleshooting by handling hardware exceptions and other hardware-dependent functions.

The kernel works closely with the hardware abstraction layer (HAL). The HAL encapsulates services that allow the operating system to be very portable from one hardware platform to another. It is the principal layer of software that interacts with the processor hardware directly. It provides hardware-dependent implementations for services like thread dispatching and context switching, interrupt processing, instruction-level serialization, and inter-processor signaling. But it hides the details of these implementations from the kernel. An example HAL function is translating a serialization primitive like a spin lock into a hardware-specific instruction sequence that will test and set a data word in memory in an uninterruptible, atomic operation. The HAL exports abstract versions of these services that can be called by the kernel, as well as other Kernel mode programs, allowing them to be written in a manner that is independent of the specific hardware implementation. Kernel mode components can gain access to very efficient HAL spin lock services, for example, without needing to know how these services are actually implemented on the underlying hardware. The HAL also provides routines that allow a single device driver to support the same device on all platforms. Having this hardware abstraction layer between other operating system functions and specific processor hardware is largely responsible for the ease with which the Windows Server 2003 operating system can support so many different processor architectures.

Another hardware-oriented module, Processr.sys, contains routines to take advantage of processor power management interfaces, where they exist.

The Executive provides higher-level operating system functions than the kernel, including Plug and Play services, power management, memory management, process and thread management, and security. The Memory Manager, for example, plays a major role in memory and paging performance, as will be discussed in more detail later.

The Win32k.sys module consolidates support for the elements of the Windows Graphical User Interface (GUI) into a set of highly optimized routines. These routines reside inside the operating system to improve the performance of desktop applications.

The operating system also contains a complete TCP/IP network protocol stack and an I/O Manager stack for communicating with peripheral devices such as disks, tape drives, DVD players, and CD players. The video driver that communicates directly with the graphics display adapter also resides inside the Executive and runs in Privileged mode. The network file server and file server client, respectively Server and Redirector, are services that have major Kernel mode components that run inside the Executive.

Functions

Figure 1-8 also shows several important operating system functions that run in User mode outside the Executive. These include the security subsystem, Smss.exe, and the encryption service contained in Lsass.exe. There are even some Windows graphical user interface (GUI) functions that reside in the client/server subsystem, Csrss.exe. A number of other operating system service processes also perform a variety of valuable functions. Finally, the Windows Logon process, Winlogon.exe, is responsible for authenticating user logons, which are required to establish a desktop session running the GUI shell.

Note Application programs running in User mode are restricted from accessing protected mode memory locations, except through standard operating system calls. To access a protected mode operating system service, User mode applications call the Ntdll.dll communications module that executes a state switch to Kernel mode before calling the appropriate Kernel mode service. Application programs running in User mode are also restricted from accessing memory locations associated with other User mode processes. Both of these restrictions protect the integrity of the Windows Server 2003 system from inadvertent damage by a User mode program. This is why some applications that were designed to run on earlier versions of the DOS or Windows operating systems might not run on Windows Server 2003.

Many important Windows Server 2003 performance considerations revolve around the operating system's support and interaction with hardware components. These include the processor, memory, disks, and network interfaces. In the next sections, the way the operating system manages these hardware components is discussed, along with the most important performance ramifications.

Processors

At the heart of any computer is a central processing unit (CPU), or simply the *processor* for short. The processor is the hardware component responsible for computation—it is a machine that performs arithmetic and logical operations that are presented to it in the form of computer instructions. These instructions are incorporated into computer programs, which are loaded into the computer's memory and become the software that the machine executes. Windows Server 2003 runs on a wide variety of 32-bit and 64-bit processors that vary in speed and architecture. In this section, several important aspects of processor performance in a Windows Server 2003 machine are discussed.

Windows Server 2003 is a computer software program like any other, except that it is designed to interface directly with the computer hardware and control it. The operating system loads first, using a bootstrap program (or boot, for short) that gains control of the machine a little at a time. After the operating system program initializes, it is responsible for loading any additional programs that are scheduled to run. These include the *services* that are scheduled to be loaded automatically immediately following the operating system kernel initialization and, finally, the Winlogon process that allows you access to the Windows desktop. For a more detailed account of the boot process of operating system initialization, see the Microsoft Windows Server 2003 Resource Kit Troubleshooting Guide.

The operating system detects the presence of other hardware resources that are attached to the computer such as memory, disks, network interfaces, and printers, and loads the device driver programs that control access to them. Device drivers that are loaded become part of the operating system. The Plug and Play facilities of Windows Server 2003 allow the operating system to detect and support any additional hardware devices that you happen to connect to the computer at any time after it is up and running.

The operating system is also responsible for determining what other programs are actually run on the computer system. This involves a Scheduler function for running applications fast and efficiently. The Scheduler is responsible for selecting the next program thread for the processor to execute and setting the processor controls that allow the selected thread to run.

Threads

A *thread* is the unit of execution in Windows Server 2003. Every *process* address space has one or more execution threads that contain executable instructions. There are both operating system threads (or kernel threads) and application program threads that the operating system keeps track of. A thread can be in one of several, mutually exclusive Execution states, as illustrated in Figure 1-9.

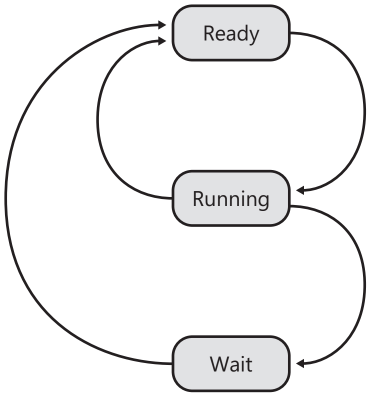


Figure 1-9 Thread Execution state

A thread can be:

- **Running** The running thread is the set of computer instructions the processor is currently executing. The processor hardware is capable of executing only one set of instructions at a time; so, at any one time, only one thread is executing per processor.
- **Ready** A ready thread is one that is eligible to be executed but is waiting for the processor to become free before it actually can execute. The operating system stores the handles of all ready threads in the processor Ready queue, where they are ordered by priority. The priority scheme used in Windows Server 2003 is described a little later.
- **Waiting** A waiting thread is not eligible to run. It is *blocked*. It remains loaded in the computer while it is blocked from running until an event that the thread is waiting for occurs. When a waiting thread unblocks, it transitions to the Ready state. Blocked threads are frequently waiting for an I/O executing on an external device to complete.

Note When a peripheral device such as a disk or a network interface card finishes an operation that it was previously instructed to perform, the device *interrupts* the processor to demand servicing. You can observe the rate that interrupts occur on your machine by accessing the Interrupts/sec counter in the Processor object.

Context switches Computers are loaded with many program threads that all require execution at the same time. Yet only one program thread can actually execute at a time. This is known as *multiprogramming* or *multithreading*. The Windows Server 2003 operating system, of course, keeps track of all the program threads that are loaded and their Execution state. As soon as a running thread blocks—usually because it needs to use one of the devices—the Scheduler function selects among the eligible threads that are ready to run. The Scheduler selects the ready thread with the highest priority waiting in the Ready queue to run next. It then sets the control registers on the processor that determine which program thread executes next and passes control to the thread selected so that its instructions do execute next. This operation is known as a *context switch*. A context switch occurs whenever the operation system passes control from one executing thread to another. Switching threads is one of the functions that the HAL implements because the mechanics of a context switch are processor-specific.

Tip Accessing a counter called System\Context switches/sec allows you to observe the rate at which context switches occur on your machine. You can also observe the rate at which individual program threads execute by accessing the Thread\Context switches/sec counter.

There is also a Thread\Thread State counter that tells you the current Execution state of every thread.

Once a thread is running, it executes continuously on the processor until one of the following events occurs:

- A high priority interrupt occurs, signaling that an external device has completed an operation that was initiated previously.
- The thread voluntarily relinquishes the processor. This is usually because the thread needs to perform I/O or is waiting on a lock or a timer.
- The thread involuntarily relinquishes the processor, usually because it incurred a *page fault*, which requires the system to perform I/O on its behalf. Page fault resolution is discussed in detail in a later section of this chapter.
- A maximum uninterrupted execution time limit is reached. This is known as a *time slice*. At the end of a time slice, the thread remains in the Ready state. The Scheduler returns a thread that has exhausted its time-slice on the processor to the Ready queue and then proceeds to dispatch the highest priority thread that is waiting in the Ready queue. If the long-running thread that was executing is still the highest priority thread on the Ready queue, it will receive another time slice and be scheduled to run next anyway. Time-slicing is discussed in greater detail later in this chapter.

Multithreading The basic rationale for multithreading is that most computing tasks do not execute instructions continuously. After a typical program thread executes for some period of time, it often needs to perform an input/output (I/O) operation like reading information from the disk, printing some text or graphics to a printer, or drawing on the video display. While a program thread is waiting for this input/output function to complete, it is not necessary for the program to hang on to the processor. An operating system that supports multithreading saves the status of a thread that is waiting, restores its status when it is ready to resume execution, and tries to find something else that can run in the meantime.

I/O devices are much slower than the processor, and I/O operations typically take a long time compared to CPU processing. A single I/O operation to a disk might take 5 or 10 milliseconds, which means that the disk is capable of executing perhaps 100 or so such operations per second. Printers, which are even slower, are usually rated in pages printed per minute. In contrast, processors typically execute an instruction at least every one or two clock cycles, where you might be running processors capable of running at 1500–3000 *million* cycles per second. During the time that one thread is delayed doing one disk I/O operation, the processor could be executing some 30,000,000 instructions on behalf of *another* thread.

Although it leads to more efficient utilization of the processor, multithreading actually *slows down* individual execution threads because they are not allowed to run uninterrupted from start to finish. In other words, when the thread that was waiting becomes ready to execute again, it is quite possible for it to be delayed because some higher priority thread is in line ahead of it. Selecting eligible threads from the Ready queue in order by priority is an attempt to ensure that more important work is delayed the least.

Preemptive scheduling Like other multiprogrammed operating systems, Windows Server 2003 manages multiple program threads that are all running concurrently. Of course, only one program can execute at a time on the processor. Threads selected to run by the Windows Server 2003 Scheduler execute until they block, normally because they need to perform an I/O operation to access a device or are waiting for a lock or a timer. Or they execute until an interrupt occurs, which usually signals the completion of an event that a blocked thread was waiting for. After a thread is activated following an interrupt, Windows Server 2003 boosts the dispatching priority of that thread. This means that the thread that was being executed at the time the interrupt occurred is likely to be *preempted* by the now higher priority thread that was waiting for the interrupt to occur. Preemptive scheduling of higher priority work can delay thread execution, but it typically helps to balance processor utilization across CPU and I/O bound threads.

Note Thread priority is boosted following an interrupt and decays over time as the thread executes. This *dynamic* thread scheduling priority scheme in Windows Server 2003 works with time-slicing to help ensure that a CPU-bound thread cannot monopolize the processor when other Ready threads are waiting.

Thread state When an I/O operation completes, a thread that was blocked becomes eligible to run again. This scheme means that a thread alternates back and forth between three states: the *Ready* state where it is eligible to execute, the *Running* state where it actually executes instructions, and a *Wait* state where it is blocked from executing. Logically, a Thread state transition diagram like the one in Figure 1-9 models this behavior.

How Windows Server 2003 Tells Time

The way that Windows Server 2003 tells time is crucial to many of the performance measurements that the operating system takes. To understand how the operating system tells time, you must differentiate between several types of machine hardware "clocks."

The first clock is the machine instruction execution clock cycle, which is measured in MHz. The machine's instruction execution clock is a good indicator of relative processor speed, but is not accessible externally. The operating system has no access to the machine instruction execution clock.

Standard Windows timer services create a virtual system clock in 100 nanosecond units. Because this time unit might or might not map easily into the machine's real-time clock hardware, maintaining the virtual clock is a HAL function. These timer services are built around a periodic native clock interrupt, which is set to occur every 10 milliseconds. Even though the granularity of the clock is in 100 nanosecond units, a clock "tick" actually occurs only once every 10 milliseconds. This is the most familiar form of clock services available in Windows Server 2003. Programmers, for example, call the *SetTimer* application programming interface (API) function to receive a notification for a specific virtual clock interrupt. You can get access to the standard clock interval value by calling *GetSystemTimeAdjustment*.

There are important operating system functions that work off this standard clock interrupt. The first is the Scheduler function that checks the current running thread and performs the % Processor Time accounting that is discussed in this section. A second Scheduler function checks to see whether the running thread has exhausted its time slice.

The clock interrupt that drives the standard timer services relies on a native system clock, normally a hardware function provided by a chipset external to the processor. This is also known as the High Precision clock because the native system clock usually has significantly higher resolution than the standard Windows timer services. The precise granularity of the native system clock is specific to the external clock hardware. Win32 programmers can get access to a high precision clock using *QueryPerformanceCounter* and *QueryPerformanceFrequency*. For example, consider http://support.microsoft.com//kb/172338. In this example, the standard 10-millisecond clock timer does not offer enough resolution to time an instruction loop with sufficient granularity.

Tip A thread that is blocked is waiting for some system event to occur. The event signals that the transition from Waiting to Ready can occur. The Thread\Wait Reason counter shows the reason threads are blocked. You will find that most threads are waiting for a signal from the operating system Executive, which corresponds to a Thread\Wait Reason value of 7. The operating system activates a thread that is waiting on an event and makes it Ready to run when that event occurs. Common events include waiting for a clock timer to expire; waiting for an I/O to complete; or waiting for some requested system service, such as authentication or encryption, to complete.

Interrupt Processing

An interrupt is a signal from an external device to the processor. Hardware devices *raise* interrupts to request servicing immediately.

When an I/O request to a disk device, for example, is initiated, the device processes the request independently of the processor. When the device completes the request, it raises an interrupt to signal the processor that the operation has completed. This signal is treated as a very high priority event: the device is relatively slow compared to the processor, the device needs attention, and some other user might be waiting for the device to become free. When the processor recognizes the interrupt request (IRQ), it does the following:

1. It stops whatever it is doing immediately (unless it is already servicing a higher priority interrupt request).
2. The device's Interrupt Service Routine (ISR) is dispatched to begin processing the interrupt. The Interrupt Service Routine is a function of the device driver.
3. The ISR saves the status of the current running thread. This status information is used to restore the interrupted thread to its previous Execution state the next time it is selected to run.
4. The ISR stops the device from interrupting and then services the device that raised the interrupt.

This is why the process is known as an *interrupt*: the normal flow of thread execution is interrupted. The thread that was running when the interrupt occurred returns to the Ready queue. It might *not* be the thread the Scheduler selects to run following interrupt processing. In addition, interrupt processing is likely to add another thread to the Ready queue, namely the thread that was waiting for the event to occur.

Note The thread execution status information that the ISR saves is also known as the *thread context*. The thread context includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. For more information about the thread context, see the Platform SDK.

In Windows Server 2003, one consequence of an interrupt occurring is a likely reordering of the Scheduler Ready queue following interrupt processing. The device driver that completes the interrupt processing supplies a *boost* to the priority of the application thread that transitions from Waiting to Ready when the interrupt processing completes. Interrupt processing juggles priorities so that the thread made ready to run following interrupt processing is likely to be the highest thread waiting to run in the Ready queue. Consequently, the application thread that had been waiting for an I/O request to complete is likely to receive service at the processor next.

Voluntary Wait

A thread voluntarily relinquishes the processor when it issues an I/O request and then waits for the request to complete. Other voluntary waits include a timer wait, or waiting for a serialization signal from another thread. A thread issuing a voluntary wait enters the Wait state. This causes the Windows Server 2003 Scheduler to select the highest priority task waiting in the Ready queue to execute next. Threads with a Wait State Reason value of 7, waiting for a component of the Windows 2000 Executive, are in the voluntary Wait state.

Involuntary Wait

Involuntary waits are usually associated with virtual memory management. For example, a thread enters an involuntary Wait state when the processor attempts to execute an instruction that references data in a buffer that happens not to be currently resident in physical memory (or RAM). Because the instruction indicated cannot be executed, the processor generates a *page fault* interrupt, which the Virtual Memory Manager (VMM) must resolve by allocating a free page in memory, reading the appropriate page containing the necessary instruction or data into memory from disk, and re-executing the failed instruction.

A currently running thread encountering a page fault is promptly suspended with the thread context reset to re-execute the failing instruction. The suspended task is placed in an involuntary Wait state until the page requested can be brought into memory and the instruction executed successfully. At that point, the Virtual Memory Manager component of the operating system is responsible for resolving the page fault and transitioning the thread from the Wait state back to Ready. Virtual memory and Paging are topics that are revisited later in this chapter.

Tip There are several Thread Wait Reason values that correspond to virtual memory involuntary waits. See the Thread\Thread Wait Reason Explain text for more details. When you are able to observe threads delayed with these Wait Reasons, it is usually a sign that there is excessive memory management overhead, an indication that the machine has a shortage of RAM, which forces the memory management routines to work harder.

Time-Slicing

A running thread that almost never needs to perform I/O or block waiting for an event is not allowed to monopolize the processor completely. Without intervention from the Scheduler, some very CPU-intensive execution threads will attempt to do this. There is also the possibility that a program bug will cause the thread to go into an infinite loop in which it will attempt to execute continuously. Either way, the Windows Server 2003 Scheduler will eventually interrupt the running thread if no other type of interrupt occurs. If the thread is not inclined to relinquish the processor voluntarily, the Scheduler eventually forces the thread to return to the Ready queue. This form of processor sharing is called *time-slicing*, and it is designed to prevent a CPU-bound task from dominating the use of the processor for an extended period of time. Without time-slicing, a high priority CPU-intensive thread could indefinitely delay other threads waiting in the Ready queue. The Scheduler implements time-slicing using a high-priority clock timer interrupt that is set to occur at regular intervals to check on the threads that are running. For more information about this clock interrupt, see "How Windows Server 2003 Tells Time" sidebar.

When a thread's allotted time slice is exhausted, the Windows Server 2003 Scheduler interrupts it and looks for another Ready thread to dispatch. Of course, if the interrupted thread still happens to be the highest priority Ready thread (or the only Ready thread), the Scheduler is going to select it to run again immediately. The Scheduler also lowers the priority of any thread that was previously boosted when the thread executes for the entire duration of its time-slice. This further reduces the likelihood that a CPU-intensive thread will monopolize the processor. This technique of boosting the relative priority of threads waiting on device interrupts and reducing the priority of CPU-intensive threads helps to ensure that a CPU-bound thread cannot monopolize the processor when other Ready threads are waiting.

The duration of a thread's time-slice is established by default. Under Windows Server 2003, the time slice value is a long interval, which usually benefits long-running server application threads. Longer time slices lead to less overhead from thread context switching. Shorter time slices generally benefit interactive work. You might consider changing the default time slice value to a shorter interval on a Windows Server 2003 machine that was used predominantly for interactive work under Terminal Services, for example. The time-slice settings and criteria for changing them are discussed in Chapter 6, "Advanced Performance Topics."

Note The Scheduler uses a simple but flexible mechanism for making sure that running threads do not execute continuously for more than their allotted time-slice. At the time the Scheduler initially selects a thread to run, the thread receives an initial time-slice allotment in quantums. The quantum corresponds to the periodic clock interrupt interval. During each periodic clock interval that the thread is found running, the Scheduler subtracts several quantums from the allotment. When the thread has exhausted its time allotment—that is, the number of quantums it has remaining falls to zero—the Scheduler forces the thread to return to the Ready queue.

Idle Thread

When the current thread that is running *blocks*, often because of I/O, the Windows Server 2003 Scheduler finds some other thread that is ready to run and schedules it for execution. What if no other program threads are ready to run?

If no threads are ready to run, the Windows Server 2003 Scheduler calls a HAL routine known as the Idle thread. The *Idle* thread is not a true thread, nor is it associated with a real process. There is also no Scheduling priority associated with the Idle thread. In reality, the Idle thread is a bookkeeping mechanism that is provided to allow the operating system to measure processor utilization.

Normally, the Idle thread routine will execute a low priority instruction loop continuously until the next interrupt occurs, signaling that there is real work to be done. But, for example, if the processor supports power management, the Idle thread routine will eventually call the Processr.sys module to instruct the processor to change to a state where it consumes less power. The way the Idle thread is implemented is discussed in greater detail in Chapter 5, "Performance Troubleshooting."

Accounting for Processor Usage

Windows Server 2003 uses a sampling technique to account for processor usage at the thread, process, and processor level. The operating system allows the built-in system clock to generate a high priority interrupt periodically, normally 100 times per second. During the servicing of this periodic interval interrupt, the Interrupt Service Routine checks to see which thread was running when the interrupt occurred. The ISR then increments a timer tick counter field (a timer tick is 100 nanoseconds) in the Thread Environment Block to account for the processor usage during the last interval between periodic interrupts. Note that all the processor time during the interval is attributed to the thread that was executing when the periodic clock interrupt occurred. This is why the % Processor Time measurements Windows Server 2003 makes should be interpreted as sampled values.

Tip Use the Thread\% Processor Time counter to see how much processor time a thread accumulates. This data is also rolled up to the process level where you can watch a similar Process\% Processor Time counter. Because the sampling technique used to account for processor usage requires a respectable number of samples for any degree of accuracy, the smallest data collection interval that System Monitor allows is 1 second.

Because the periodic clock interrupt is very high priority, it is possible to account for processor usage during the Interrupt state as well as threads running in either the Privileged state or the User state.

Tip The portion of time that a thread is executing in User mode is captured as Thread\% User Time. Privileged mode execution time is captured in the Thread\% Privileged Time counter. % User Time and % Privileged Time are also measured at the Process and Processor levels.

The periodic clock interrupt might also catch the processor when it was previously executing the Idle thread function in either the HAL or the Processr.sys routine. The Idle thread is allowed to accumulate processor utilization clock tick samples exactly like real threads. By subtracting the amount of time the periodic clock interval routine found the system running the Idle thread from 100 percent, it becomes possible to calculate accurately how busy the processor is over any extended period of observation.

Note Think of the Idle thread as a bookkeeping mechanism instead of a true execution thread. The Processor\% Processor Time counter is calculated by subtracting the amount of time the system found that the Idle thread was running from 100 percent. On a multiprocessor machine, there are separate bookkeeping instances of the Idle thread, each dedicated to a specific processor. The _Total instance of the Processor object actually reports the *average* % Processor Time across all processors during the interval.

There are two additional subcategories of processor time usage that Windows Server 2003 breaks out. The % Interrupt Time represents processor cycles consumed in device driver Interrupt Service Routines, which process interrupts from attached peripherals such as the keyboard, mouse, disks, and network interface cards. Interrupt processing was discussed earlier in this section. This is work performed at very high priority, typically while other interrupts are disabled. It is captured and reported separately not only because of its high priority, but also because it is not easily associated with any particular User mode process.

Windows Server 2003 also tracks the amount of time device drivers spend executing deferred procedure calls (DPCs), which also service peripheral devices, but run with interrupts enabled. DPCs represent higher priority work than other system calls and kernel thread activity. Note that both ISRs and DPCs are discussed later in this chapter when the priority queuing mechanism in Microsoft Windows Server 2003 is described in more detail.

Note % DPC Time is also included in % Privileged Time. Like % Interrupt Time, it is only available at the Processor level. When the periodic clock interval catches the system executing an ISR or a DPC, it is not possible to associate this interrupt processing time with any specific User or Kernel mode thread.

Transient threads and processes Although it is usually very accurate, the sampling technique that the operating system uses to account for processor usage can miss some of what is happening on your system. Watch out for any transient threads and processes that execute for so little time that you can miss them entirely. Once a thread terminates, the processor timer ticks it has accumulated are also destroyed. This is also true at the process level. When a process terminates, all its associated threads are destroyed. At the next Performance Monitor collection interval, there is no evidence that process or thread ever existed!

Tip If you discover that too much of the % Processor Time you gather at the processor level is unaccounted for at the process or thread level, your machine might be running many transient processes. Examine the process Elapsed Time counter to determine whether you have a large number of transient processes that execute very quickly. Increase the rate of Performance Monitor data collection until the sample rate is less than 1/2 the average elapsed time of your transient processes. This will ensure that you are gathering performance data rapidly enough to catch most of these transient processes in action.

Increasing the Performance Monitor data collection sample rate also has overhead considerations, which are discussed in Chapter 2, "Performance Monitoring Tools."

Normalizing CPU time All the % Processor Time utilization measurements that the operating system gathers are reported relative to the processing power of the hardware. When you use a measurement that reports the processor as being, say, 60 percent, the logical question to ask is "Percent of what?" This is a good question to ask because you can expect a program running on a 1 GHz Pentium to use three times the amount of processor time as the same program running on a 3 GHz Pentium machine.

For comparisons across hardware, normalizing CPU seconds based on a standard hardware platform can be useful. Fortunately, both Intel and AMD microprocessors identify their clock speed to the initialization NTDETECT routine. Use the System item in Control Panel to determine the clock speed of the processor installed in your machine. This clock speed value is also stored in the Registry in the ~MHz field under the HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor key. When it is available, a *ProcessorNameString* can also be found there that provides similar information.

Processor Ready Queue

The System\Processor Queue Length counter is another important indicator of processor performance. This is an instantaneous peek at the number of Ready threads that are currently delayed waiting to run. The Processor Queue Length counter is reported only at the system level because there is a single Scheduler Dispatch Queue containing all the threads that are ready to run that is shared by all processors on a multiprocessor. (The operating system does maintain separate queue structures per processor, however, to enhance performance of the Scheduler on a multiprocessor.) The Thread State counter in the Thread object, as discussed earlier, indicates *which* threads at the moment are waiting for service at the processor or processors.

When the processor is heavily utilized, there is a greater chance that large values for the Processor Queue Length can also be observed. The longer the queue, the longer the delays that threads encounter waiting to execute.

Keep in mind when you are comparing processor utilization and Processor Queue Length that the former represents a continuously sampled value, whereas the latter represents a single observation reflecting the measurement taken at the last periodic clock interval. For example, if you gather performance measurements once per second, the processor utilization statistics reflect about 100 samples per second. In contrast, the System\Processor Queue Length counter is based only on the last of these samples. This discontinuity can distort the relationship between the two measurements.

> **Note** When the system is lightly utilized, you might see unexpectedly large values of the Processor Queue Length counter. This is an artifact of the way the counter is derived using the periodic clock interval.

Priority Queuing

Three major processor dispatching priority levels determine the order in which Ready threads are scheduled to run. The highest priority work in the system is performed at Interrupt priority by ISRs. The next highest priority is known as Dispatch level. Dispatch level is where high priority systems routines known as asynchronous procedure calls (APCs) and deferred procedure calls (DPCs) run. Finally, there is the Passive Dispatch level where both Kernel mode and User mode threads are scheduled. As illustrated in Figure 1-10, the three major processor dispatching priority levels determine the order in which Ready threads are scheduled to run.

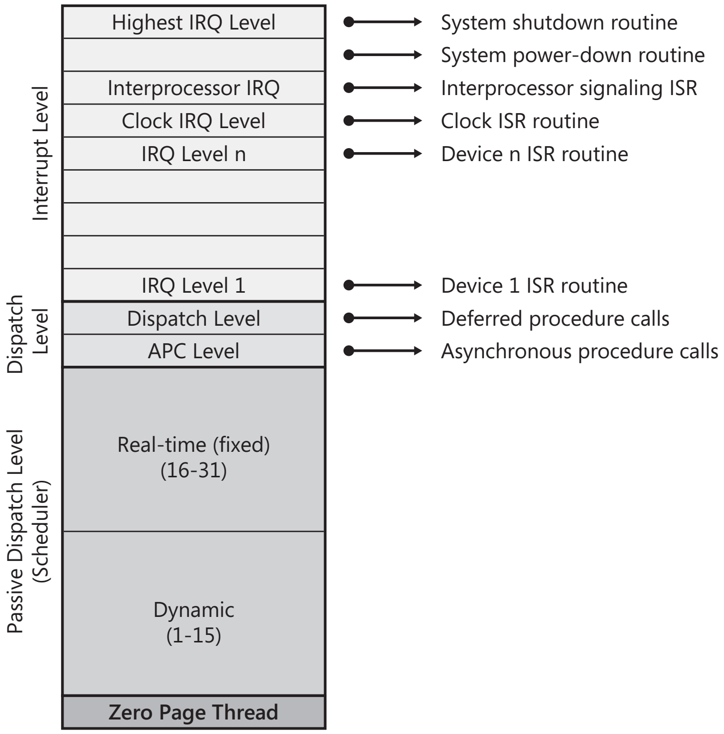


Figure 1-10 The overall priority scheme

Interrupt priority Interrupts are subject to priority. The interrupt priority scheme is determined by the hardware, but in the interest of portability, the priority scheme is abstracted by the Windows Server 2003 HAL. During interrupt processing, interrupts from lower priority interrupts are disabled—or *masked*, in hardware terminology—so that they remain pending until the current interrupt processing routine completes. Lower priority devices that attempt to interrupt the processor while it is running disabled for interrupts remain pending until the ISR routine finishes and once again enables the processor for interrupts. Following interrupt processing, the operating system resets the processor to return to its normal operating mode, where it is once again able to receive and process interrupt signals.

> **Note** Running in Disabled mode has some important consequences for Interrupt Service Routines. Because interrupts are disabled, the ISR cannot sustain a page fault, something that would normally generate an interrupt. A page fault in an ISR is a fatal error. To avoid page faults, device drivers allocate memory-resident work areas from the system's Nonpaged pool. For a more detailed discussion of the Nonpaged pool, see the section entitled "System Working Set," later in this chapter.

Hardware device interrupts are serviced by an Interrupt Service Routine, or ISR, which is a standard device driver function. Device drivers are extensions of the operating system tailored to respond to the specific characteristics of the devices they understand and know how to control. The ISR code executes at the interrupt level priority, with interrupts disabled at the same level or lower level. An ISR is high priority by definition because it interrupts the regularly scheduled thread and executes until it voluntarily relinquishes the processor (or is itself interrupted by a higher priority interrupt).

Deferred procedure calls For the sake of performance, ISRs should perform the minimum amount of processing necessary to service the interrupt in Disabled mode. Any additional device interrupt-related processing that can be performed with the system once again enabled for interrupts is executed in a routine that the ISR schedules to run after interrupts are re-enabled. This special post-interrupt routine is called a deferred procedure call (DPC).

After all pending interrupts are cleared, queued DPCs are dispatched until the DPC queue itself is empty.

> **Tip** For each processor, % Interrupt Time and % DPC Time counters are available. Both % Interrupt Time and % DPC Time are also included in the % Privileged Time counter. If you want to report on % Interrupt Time and % DPC Time separately, make sure you then subtract them from % Privileged Time.

Thread dispatching priority After all interrupts are cleared and the DPC queue is empty, the Windows Server 2003 Scheduler is invoked. The Scheduler examines the Ready queue, selects the highest priority ready thread, and instructs the processor to begin executing that thread.

Threads that are ready to run are ordered by priority. Thread dispatching priority is a number that ranges from zero through 31. The higher the number, the higher the priority. The thread dispatching priority scheme is illustrated in Figure 1-11. Zero is the lowest priority and is reserved for use by the system zero page thread. The priority values 1–31 are divided into two sections, called the *dynamic* range (1–15) and the *real-time* range (16–31). Priority values in the range of 16–31 are used by many operating system kernel threads. They are seldom used by normal User mode applications. Real-time priorities are fixed values.

The remaining priorities with values from 1 through 15 are known as the *dynamic priority* range. When a User mode process is created, it begins life at the Normal base priority, which corresponds to a priority of 8. A priority of 8 is the midpoint of the dynamic range. The base priority of a process can be adjusted by making a call to the *SetPriorityClass* Win32 API function. Using *SetPriorityClass*, you can choose among the following base priorities: idle, below normal, normal, above normal, and high, as shown in Figure 1-11. (Below Normal and Above Normal are seldom used.)

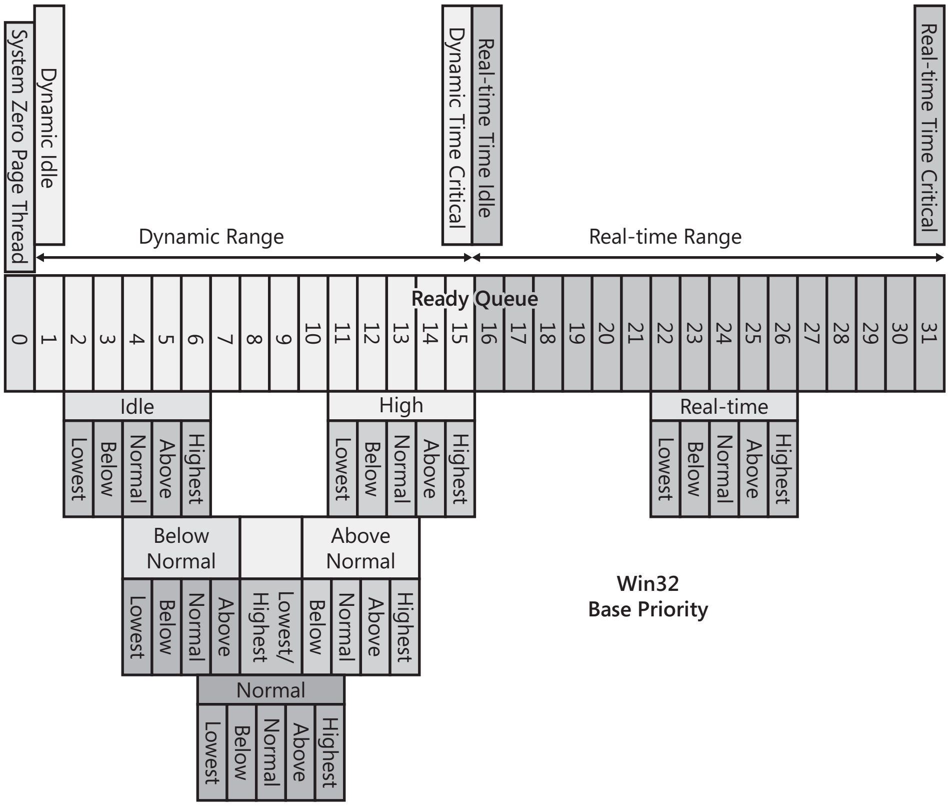


Figure 1-11 Base priorities and their ranges

Note Priority 0 is reserved for exclusive use by the operating system's zero page thread. This is a low priority thread that places zero values in free memory pages. The Idle Thread, which is a bookkeeping mechanism rather than an actual execution thread, has no priority level associated with it.

Threads inherit the base priority of the process when they are created, but they can adjust their priority upward or downward from the base setting dynamically during run time. Within each dynamic priority class, five priority adjustments can be made at the thread level by calling *SetThreadPriority*. These adjustments are *highest*, *above normal*, *normal*, *below normal*, and *lowest*. They correspond to +2, +1, +0, –1, and –2 priority levels above or below the base level, as illustrated in Figure 1-11. The Win32 API also provides for two extreme adjustments within the dynamic range to either *time-critical*, or priority 15, and *idle*, or priority 1.

Tip You can monitor a thread's Base Priority and its Priority Current, the latest priority level of the thread. Current priority is subject to adjustment in the dynamic range, with a boost applied to the priority of a thread transitioning from a voluntary Wait to Ready. Boosted thread scheduling priority decays over time. At the process level, you can monitor a Process's Base Priority.

The thread priority adjustments allow an application designer to give threads performing time-critical work like managing the application's main window or responding to keyboard and mouse events higher dispatching priority than other threads processing within the application. On the other hand, threads performing longer-running background processes can be set to priorities below normal. An application might even expose a tuning parameter that allows a system administrator to determine what the dispatching priority of a process or some of its threads should be. Several of these Dispatch priority tuning parameters are discussed in Chapter 6, "Advanced Performance Topics."

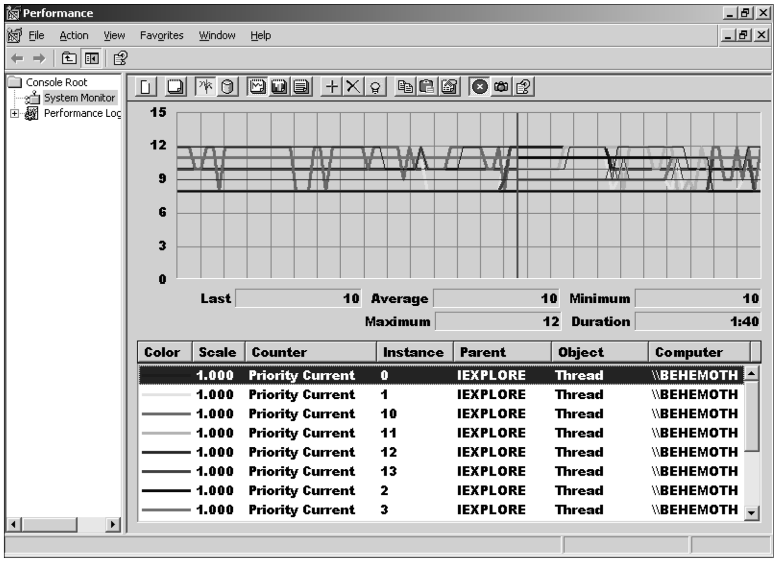
When there is no higher priority interrupt processing, DPCs, or APCs to dispatch, the Windows Server 2003 Scheduler scans the Ready queue to find the highest priority ready thread to run next. Ready threads at the same priority level are ordered in a FIFO (First In, First Out) queue so that there are no ties. This type of scheduling within the priority level is also called *round robin*. Round robin, as noted earlier, is considered a form of fair scheduling.

Dynamic priority adjustments Priorities in the 1–15 range are called *dynamic* because they can be adjusted based on their current behavior. A thread that relinquishes the processor voluntarily usually has its priority *boosted* when the event it is waiting for finally occurs. Boosts are cumulative, although it is not possible to boost a thread above priority 14, the next-to-highest priority level in the dynamic range. (This leaves priority level 15 in the dynamic range available for time-critical work.)

Threads that have their priority boosted are also subject to *decay* back to their current base. Following a dynamic priority boost, a thread in the Running state has its priority reduced over time. As a boosted priority of a running thread decays over time, priority is never pushed below its original base priority. At the expiration of its time slice, a running thread is forced to return to the Ready queue. At the end of its time slice, thread priority is reset to the level prior to its original boost.

Tip Using the System Monitor console, you can observe these thread dynamic priority adjustments in action. Observe the Thread\ Priority Current counter for all the threads of an application like Microsoft Internet Explorer while you are using it to browse the Web. You will see that the current priority of some of the threads of the Internet Explorer process are constantly being adjusted upward and downward, as illustrated.

These priority adjustments only apply to threads in the dynamic range—that is why it is known as the "dynamic" range, as Figure 1-12 illustrates. The effect of these dynamic adjustments is to boost the priority of threads waiting on I/O devices and lower the priority of long running threads. This approximates the Mean Time To Wait scheduling algorithm, which optimizes processor throughput and minimizes processor queuing.



Last 10 Average 10 Minimum 10
Maximum 12 Duration 1:40

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	1.000	Priority Current	0	IEXPLORE	Thread	\\BEHEMOTH
▬	1.000	Priority Current	1	IEXPLORE	Thread	\\BEHEMOTH
▬	1.000	Priority Current	10	IEXPLORE	Thread	\\BEHEMOTH
▬	1.000	Priority Current	11	IEXPLORE	Thread	\\BEHEMOTH
▬	1.000	Priority Current	12	IEXPLORE	Thread	\\BEHEMOTH
▬	1.000	Priority Current	13	IEXPLORE	Thread	\\BEHEMOTH
▬	1.000	Priority Current	2	IEXPLORE	Thread	\\BEHEMOTH
▬	1.000	Priority Current	3	IEXPLORE	Thread	\\BEHEMOTH

Figure 1-12 Dispatching priorities in the dynamic range are subject to dynamic adjustment

Caution Because these thread adjustments are made automatically thousands of time per second based on current thread execution behavior, they are likely to be much more effective than almost any static priority scheme that you can devise yourself. Override the priority adjustments that the operating system makes automatically only after a thorough study and careful consideration.

There are no priority adjustments made for threads running in the real-time range.

Important You normally do not need to worry much about the detailed mechanisms that the Windows Server 2003 Scheduler uses unless:

- The processor is very busy for extended periods of time
- There are too many threads delayed in the Ready queue

When these two conditions are true, the processor itself is a potential performance bottleneck. Troubleshooting processor bottlenecks is one of the topics discussed in Chapter 5, "Performance Troubleshooting."

If you are experiencing a processor bottleneck, you might consider adjusting the default settings that govern time-slicing. These settings are discussed in Chapter 6, "Advanced Performance Topics."

Although it is seldom necessary to fine-tune Scheduler parameters like the length of a time slice, Windows Server 2003 does expose one tuning parameter in the Registry called *Win32PrioritySeparation*. When to use the *Win32PrioritySeparation* setting is discussed in Chapter 6, "Advanced Performance Topics."

Processor Affinity

On machines with multiple processors, there are additional thread scheduling considerations. By default, Windows Server 2003 multiprocessors are configured *symmetrically*. Symmetric multiprocessing means that any thread can be dispatched on any processor. ISRs and DPCs for processing external device interrupts can also run on any available processor when the configuration is symmetric.

Even though symmetric multiprocessing is easy and convenient, it is not always optimal from a performance standpoint. Multiprocessor performance will often improve if very active threads are dispatched on the same physical processors. The key to this performance improvement is the *cache* techniques that today's processors utilize to speed up instruction execution rates. Processor caches store the contents of frequently accessed memory locations. These caches—and there are several—are high-speed buffers located on the microprocessor chip. Fetching either instructions or data from a cache is several times faster than accessing RAM directly.

When an execution thread is initially loaded on the processor following a context switch, the processor cache is empty of the code and data areas associated with the thread. This is known as cache *cold start*. Time-consuming memory fetches slow down instruction execution rates during a cache cold start until some time afterwards as the cache begins to fill up with the code and data areas the thread references during the execution interval. Over time, the various processor caches become loaded with the thread's frequently accessed data, and the instruction execution rate accelerates.

Because the performance difference between a cache cold start and a warm start is substantial, it is worthwhile to use a thread's history to select among the available processors where the thread can be scheduled to run. Instead of always experiencing a slow cache cold start, a thread dispatched on the same physical processor where it executed last might experience a faster cache warm start. Some of the data and instructions from memory that the thread accesses might still be in the processor caches from the last time the thread was running.

Windows Server 2003 uses the thread's history on a multiprocessor to make its scheduling decisions. Favoring one of the processors out of the many that could be available for thread dispatching on a multiprocessor is known as *processor affinity*. The physical processor where the thread was dispatched last is known as the thread's *ideal processor*. If the ideal processor is available when a thread is selected to run, the thread is dispatched on that processor. This is known as *soft affinity*, because if the thread's ideal processor is not available—it is already busy running a thread of equal or higher priority—the thread will be dispatched on a less than ideal, but still available processor. If the ideal processor is busy, but it is running a lower priority thread, the lower priority thread is pre-empted.

Windows Server 2003 also supports *hard affinity* in which certain threads can be restricted to being dispatched on a subset of the available processors. Hard processor affinity can be an important configuration and tuning option to use for larger multiprocessor configurations, but it needs to be used with care. More information about using hard processor affinity is available in Chapter 6, "Advanced Performance Topics."

Memory and Paging

Random access memory (RAM) is an essential element of your computer. Programs are loaded from disk into memory so that they can be executed. Each memory location has a unique address, allowing instructions to access and modify the data that is stored there. Data files stored on the disk must first be loaded into memory before instructions to manipulate and update that data can be executed. In this section, several important performance-related aspects of memory usage and paging are discussed.

Physical memory (or real memory) needs to be distinguished from *virtual memory*. Only specific operating system kernel functions access and operate on physical memory locations directly on a machine running Windows Server 2003. Application programs use virtual memory instead, addressing memory locations indirectly.

Virtual memory addressing is a hardware feature of all the processors that Windows Server 2003 supports. Supporting virtual memory requires close cooperation between the hardware and the operating system software. The operating system is responsible for mapping virtual memory addresses to physical memory locations so that the processor hardware can translate virtual addresses to physical ones as program threads execute. Virtual memory addressing also allows executing programs to reference larger ranges of memory addresses than might actually be installed on the machine. The operating system is responsible for managing the contents of physical memory so that this virtual addressing scheme runs as efficiently as possible.

Virtual Addressing

Virtual memory is a feature supported by most advanced processors. Hardware support for virtual memory includes a hardware mechanism to map from logical (that is, virtual) memory addresses that application programs reference to physical memory hardware addresses. When an executable program's image file is first loaded into memory, the logical memory address range of the application is divided into fixed size chunks called *pages*. As these logical pages are referenced by an executing program, they are then mapped to similar-sized physical pages that are resident in physical memory. This mapping is dynamic so that the operating system can ensure that frequently referenced logical addresses reside in physical memory, while infrequently referenced pages are relegated to paging files on secondary disk storage.

Virtual memory addressing allows executing processes to co-exist in physical memory without any risk that a thread executing in one process can access physical memory belonging to another. The operating system creates a separate and independent *virtual address space* for each individual process that is launched. On 32-bit processors, each process virtual address space can be as large as 4 GB. On 64-bit processors, process virtual address spaces can be as large as 16 terabytes in Windows Server 2003. Note that each process virtual address space must allow for the range of virtual addresses that operating system functions use. For example, the 4-GB process virtual address space on 32-bit machines is divided by default into a 2-GB range of private addresses that User mode threads can address and a 2-GB range of system addresses that only kernel threads can address. Application program threads can access only virtual memory locations associated with their parent process virtual address space. A User mode thread that attempts to access a virtual memory address that is in the system range or is outside the range of currently allocated virtual addresses causes an Access violation the operating system will trap.

Virtual memory systems work well because for executing programs to run, they seldom require all their pages to be resident in physical memory concurrently. The active subset of virtual memory pages associated with a single process address space that is currently resident in RAM is known as the process's *working set* because those are the active pages the program references as it executes. With virtual memory, only the active pages associated with a program's current working set remain resident in physical memory. On the other hand, virtual memory systems can run very poorly when the working sets of active processes greatly exceed the amount of RAM that the computer contains. Serious performance problems can arise when physical memory is over-committed. Windows Server 2003 provides virtual and physical memory usage statistics so that you can recognize when an acute shortage of physical memory leads to performance problems.

Page Tables

Virtual memory addresses are grouped into fixed-size blocks of memory called *pages*. The virtual memory pages of a process are backed by pages in physical memory that are the same size. Page Tables, built and maintained by the operating system, are used to map virtual memory pages to physical memory. The processor hardware specifies the size of pages and the format of the Page Tables that are used to map them. This mapping is dynamic, performed on demand by the operating system as threads running in the process address space access new virtual memory locations. Because available RAM is allocated for active pages on demand, virtual memory systems use RAM very efficiently.

One advantage of virtual memory addressing is that separate application programs loaded into RAM concurrently are both isolated and protected from each other when they run. Threads associated with a process can reference only the physical memory locations that correspond to the process's unique set of virtual memory pages. This makes it impossible for a bug in one program to access memory in another executing program's virtual address space. Another advantage is that User mode programs can be written largely independently of how much RAM is actually installed on any particular machine. A process can be written to reference a uniform-sized virtual address space regardless of how much physical memory is present on the machine.

Virtual memory addresses are assigned to physical memory locations *on demand*, which has a number of implications for the performance of virtual memory machines. The Memory Manager component of the Windows Server 2003 Executive is responsible for building and maintaining process address space Page Tables. The Memory Manager is also responsible for managing physical memory effectively. It attempts to ensure that an optimal set of pages for each running process—its working set of active pages—resides in RAM.

Note Working set pages are the active pages of a process address space currently backed by RAM. These are *resident* pages. *Nonresident* pages are virtual memory addresses that are allocated but not currently backed by RAM. *Committed pages* are those that have Page Table entries (PTEs). Committed pages can be either resident or nonresident.

Virtual memory addressing makes life easier for programmers because they no longer have to worry about how much physical memory is installed. Virtual memory always makes it appear as if 4 GB of memory is available to use. Virtual memory addressing is also transparent to the average application programmer. User mode threads never access anything but virtual memory addresses.

The Virtual Memory Manager performs several vital tasks in support of virtual addressing. It constructs and maintains the Page Tables. Page Tables are built for each process address space. The function of the Page Tables is to map logical program virtual addresses to physical memory locations. The location of a process's set of Page Tables is passed to the processor hardware during a context switch. The processor loads them and refers to them to perform virtual-to-physical address translation as it executes the thread's instruction stream. This is illustrated in Figure 1-13.

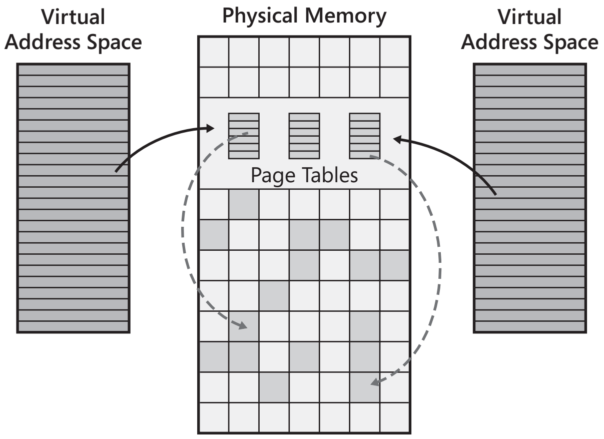


Figure 1-13 Virtual-to-physical address translation

Another key role the operating system plays is to manage the contents of physical memory effectively. VMM implements a Least Recently Used (LRU) page replacement policy to ensure that frequently referenced pages remain in physical memory. VMM also attempts to maintain a pool of free or available pages to ensure that pages faults can be resolved rapidly. Whenever physical memory is in short supply, the VMM page replacement policy replenishes the supply of free (available) pages.

When the virtual pages of active processes overflow the size of RAM, the Memory Manager tries to identify older, inactive pages that are usually better candidates to be removed from RAM and stored on disk instead. The Memory Manager maintains a current copy of any inactive virtual memory pages in the paging file. In practice, this means that the operating systems checks to see whether a page that it temporarily removes from a process working set has been modified since the last time it was stored on the paging file. If the page is unchanged—that is, the current copy is current—there is no need to copy its contents to disk again before it is removed.

If the Memory Manager succeeds in keeping the active pages of processes in RAM, then virtual memory addressing is largely transparent to User processes. If there is not enough RAM to hold the active pages of running processes, there are apt to be performance problems. If a running thread accesses a virtual memory address that is not currently backed by RAM, the hardware generates an interrupt signaling a page fault. The operating system must then resolve the page fault by accessing the page on disk, reading it into a free page in RAM, and then re-executing the failed instruction. The running thread that has encountered the page fault is placed in an involuntary Wait state for the duration of the page fault resolution process, including the time it takes to copy the page from disk into memory.

Page Fault Resolution

The most serious performance issues associated with virtual memory are the execution delays that programs encounter whenever they reference virtual memory locations that are not in the current set of memory-resident pages. This event is known as a *page fault*. A program thread that incurs a page fault is forced into an involuntary Wait state during *page fault resolution* for the entire time it takes the operating system to find the specific page on disk and restore it to physical memory.

When a program execution thread attempts to reference an address on a page that is *not* currently resident in physical memory, a hardware interrupt occurs that halts the executing program. If the page referenced is not currently resident in RAM, the instruction referencing it fails, creating an addressing *exception* that generates an interrupt. An operating system Interrupt Service Routine gains control following the interrupt and determines that the address referenced was valid, but that the page containing that address is not currently resident in RAM. The operating system then must remedy this situation by locating a copy of the desired page on secondary storage, issuing an I/O operation to the paging file, and copying the designated page from disk into a free page in RAM. Once the page has been copied successfully, the operating system re-dispatches the temporarily halted program, allowing the program thread to continue its normal execution cycle.

Note If a User program accesses an invalid memory location because of a logic error, for example, that references an uninitialized pointer, an addressing exception similar to a page fault occurs. The same hardware interrupt is raised. It is up to the Memory Manager's ISR that gets control following the interrupt to distinguish between the two situations.

The performance of User mode application programs suffers when there is a shortage of RAM and too many page faults occur. It is also imperative that page faults be resolved quickly so that page fault resolution does not delay the execution of program threads unduly.

Note For the sake of simplicity, the discussion of virtual memory addressing and paging in this chapter generally ignores the workings of the system file cache. The system file cache uses Virtual Memory Manager functions to manage application file data. The system file cache automatically maps open files into a portion of the system virtual address range and then uses the process working set memory management mechanisms discussed in this section to keep the most active portions of current files resident in physical memory. *Cache faults* in Windows Server 2003 are a type of page fault that occurs when an executing program references a section of an open file that is not currently resident in physical memory. Cache faults are resolved by reading the appropriate file data from disk or, in the case of a file stored remotely, accessing it across the network. On many file servers, the system file cache is one of the leading consumers of both virtual and physical memory.

Available Bytes pool The Windows Server 2003 Memory Manager maintains a pool of available (free) physical memory pages to resolve page faults quickly. Whenever the pool is depleted, the Memory Manager replenishes its buffer of available RAM by *trimming* older—that is, less frequently referenced—pages of active processes and writing these to disk. If available RAM is adequate, executing programs seldom encounter page faults that delay their execution, and the operating system has no difficulty maintaining a healthy supply of free pages. If the system is short on physical memory, high page fault rates can occur, slowing down the performance of executing programs considerably.

The operating system might be unable to maintain an adequate pool of available RAM if there is less physical memory than the workload requires. This is a situation that you can recognize by learning to interpret the memory performance statistics that are available.

Tip The primary indicator of a shortage of RAM is that the pool of Available Bytes, relative to the size of RAM, is too small. Just as important are the number of paging operations to and from disk, a Memory counter called Memory\Pages/sec. When physical memory is over-committed, your system can become so busy moving pages in and out of RAM that it is not accomplishing much in the way of real work.

Server applications that attempt to allocate as much RAM as possible further complicate matters. These programs, among them SQL Server and Exchange Server, attempt to grab as much RAM as they can. They communicate and coordinate with the Memory Manager to determine whether it is a good idea to try and allocate more memory for database buffers. When you are running these server applications, RAM should always look like it is almost full. The only way to tell that you could use more RAM on the system is to look inside these applications and see how effectively they are using the database buffers they have allocated.

Note Some people like to think of RAM as serving as a cache buffer for virtual memory addresses. Like most caching schemes, there is usually a point of diminishing return when adding more and more RAM to your system. All the virtual memory pages that processes create do not have to be resident in RAM concurrently for performance to be acceptable.

Performance considerations In a virtual memory computer system, some page fault behavior—for instance, when a program first begins to execute—is inevitable. You do not need to eliminate paging activity completely. You want to prevent *excessive* paging from impacting performance.

Several types of performance problems can occur when there is too little physical memory:

- **Too many paging operations to disk** Too many page faults that result in disk operations lead to excessive program execution delays. This is the most straightforward performance problem associated with virtual memory and paging. Unfortunately, it is also the one that requires the most intense data gathering to diagnose.
- **Disk contention** Virtual memory machines that sustain high paging rates to disk might also encounter disk performance problems. The disk I/O activity because of paging might contend with applications attempting to access their data files stored on the same disk as the paging file. The most common sign of a memory shortage is seeing disk performance suffer because of disk contention. Even though it is a secondary effect, it is often easier to recognize.

Tip The *Memory* Pages/sec counter reports the total number of pages being moved in and out of RAM. Compare the number Pages/sec to the total number of Logical Disk\Disk Transfers/sec for the paging file disk. If the disk is saturated and Pages/sec represents 20–50 percent or more of total Disk transfers/sec, paging is probably absorbing too much of your available I/O bandwidth.

- **A general physical memory shortage** User programs *compete* for access to available physical memory. Because physical memory is allocated to process virtual address spaces on demand, when memory is scarce, all running programs can suffer.

Note Memory does not get utilized like other shared computer resources. Unlike processors, disks, and network interfaces, it is not possible to measure memory request rates, service times, queue time, and utilization factors. For example, a page in RAM is either occupied or free. While memory is occupied, it is 100 percent utilized. While it is occupied, it is occupied *exclusively* by a virtual memory page from a single process address space. How long memory remains occupied depends on how often it is being accessed and what other memory accesses are occurring on the system. When memory locations are no longer active, they might still remain occupied for some time. None of these usage characteristics of memory is analogous to the way computer resources like processors, disks, and network interfaces are used.

A severe shortage of physical memory can seriously impact performance. Moving pages back and forth between disk and memory consumes both processing and disk capacity. A system forced to use too many CPU and disk resources on virtual memory management tasks and too few on the application workload is said to be *thrashing*. The image that the term thrashing conjures up is a washing machine so overloaded with clothes that it expends too much energy sloshing laundry around without succeeding in getting the clothes very clean. Troubleshooting memory bottlenecks is one of the topics discussed at length in Chapter 5, “Performance Troubleshooting.”

The solution to most paging problems is to install more physical memory capacity to reduce the amount of paging to disk that needs to occur. If you cannot add memory capacity immediately, you can take other effective steps to minimize the performance impact of an acute memory shortage. For instance, you might be able to reduce the number and size of processes that are running on the system or otherwise reduce the physical memory demands of the ones that remain. Because a memory shortage often manifests itself as disk contention, you can also attack the problem by improving disk performance to the paging file. Possible remedies include:

- Defining additional paging files across multiple (physical) disks
- Reducing disk contention by removing other heavily accessed files from the paging file physical disk or disks
- Upgrading to faster disks

These disk tuning strategies will speed up page fault resolution for the page faults that do occur or increase the effective disk bandwidth to allow the system to sustain heavier paging rates. Disk performance is discussed in greater detail later in this chapter in "The I/O Subsystem" section. Disk tuning strategies are discussed in depth in Chapter 5, "Performance Troubleshooting."

Committed Pages

The operating system builds page tables on behalf of each process that is created. A process's page tables get built on demand as virtual memory locations are accessed, potentially mapping the entire virtual process address space range. The Win32 *VirtualAlloc* API call provides both for *reserving* contiguous virtual address ranges and *committing* specific virtual memory addresses. Merely allocating virtual memory does not trigger building Page Table entries because you are not yet accessing the virtual memory address range to store data.

Reserving a range of virtual memory addresses is something your application might want to do in advance for a data file or other data structure that needs to be mapped into contiguous virtual storage addresses. Only later, when those virtual addresses are accessed, is physical memory allocated to allow the program access to those reserved virtual memory pages. The operating system constructs a Page Table entry to map the virtual address into RAM. Alternatively, a PTE points to the address of the page where it is stored on one of the paging files. The paging files that are defined allow virtual memory pages that will not all fit in RAM to spill over onto disk.

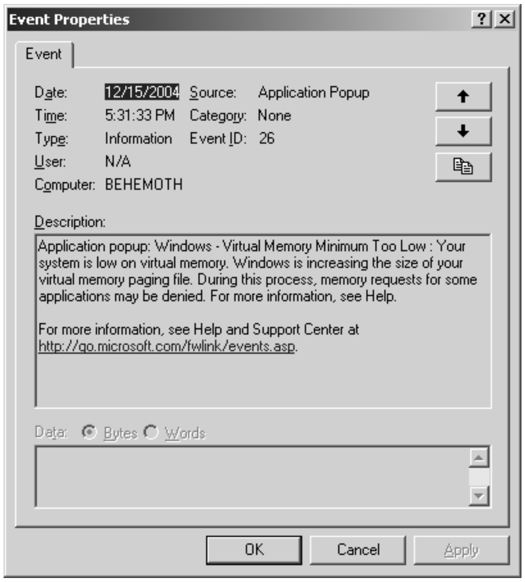
Committing virtual memory addresses causes the Virtual Memory Manager to ensure that the process address space requesting the memory will be able to access it. This is accomplished by charging the request against the system's commit limit. Any unreserved and unallocated process virtual memory addresses are considered free.

Commit Limit The *Commit Limit* is the upper limit on the total number of Page Table entries the operating system will build on behalf of all running processes. The virtual memory Commit Limit prevents the system from creating a virtual memory page that cannot fit somewhere in either RAM or the paging files.

Note The number of PTEs that can be built per process is limited by the width of a virtual address. For machines using a 32-bit virtual address, there is a 4-GB limit on the size of the process virtual address space. A 32-bit machine with a 4-bit Physical Address Extension (PAE) can be configured with more than 4 GB of RAM, but process virtual address spaces are still limited to 4 GB. Machines with 64-bit virtual addressing allow the operating system to build process address spaces larger than 4 GB. Windows Server 2003 builds process address spaces on 64-bit machines that can be as large as 16 TB. For more information about 64-bit addressing, see Chapter 6, "Advanced Performance Topics."

The Commit Limit is the sum of the amount of physical memory, plus the size of the paging files, minus some system overhead. When the Commit Limit is reached, a process can no longer allocate virtual memory. Programs making routine calls to *VirtualAlloc* to allocate memory will fail.

Paging file extension Before the Commit Limit is reached, Windows Server 2003 will alert you to the possibility that virtual memory could soon be exhausted. Whenever a paging file becomes nearly full, a distinctive warning message, shown in Figure 1-14, is issued to the console. Also generated is a System Event log message with an ID of 26 that documents the condition.



Date: 12/15/2004 Source: Application Popup
Time: 5:31:33 PM Category: None
Type: Information Event ID: 26
User: N/A
Computer: BEHEMOTH

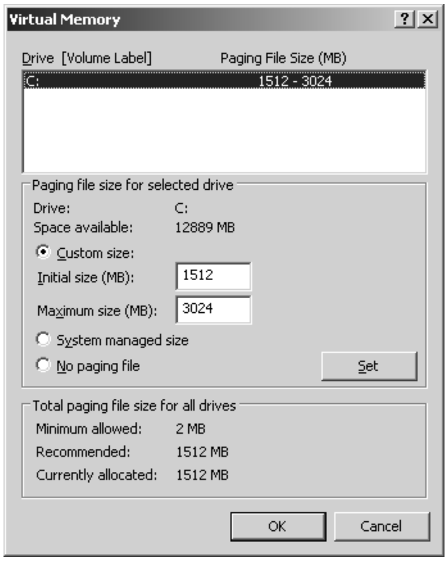
Description:

Application popup: Windows - Virtual Memory Minimum Too Low : Your system is low on virtual memory. Windows is increasing the size of your virtual memory paging file. During this process, memory requests for some applications may be denied. For more information, see Help.

For more information, see Help and Support Center at http://go.microsoft.com/fwlink/events.asp.

Figure 1-14 Out of Virtual Memory console error message

Following the instructions in the error message directs you to the Virtual Memory control (Figure 1-15) from the Advanced tab of the System item in Control Panel, where additional paging files can be defined or the existing paging files can be extended (assuming disk space is available and the page file does not already exceed its 4-GB upper limit). Note that this extension of the paging file occurs immediately while the system is running—it is not necessary to reboot the system.



C: 1512 - 3024

Paging file size for selected drive

Drive: C:

Space available: 12889 MB

(●) Custom size:

Initial size (MB): 1512

Maximum size (MB): 3024

( ) System managed size

( ) No paging file

Total paging file size for all drives

Minimum allowed: 2 MB

Recommended: 1512 MB

Currently allocated: 1512 MB

Figure 1-15 The Virtual Memory dialog for configuring the location and size of the paging files

Windows Server 2003 creates an initial paging file automatically when the operating system is first installed. The default paging file is built on the same logical drive where Windows Server 2003 is installed. The initial paging file is built with a minimum allocation equal to 1.5 times the amount of physical memory. It is defined by default so that it can extend to approximately two times the initial allocation.

The Virtual Memory dialog illustrated in Figure 1-15 allows you to set initial and maximum values that define a range of allocated paging file space on disk for each paging file created. When the system appears to be running out of virtual memory, the Memory Manager will automatically extend a paging file that is running out of space, has a range defined, and is currently not at its maximum allocation value. This extension, of course, is also subject to space being available on the specified logical disk. The automatic extension of the paging file increases the amount of virtual memory available for allocation requests. This extension of the Commit Limit might be necessary to keep the system from crashing.

Warning Windows Server 2003 supports a maximum of 16 paging files, each of which must reside on distinct logical disk partitions. Page files are named Pagefile.sys and are always created in the root directory of a logical disk. On 32-bit systems, each paging file can hold up to 1 million pages, so each can be as large as 4 GB on disk. This yields an upper limit on the amount of virtual memory that can be allocated, 16 × 4 GB, or 64 GB, plus whatever amount of RAM is installed on the machine.

Extending the paging file automatically might have some performance impact. When the paging file allocation extends, it no longer occupies a contiguous set of disk sectors. Because the extension fragments the paging file, I/O operations to disk might suffer from longer seek times. On balance, this potential performance degradation is far outweighed by availability considerations. Without a paging file extension occurring automatically, the system is vulnerable to running out of virtual memory and crashing.

Warning Although you can easily prevent paging files from being extended automatically by setting the Maximum Size of the paging file equal to its Initial Size, doing this is seldom a good idea. Allowing the paging file to be extended automatically can save the system from crashing. This benefit far outweighs any performance degradation that might occur. To maintain highly available systems, ensure that paging files and their extensions exceed the demand for committed bytes of your workload. See Chapter 3, "Measuring Server Performance," for a description of the Performance Monitor counters you should monitor to ensure you always have enough virtual memory space defined.

Please note that a fragmented paging file is not always a serious performance liability. Because your paging files coexist on physical disks with other application data files, some disk seek activity that moves the disk read/write head back and forth between the paging file and application data files is unavoidable. Extending the paging file so that there are noncontiguous segments, some of which might be in areas of the disk that are surrounded by application data files, might actually reduce overall average seek distances for paging file operations.

Multiple paging files Windows Server 2003 supports up to 16 paging files. Having multiple paging files has possible performance advantages. It provides greater bandwidth for disk I/O paging operations. In Windows Server 2003, you can define only one paging file per logical disk.

Caution The contents of physical memory are copied to the paging file located on the system root volume whenever the system creates a memory crash dump. To generate a complete diagnostic crash dump, the paging file located on the system root volume must be at least as large as the size of RAM. Reducing the size of the primary paging file to below the size of RAM or eliminating it altogether will prevent you from generating a complete crash dump.

To maximize disk throughput, try to define paging files on separate *physical* disks, if possible. The performance benefit of defining more than one paging file depends on being able to access multiple physical disks in parallel. However, multiple paging files configured on the same physical disk leads only to increased disk contention. Of course, what looks like a single physical disk to the operating system might in fact be an *array* of disks managed by a hardware disk array controller. Again, the rule of thumb of allocating no more than one paging file per physical disk usually applies.

Caution Because paging files often sustain a higher percentage of write than read operations, it is recommended that you avoid using RAID 5 disk arrays for the paging file, if possible. The RAID 5 write performance penalty makes this type of disk array a bad choice for a paging disk.

Clustered paging I/O Windows Server 2003 performs *clustered paging* file I/O. This feature might encourage you to configure multiple paging files. When a page fault occurs and the Memory Manager must retrieve it from the paging file, additional related pages are also copied into memory in the same operation. The rationale for clustered paging is simple. Individual disk I/O operations to the paging file are time-consuming. After spending considerable time positioning the disk arm over the correct disk sector, it makes sense to gather any related nearby pages from the disk in one continuous operation. This is also known as *prefetching* or *anticipatory paging*.

The reasoning behind anticipatory paging is similar to that which leads to your decision to add a few extra items to your shopping cart when you make an emergency visit to the supermarket to buy a loaf of bread and a carton of milk. Picking up a dozen eggs or a pound of butter at the same time might save you from making a second time-consuming visit to the same store later on. It takes so long (relatively speaking) to get to the disk in the first place that it makes sense for the Memory Manager to grab a few extra pages while it is at it. After all, these are pages that are likely to be used in the near future.

Anticipatory paging turns individual on-demand page read requests into bulk paging operations. It is a throughput-oriented optimization that tends to increase both disk and memory utilization rather than a response-oriented one. Clustered paging elongates page read operations. Because it is handling bulk paging requests, the paging file disk is busier for somewhat longer periods than it would be if it were just performing individual on-demand page read operations.

Having more time-consuming paging file requests normally does not affect the thread that is currently delayed waiting for a page fault to be resolved. Indeed, this thread might directly benefit from the Memory Manager correctly anticipating a future page access. However, *other* executing threads that encounter page faults that need to be resolved from the same paging file disk can be impacted. Having multiple paging files can help with this condition. With only one paging file, other threads are forced to wait until the previous bulk paging operation completes before the Memory Manager can resolve their page faults. Having a second (or third or fourth, and so on) paging file allocated increases the opportunity for paging file I/O parallelism. While one paging file disk is busy with a bulk paging operation, it might be possible for the Memory Manager to resolve a page fault quickly for a second thread—if it is lucky enough to need a page from a different paging file than the one that is currently busy.

Tip The Memory\Page Reads/sec counter reports the number of I/O operations to disk to resolve page faults. The Memory\Pages Input/sec counter reports the total number of pages that were read from disk during those operations. The ratio of Pages Input/sec to Page Reads/sec is the average number of pages fetched from disk per paging operation.

Process Virtual Address Spaces

The operating system constructs a separate virtual memory address space on behalf of each running process, potentially addressing up to 4 GB of virtual memory on 32-bit machines. Each 32-bit process virtual address space is divided into two equal parts, as depicted in Figure 1-16. The lower 2 GB of each process address space consists of private addresses associated with that specific process only. This 2-GB range of addresses refers to pages that can be accessed only by threads running in that process address space *context*. Each per process virtual address space can range from 0x0000 0000 to address 0x7fff ffff, spanning 2 GB, potentially. Each process gets its own unique set of user addresses in this range. Furthermore, no thread running in one process can access virtual memory addresses in the private range that is associated with a different process.

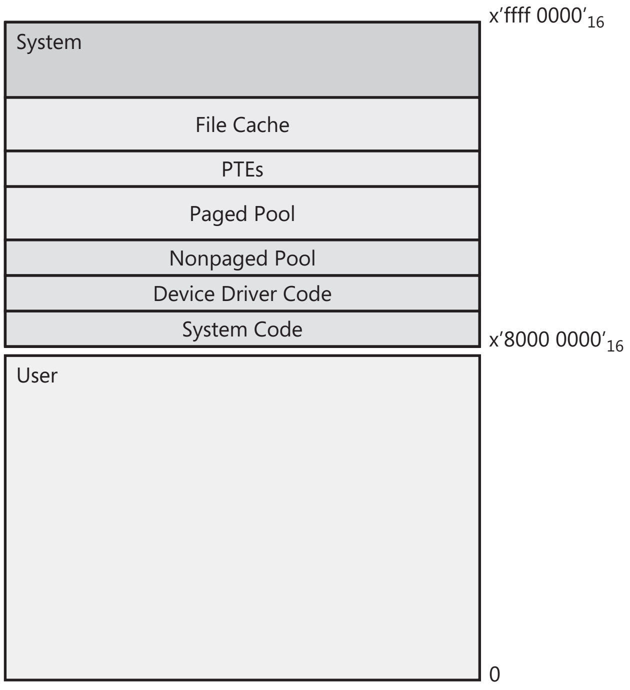


Figure 1-16 The 4-GB address space used in 32-bit systems

Because the operating system builds a unique address space for every process, Figure 1-17 is perhaps a better picture of what the User virtual address spaces look like. Notice that the System portion of each process address space is identical. One set of System PTEs—augmented by per process page tables and session space—maps the System portion of the process virtual address space for every process. Because System addresses are common to all processes, they facilitate high performance communication with the operating system functions and device drivers. These common addresses also offer a convenient way for processes to communicate with each other, when necessary.

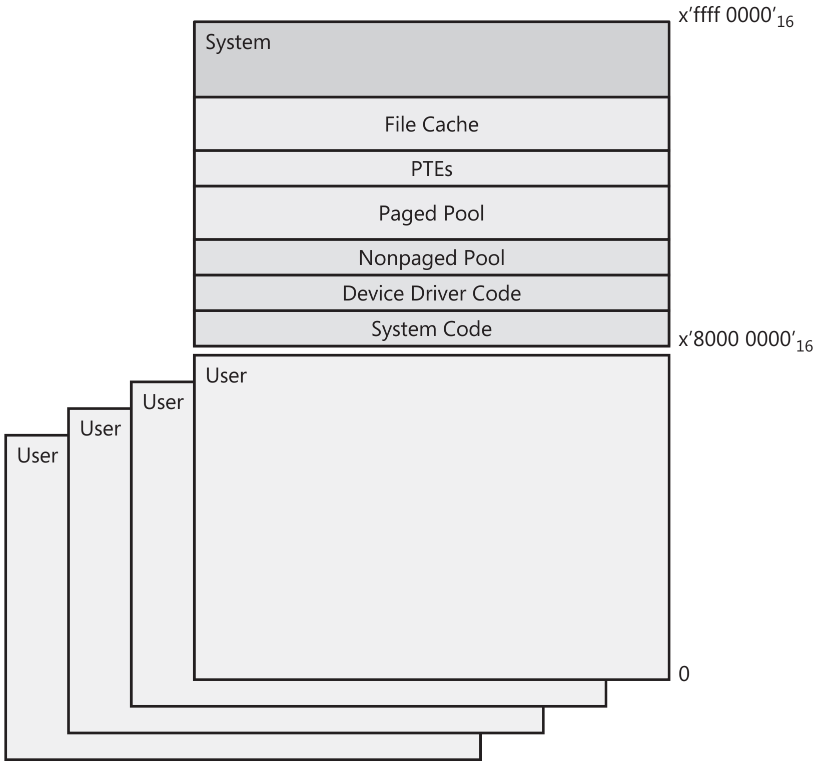


Figure 1-17 User processes share the System portion of the 4-GB virtual address space

Shared system addresses The upper half of each per-process address space in the range of '0x8000 0000' to '0xffff ffff' consists of system addresses common to *all* virtual address spaces. All running processes have access to the same set of addresses in the system range. This feat is accomplished by combining the system's Page Tables with each unique per process set of Page Tables.

However, User mode threads running inside a process cannot directly address memory locations in the system range because system virtual addresses are allocated using Privileged mode. This restricts memory access to addresses in the system range to kernel threads running in Privileged mode. This is a form of protection that restricts access to kernel memory to authorized kernel threads. When a User mode application thread calls a system function, the thread transfers to an associated Kernel mode address where its calling parameters are then checked. If the call is validated, the thread safely transitions to Kernel mode, changing its Execution state from User mode to Privileged. It is in this fashion that an application thread gains access to common system virtual memory addresses.

Commonly addressable system virtual memory locations play an important role in *interprocess communication*, or IPC. Win32 API functions can be used to allocate portions of commonly addressable system areas to share data between two or more distinct processes. For example, the mechanism that Windows Server 2003 uses to allow multiple process address spaces to access common modules, known as *dynamic-link libraries* (DLLs), utilizes this form of shared memory addressing. (DLLs are library modules that contain subroutines and functions that can be called dynamically at run time, instead of being linked statically to the programs that utilize them.)

Extended user virtual addressing Windows Server 2003 permits a different partitioning of user and system addressable storage locations using the */userva* boot switch. This extends the private User address range to as much as 3 GB and shrinks the system area to as little as 1 GB, as illustrated in Figure 1-18. When to use Extended User virtual addressing is a topic discussed in Chapter 6, "Advanced Performance Topics."

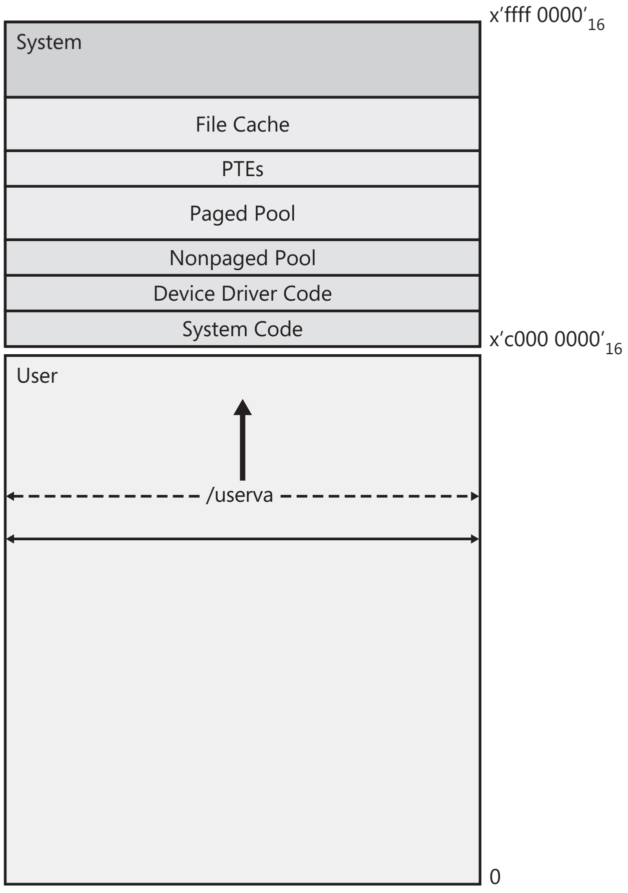


Figure 1-18 The /userva boot switch increases the size of the User virtual address range

Page Table Entries

During instruction execution, virtual addresses are translated into physical (real) memory addresses. This *virtual address translation* takes place inside the instruction execution pipeline internal to each processor. For example, during the Prefetch stage of a pipelined instruction execution, the pipeline translates the logical address of the next instruction to be executed, pointed to by the Program Counter (PC) register, into its corresponding physical address. Similarly, during the instruction Decode phases, virtual addresses pointing to instruction operands are translated into their corresponding physical addresses.

The precise mapping function that is used to translate a running program's virtual addresses into physical memory locations is hardware-specific. Hardware requirements specify the following:

- The mechanism that establishes the virtual address translation *context* for individual address spaces
- The format of the virtual-to-physical address translation tables used
- The method for notifying the operating system that page faults have occurred

Intel-compatible 32-bit processors have the specific hardware requirements for building and maintaining 32-bit page translation tables illustrated in this section. Other processor architectures that Windows Server 2003 supports are conceptually similar.

The processor architecture specifies the format of the page tables that the Windows Server 2003 operating system must build and maintain to enable the computer to perform virtual-to-physical address translation. The Intel 32-bit architecture (IA-32) specifies a two-level indexing scheme using a Page Directory, which then points to the Page Tables themselves. The Page Directory resides in a single 4-KB page and resides in memory while the process executes. The processor's internal Control Register 3 points to the origin of the Page Directory. Page Tables, also 4 KB in size, are built on demand as virtual memory locations are accessed. These consist of 32-bit Page Table entries that contain the physical memory address where the page of virtual addresses is currently mapped. Each Page Table can map 1024 4-KB pages (a 4-MB range), whereas the Page Directory can point to 1024 Page Tables. The combination supports the full 4-GB addressing scheme.

As required by the hardware, Windows Server 2003 builds and maintains one set of Page Tables capable of accessing the full 4-GB range of virtual addresses per process. Because each process is a separate and distinct address space, each execution thread inherits a specific address space context. A thread can access only virtual addresses associated with its specific process address space—with the exception of common system virtual addresses, which are accessible by any thread running in Privileged mode. Any code that attempts to access a memory location that is not valid for that process context encounters an invalid Page Table entry that causes an addressing exception—a page fault. The addressing exception occurs when the hardware attempts to translate the virtual address reference by an instruction to a physical one.

Page faults are processed by the operating system's Virtual Memory Manager, which then has to figure out whether they are the result of a programming bug or of accessing a page that is not currently in RAM. If the culprit is a programming bug, you will receive a familiar Access Violation message allowing you to attempt to debug the process before the operating system destroys it.

Being a repetitive task, virtual address translation can be sped up by buffering virtual address mapping tables in fast cache memory on board the processor chip. Like other computer architectures that support virtual memory, Intel-compatible processors provide hardware Translation Lookaside Buffers (TLBs) to speed up virtual address translation. When Control Register 3 is reloaded to point to a new set of per-process Page Tables, a context switch occurs, which has performance implications. A context switch flushes the TLB, slowing down instruction execution for a transitional period known as a *cache cold start*.

Memory status bits For a valid page, the high order 20 bits of the PTE reference the address of the physical memory location where the page resides. During virtual address translation, the processor replaces the high-order 20 bits of the virtual address with the 20 bits contained in the PTE to create the physical memory address. As illustrated in Figure 1-19, the Intel IA-32 hardware PTE also maintains a number of 1-bit flags that reflect the current status of the virtual memory page. Bit 0 of the PTE is the *present* bit—the valid bit that indicates whether the virtual address currently resides in physical memory. If bit 0 is set, the PTE is valid for virtual address translation, and the interpretation of the other bits is hardware-determined, as shown in Figure 1-19. If the present bit is not set, an Intel-compatible processor ignores the remainder of the information stored in the PTE.

The *status* bits in the PTE perform a variety of functions. Bit 2, for example, is an authorization bit set to prevent programs executing in User mode from accessing operating system memory locations allocated by kernel threads running in Privileged mode. It is called the *supervisor* bit. The *dirty* bit, which is bit 6, is set whenever the contents of a page are changed. The Memory Manager refers to the dirty bit during page replacement to determine whether the copy of the page on the paging file is current. Bit 5 is an *access* bit that the hardware sets whenever the page is referenced. It is designed to play a role in page replacement, and it is used for that purpose by the Virtual Memory Manager, as will be described shortly. Likewise, the Virtual Memory Manager turns off the *read/write* bit to protect code pages from being overwritten inadvertently by executing programs. The Virtual Memory Manager does not utilize the Intel hardware status bits 3 and 4, which are "hints" that are designed to influence the behavior of the processor cache. Windows Server 2003 does use 4-MB large pages to load sections of the operating system, which cuts down on the number of PTEs that need to be defined for the system virtual memory areas and saves space in the processor TLB.

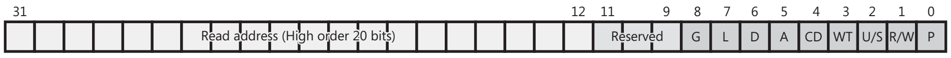


Figure 1-19 The format of an Intel 32-bit Page Table entry (PTE)

Invalid PTEs When the PTE bit 0 (the present bit) is not set, it is an invalid PTE, and the hardware ignores the remaining contents of the PTE. In the case of an invalid PTE, the operating system is free to use this space any way it sees fit. The Virtual Memory Manager uses the empty space in an invalid PTE to store the essential information about where a paged-out page can be located—in physical memory in the VMM Standby List, on the paging file, or, in the case of a file cache fault, in the file system. This information is stored in an invalid PTE, as shown in Figure 1-20.

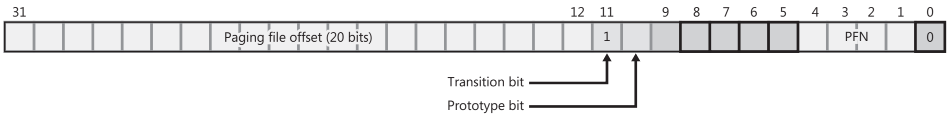


Figure 1-20 The format of an invalid 32-bit PTE

Invalid PTEs contain a paging file number (PFN) and a 20-bit offset to identify the exact location on disk where the page is stored. The paging file number is a 4-bit index that is used to reference up to 16 unique paging files. The PFN references the HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management Registry key, where information about the paging file configuration is stored in the PagingFiles field. The 20-bit paging file offset then references a page slot somewhere in that paging file. The Memory Manager maintains a *transition* bit that is used to identify a trimmed process page that is still resident in physical memory in the VMM Standby List. Its role in page replacement is discussed later.

Windows Server 2003 also uses invalid PTEs as build Prototype PTEs, identified in Figure 1-20 with the *prototype* bit set. The prototype PTE is the mechanism used for mapping shared memory pages into multiple process address spaces. DLLs exploit this feature. DLL modules are loaded once by the operating system into an area of commonly addressable storage backed by a real Page Table entry. When referenced by individual processes, the operating system builds a prototype PTE that points to the real PTE. Using the prototype PTE mechanism, a DLL is loaded into RAM just once, but that commonly addressable page can be shared and referenced by many different process virtual address spaces.

Prototype PTEs are also used by the built-in Windows Server 2003 file cache. The file cache is a reserved area of the system's virtual address space where application files are mapped.

Page Replacement

Following a policy of allocating physical memory page slots on demand as they are referenced inevitably fills up all available physical memory. A common problem virtual memory operating systems face is what to do when a page fault occurs, a valid page must be retrieved from the paging file, and there is little or no room left in physical memory for the referenced page. When physical memory is fully allocated and a new page is referenced, something has to give. The page replacement policy decides what to do when a new page is referenced and physical memory is full.

Tip You can watch physical memory filling up in Windows Server 2003 by monitoring Memory\Available Bytes. Available Bytes reports the amount of free, unallocated memory in RAM. It is a buffer of free pages that the Virtual Memory Manager maintains so that page faults can be resolved as rapidly as possible. As long as there are free pages in RAM, the Virtual Memory Manager does not have to remove an old page from a process working set first when a page fault occurs.

LRU A *Least Recently Used* (LRU) page replacement policy is triggered whenever the pool of Available Bytes is running low. LRU is a popular solution to the page replacement problem. The name, Least Recently Used, captures the overall flavor of the strategy. LRU tries to identify "older" pages and replace them with new ones, reflecting the current virtual memory access patterns of executing programs. Older pages, by inference, are less likely to be referenced again soon by executing programs, so they are the best candidates for page replacement.

When Available Bytes is low, the Virtual Memory Manager scans all the currently resident pages of each process's working set and identifies those that have not been referenced recently. It then trims the oldest pages from a process's working set and places them in the Available Bytes pool to replenish it.

The way the Windows Server 2003 page replacement policy works is illustrated in Figure 1-21.

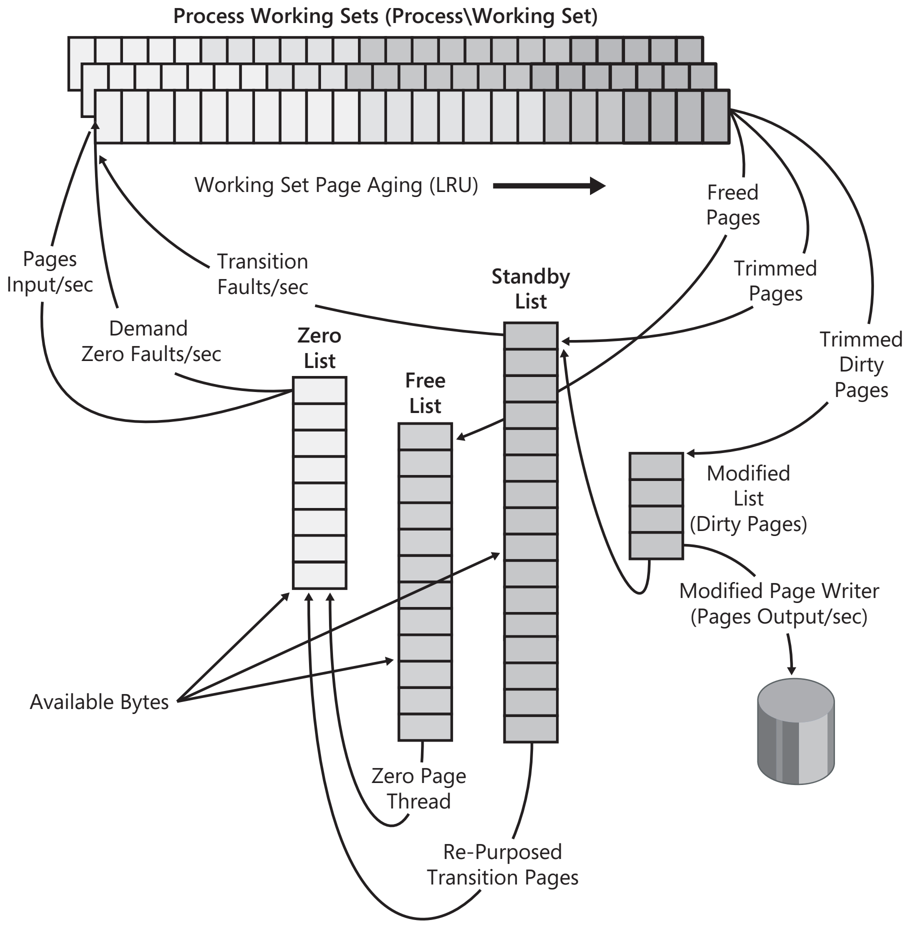

Process Working Sets (Process\Working Set)
Working Set Page Aging (LRU)
Pages Input/sec
Demand Zero Faults/sec
Transition Faults/sec
Freed Pages
Trimmed Pages
Trimmed Dirty Pages
Standby List
Zero List
Free List
Modified List (Dirty Pages)
Modified Page Writer (Pages Output/sec)
Available Bytes
Zero Page Thread
Re-Purposed Transition Pages

Figure 1-21 Process working set trimming in Windows Server 2003

The Available Bytes pool consists of three lists of available pages:

- **Zero List** Pages that are available immediately to satisfy new virtual memory allocation requests.
- **Free List** Pages previously allocated that were explicitly freed by the application. These are available to satisfy new virtual memory allocation requests only after they have been zeroed out for the sake of integrity.
- **Standby List** Contains recently trimmed process working set pages that are also eligible to satisfy new virtual memory allocation requests. Standby List pages still contain current data from the process working set that they were recently removed from. The PTE associated with a page on the Standby List has its transition bit set. A page fault that references a page on the Standby List can be resolved immediately without having to access the disk. This is called a *tran-*

sition fault, or *soft fault*. For this reason, the Standby List is also known as the VMM cache. If both the Zero List and Free List are depleted, a page on the Standby List can be migrated, or repurposed, to the Zero List.

Note Before dirty pages containing changed data can be removed from memory, the operating system must first copy their contents to the paging file.

The Virtual Memory Manager maintains information about the age of each page of each process working set. A page with its access bit in the PTE set is considered recently referenced. It should remain in the process working set. During its periodic scans of memory, VMM checks each page PTE in turn, checking whether the access bit is set, and then clearing the bit. (Processor hardware will turn the access bit on the next time the page is referenced during virtual address translation.) Pages without their access bits set are aged into three categories of old, older, and oldest pages. (The durations represented by these categories vary with memory pressure so that pages are moved to older categories more quickly when memory is tight.) VMM trims pages from working sets when there is a shortage of Available Bytes or a sudden drop in Available Bytes. When trimming, VMM makes multiple scans, taking oldest pages from each process's working set first, then taking newer pages, stopping when the Available Bytes pool is replenished. Trimmed pages are placed on the Standby List if they are unmodified (the dirty bit in their PTE is clear) and on the Modified List if they are dirty.

Trimmed process working set pages placed on the Standby List or the Modified List receive a second chance to be referenced by a process before they are replaced by new pages. PTEs for pages on the Standby List are marked invalid, but have their transition bits set, as illustrated in Figure 1-20. If any trimmed pages on the Standby List marked in transition are referenced again by a process's threads before they are repurposed and their contents overwritten, they are allowed to *transition fault* back into their process working set without the need to perform an I/O to the underlying file or to the paging file. Transition faults are distinguished from *hard page faults*, which must be satisfied by reading from the disk. Transition faults are also known as *soft faults* because VMM does not need to issue an I/O to recover their contents.

Trimmed pages that have the dirty bit in their PTE set are placed on the Modified List. Once this list grows to a modest size, the VMM schedules a modified page writer thread to write the current page contents to the paging file. This I/O can be performed very efficiently. After the paging file is current for a page, VMM clears the dirty bit and moves the page to the Standby List. As with other pages on the list, the page maintains its contents and still has its transition bit switched on, so it can be returned to a working set using the transition fault mechanism without additional I/O.

The size of the Standby List, the Free List, and the Zero List are added together and reported as Available Bytes. Pages from the Zero List are allocated whenever a process references a brand new page in its address space. Pages from either the Zero List or the Free List are used when a page is needed as a destination for I/O. The System Zero Page Thread zeroes out the contents of pages on the Free List and moves them to the Zero List, so the Free List tends to be quickly emptied. When there is a shortage of zeroed or free pages, pages at the front of the Standby List are repurposed. These pages at the front have been sitting on the list for the longest time, or their contents were deemed expendable when they were placed on the list (for example, pages used in a sequential scan of a large file).

Note Memory\Available Bytes counts the number of pages on the Standby List, the Free List, and the Zero List. There is no easy way to watch the size of these lists individually using Performance Monitor. The size of the Modified List is unreported, but it is presumed small because modified pages are written to the paging file (or file system) as soon as possible after they are trimmed.

The page trimming procedure is entirely threshold-driven. Page trimming is invoked as necessary whenever the pool of Available Bytes is depleted. Notice that there is no set amount of time that older pages will remain memory resident. If there is plenty of available memory, older pages in each process working set will be allowed to age gracefully. If RAM is scarce, the LRU page replacement policy will claim more recently referenced pages.

Measurement support Figure 1-21 also indicates the points where this process is instrumented. The Transition Faults/sec counter in the Memory object reports the rate at which so-called soft page faults occur. Similarly, Demand Zero Faults/sec reports the rate new pages are being created. Pages Output/sec shows the rate at which changed pages have been copied to disk. Pages Input/sec counts the number of pages from disk that the Memory Manager had to make room for in physical memory during the last measurement interval. By implication, because physical memory is a closed system,

Pages trimmed/sec + Pages freed = Transition Faults/sec + Demand Zero Faults/sec + Pages Input/sec, plus, any change in the size of the Available Bytes buffer from one interval to the next

Neither the rate of page trimming nor the rate at which applications free virtual memory pages is instrumented. These and other complications aside, such as the fact that a shared page that is trimmed could be subject to multiple page faults, those three Memory performance counters remain the best overall indicators of virtual memory management overhead.

The Page Faults/sec counter reports all types of page faults:

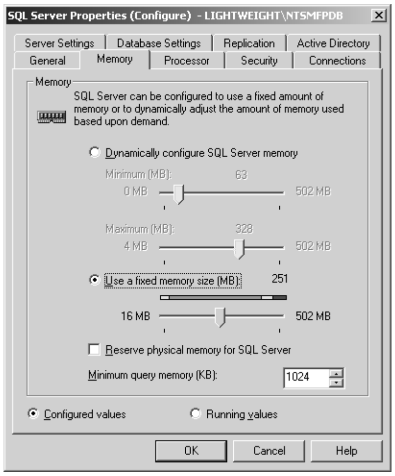
Memory\Page faults/sec = Memory\Transition Faults/sec + Memory\Demand Zero Faults/sec + Memory\Page Reads/sec

The Pages Read/sec counter corresponds to a *hard* page fault rate in this formula. Hard page faults require the operating system to retrieve a page from disk. Pages Input/sec counts hard page faults, plus the extra number of pages brought into memory at the time a page fault is resolved in anticipation of future requests.

Process Working Set Management

Windows Server 2003 provides application programming interfaces to allow individual processes to specify their physical memory requirements to the operating system and take the guesswork out of page replacement. Applications like Microsoft SQL Server, Exchange, and IIS utilize these memory management API calls. SQL Server, for example, implements the *SetProcessWorkingSetSize* Win32 API call to inform the operating system of its physical memory requirements. It also exposes a tuning parameter that allows the database administrator to plug in appropriate minimum and maximum working set values for the Sqlserver.exe process address space.

As illustrated in Figure 1-22, you can tell SQL Server to call *SetProcessWorkingSetSize* to set the process working set minimum and maximum values. The Windows Server 2003 Virtual Memory Manager will attempt to honor the minimum and maximum values you set, unless the system determines that there are not enough Available Bytes to honor that request safely.


SQL Server Properties (Configure) - LIGHTWEIGHT\NTSMFPDB

Memory

SQL Server can be configured to use a fixed amount of memory or to dynamically adjust the amount of memory used based upon demand.

( ) Dynamically configure SQL Server memory
Minimum (MB): 63 (0 MB – 502 MB)
Maximum (MB): 328 (4 MB – 502 MB)

(•) Use a fixed memory size (MB): 251 (16 MB – 502 MB)

[ ] Reserve physical memory for SQL Server

Minimum query memory (KB): 1024

(•) Configured values ( ) Running values

Figure 1-22 The SQL Server 2000 process working set management settings

The problem with controls like the *SetProcessWorkingSetSize* Win32 API call is that they are static. Meanwhile, virtual memory management is very much a dynamic process that adjusts to how various running programs are currently exercising memory. One potentially undesirable side effect of dynamic memory management is that the amount of memory one process acquires can affect what else is happening on the system. Setting fixed lower and upper limits on the number of physical pages an application can utilize requires eternal vigilance to ensure that the value you specified is the correct one.

Caution If you set the wrong value for the process working set in a control like the one in Figure 1-22, you can make the system run worse than it would have if you had given Windows Server 2003 the freedom to manage process working sets dynamically.

Because virtual memory usage is dynamic, a control (for example, like the one illustrated in Figure 1-22 for SQL Server) to set a suitable minimum and maximum range for an application is often more helpful. If you click the **Dynamically configure SQL Server memory** option, you can set a minimum and maximum working set that is appropriate for your workload. This setting instructs dynamic memory management routines of Windows Server 2003 to use a working set range for SQL Server that is more appropriate than the system defaults.

Being inside a process like Sqlserver.exe is not always the best vantage point to understand what is going on with a global LRU page placement policy. Windows Server 2003 provides a feedback mechanism for those processes that are interested in controlling the size of their working sets. This feedback mechanism is particularly helpful to processes like SQL Server, IIS, and Exchange, which utilize large portions of the User virtual address space to store files and database buffers. These applications perform their own aging of these internal cache buffer areas.

Processes can receive notifications from the Memory Manager on the state of free RAM. These processes receive a *LowMemoryResourceNotification* when Available Bytes is running low and a *HighMemoryResourceNotification* when Available Bytes appears ample. These are global events posted by the Memory Manager to let processes that register for these notifications know when the supply of free memory is rich and they can help themselves to more. Processes are also notified when available memory is depleted and they should return some of their older working set pages to the system. Processes that register to receive these notifications and react to them can grab as much RAM as they need and still remain good citizens that will not deliberately degrade system performance for the rest of the application processes running on the machine.

Accounting for process memory usage The operating system maintains a Page Frame Number (PFN) list structure that accounts for every page in physical memory and how it is currently being used. Physical memory usage statistics are gathered by traversing the PFN. The number of active pages associated for a given process is reported as the Process\Working Set bytes. The process working set measurements are mainly used to determine which processes are responsible for a physical memory shortage.

On behalf of each process virtual address space, the operating system builds a set of Virtual Address Descriptors (VADs) that account for all the virtual memory a process has reserved or committed. Tabulating the virtual memory committed by each process by reading the VADs leads to the Process\Virtual Bytes measurements. A process's current allocations in the common system Paged and Nonpaged pools are also counted separately. These two pools in the system range are discussed in more detail later in this chapter. The process virtual memory allocation counters are very useful if you are tracking down the source of a memory leak, as illustrated in Chapter 5, "Performance Troubleshooting."

Shared DLLs Modular programming techniques encourage building libraries containing common routines that can be shared easily among running programs. In the Microsoft Windows programming environment, these shared libraries are known as dynamic-link libraries (DLLs), and they are used extensively by Microsoft developers and other developers. The widespread use of shared DLLs complicates the bookkeeping that Windows Server 2003 performs to figure out how many resident pages are associated with each process working set.

Windows Server 2003 counts *all* the resident pages associated with shared DLLs as part of *every* process working set that has the DLL loaded. All resident pages of the DLL, whether or not the process has recently accessed them, are counted in the process working set. In Windows Server 2003, the Process Working Set Bytes counter includes all resident pages of all shared DLLs that the process currently has loaded. This has the effect of charging processes for resident DLL pages they might never have touched, but at least this double counting is performed consistently across all processes that have the DLL loaded.

This working set accounting procedure that also spawns the measurement data is designed to enable VMM to do a good job when it needs to trim pages. Unfortunately, it does make it difficult to account precisely for how physical memory is being used. It leads to a measurement anomaly that is illustrated in Figure 1-23. For example, because the resident pages associated with shared DLLs are included in the process working set, it is not unusual for a process to acquire a working set larger than the number of committed virtual memory bytes it has requested. Notice the number of processes in Figure 1-23 with more working set bytes (the Mem Usage column) than committed virtual bytes (the VM Size column). Because DLLs are files that are read into memory directly from the file system, few working set pages associated with shared DLLs ever need to be committed to virtual memory. They are not included in the Process Virtual Bytes counter even though all the resident bytes associated with them are included in the Process Working Set counter.

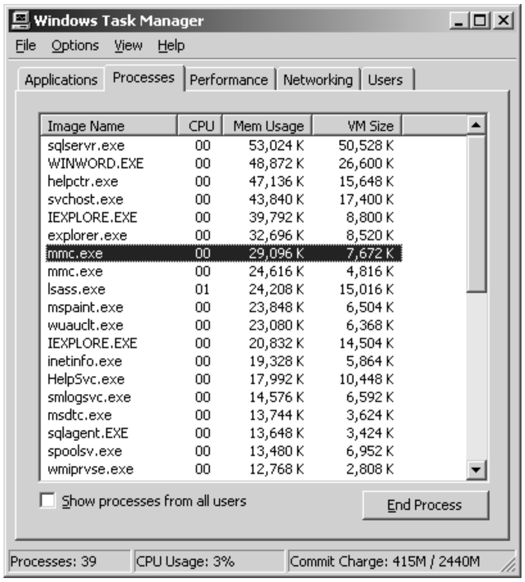

Windows Task Manager
File Options View Help
Applications Processes Performance Networking Users
Show processes from all users
End Process

Image Name	CPU	Mem Usage	VM Size
sqlservr.exe	00	53,024 K	50,528 K
WINWORD.EXE	00	48,872 K	26,600 K
helpctr.exe	00	47,136 K	15,648 K
svchost.exe	00	43,840 K	17,400 K
IEXPLORE.EXE	00	39,792 K	8,800 K
explorer.exe	00	32,696 K	8,520 K
mmc.exe	00	29,096 K	7,672 K
mmc.exe	00	24,616 K	4,816 K
lsass.exe	01	24,208 K	15,016 K
mspaint.exe	00	23,848 K	6,504 K
wuauclt.exe	00	23,080 K	6,368 K
IEXPLORE.EXE	00	20,832 K	14,504 K
inetinfo.exe	00	19,328 K	5,864 K
HelpSvc.exe	00	17,992 K	10,448 K
smlogsvc.exe	00	14,576 K	6,592 K
msdtc.exe	00	13,744 K	3,624 K
sqlagent.EXE	00	13,648 K	3,424 K
spoolsv.exe	00	13,480 K	6,952 K
wmiprvse.exe	00	12,768 K	2,808 K

Processes: 39   CPU Usage: 3%   Commit Charge: 415M / 2440M

Figure 1-23 Working set bytes compared to virtual bytes

System working set Windows Server 2003 operating system functions also consume RAM. Consequently, the system has a working set that needs to be controlled and managed like any other process. In this section, the components of the system working set are discussed.

Both system code and device driver code occupy memory. In addition, the operating system allocates data structures in two areas of memory: a pool for nonpageable storage and a pageable pool. Data structures that are accessed by operating system and driver functions when interrupts are disabled *must* be resident in RAM at the time they are referenced. These data structures are usually allocated from the nonpageable pool so that they reside permanently in RAM. The Pool Nonpaged Bytes counter in the Memory object shows the amount of RAM currently allocated in this pool that is permanently resident in RAM.

Generally, though, most system data structures are pageable—they are created in a pageable pool of storage and are subject to page replacement like the virtual memory pages of any other process. Windows Server 2003 maintains a working set of active pages in RAM for the operating system that are subject to the same LRU page replacement policy as ordinary process address spaces. The Pool Paged Bytes counter reports the amount of paged pool virtual memory that is allocated. The Pool Paged Resident Bytes counter reports on the number of page pool pages that are currently resident in RAM.

Tip The Memory\Cache Bytes counter reports the total number of resident pages in the current system working set. Cache Bytes is the sum of the System Cache Resident Bytes, System Driver Resident Bytes, System Code Resident Bytes, and Pool Paged Resident Bytes counters. The operating system's working set became known as the *Cache* because it also includes resident pages of the built-in file cache, the operating system function that historically consumed more RAM than any other.

The system virtual address range is limited to 2 GB, and by using the /3 GB boot option, it can be limited even further to as little as 1 GB. On a large 32-bit system, it is not uncommon to run out of virtual memory in the system address range. The culprit could be a program that is leaking virtual memory from the Paged pool. Alternatively, it could be caused by active usage of the system address range by a multitude of important system functions—kernel threads, TCP session data, the file cache, or many other normal functions. When the number of free System PTEs reaches zero, no function is able to map additional virtual memory within the system range. When you run out of system virtual memory addresses, the results are usually catastrophic.

The 2-GB limit on the size of the system virtual address range is a serious constraint that can sometimes be relieved only by moving to a 64-bit machine. There are examples of how to determine whether your system is running out of system virtual memory in Chapter 5, "Performance Troubleshooting." Chapter 6, "Advanced Performance Topics," discusses the virtual memory boot options, the Physical Address Extension, the Memory Manager Registry settings that influence how the system address space is allocated, and 64-bit virtual addressing.

Tip Tracking the Memory\Free System Page Table Entries counter can help you tell when the system virtual address range is going be exhausted. Unfortunately, you can sometimes run out of virtual addressing space in the Paged or Nonpaged pools before all the System PTEs are used up.

The I/O Subsystem

One of the key components of the Windows Server 2003 Executive is the I/O Manager. The I/O Manager provides a set of interfaces for applications that need to retrieve data from or store data to external devices. Because the contents of RAM are volatile, any computing results that need to be stored permanently must be stored on a disk drive or other external device. These Input/Output interfaces are consistent across all physical and logical devices. The I/O Manager also provides services that device drivers use to process the I/O requests. Figure 1-24 illustrates the relationship between User mode applications, the I/O Manager, and the device drivers under it, all the way down to the physical devices.

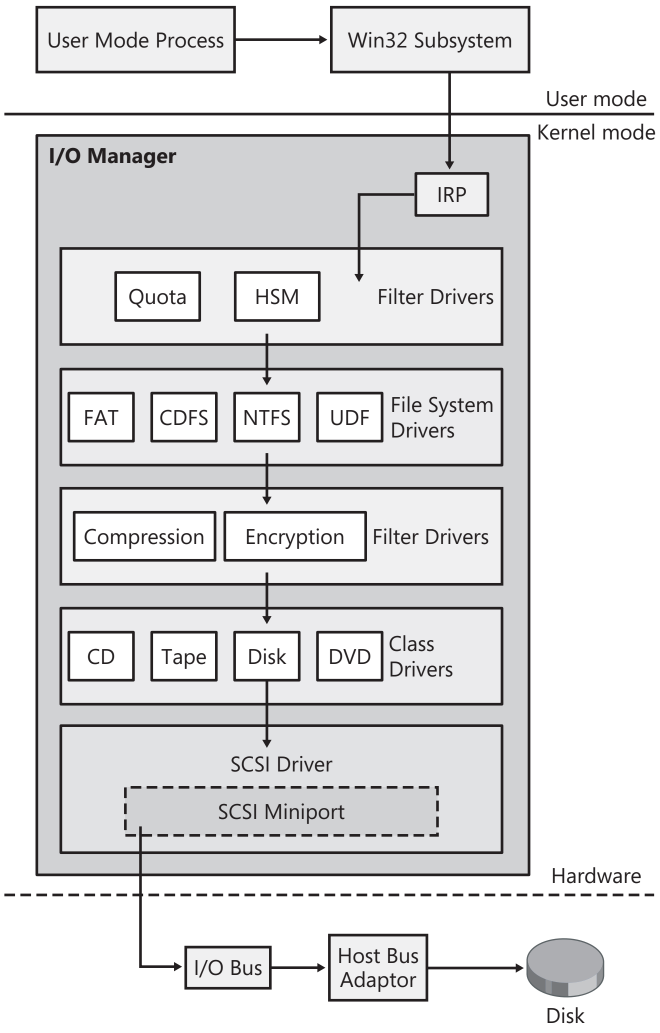


Figure 1-24 The I/O Manager

When it is called by a User mode process to perform an I/O operation against a file, the Win32 Subsystem creates an I/O Request Packet (IRP) that encapsulates the request. The IRP is then passed down through various layers of the I/O Manager for processing. The IRP proceeds down through the file system layer, where it is mapped to a physical disk location, to the physical disk device driver that generates the appropriate hardware command, and, finally, to the device itself. The I/O Manager handles all I/O devices, so a complete picture would include the driver components for the other types of devices such as network drivers, display drivers, and multimedia drivers. For the purpose of this section, it will be sufficient to concentrate on disk I/O processing.

More Info More complete documentation about the I/O Manager can be found in the Windows Device Model reference manual available at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/kmarch/hh/kmarch/wdmintro_d5b4fea2-e96b-4880-b610-92e6d96f32be.xml.asp.

Filter drivers are modules that are inserted into the I/O Manager stack at various points to perform optional functions. Some like the disk volume Quota Manager can process IRPs before they reach the file system driver. Others like the Compression and Encryption services only process requests that originated specifically for the NTFS driver.

Eventually, an IRP is passed down to the physical device driver layer, which generates an appropriate command to the hardware device. Because disk hardware is usually the most critical element of disk performance, beginning there is appropriate. Disk performance is a complex subject because of the range of hardware solutions with different cost/performance characteristics that you might want to consider. The discussion here is limited to the performance of simple disk configurations. Cached disks and disk array alternatives are discussed in Chapter 5, “Performance Troubleshooting.”

Disk Performance Expectations

The performance of disk drives and related components such as I/O buses is directly related to their hardware feeds and speeds. Using performance monitoring, you can determine how the disks attached to your Windows Server 2003 machine are performing. If the disks are performing at or near the performance levels that can be expected from the type of hardware devices you have installed, there is very little that can be accomplished by trying to “tune” the environment to run better. If, on the other hand, actual disk performance is considerably worse than reasonable disk performance expectations, mounting a tuning effort could prove very worthwhile. This section will provide a basic framework for determining what performance levels you can reasonably expect from different kinds of disk hardware. This topic is discussed in greater detail in Chapter 5, “Performance Troubleshooting.”

Because so many disk hardware and configuration alternatives are available, this complex subject can only be introduced here. Determining precisely what performance level your disk configuration is capable of delivering is something you are going to have to investigate on your own. The suggestions and guidelines introduced here should help in that quest.

After you determine what performance level your disks are capable of delivering, you need to understand the actual performance level of your disks. This requires knowing how to calculate the service time and queue time of your disks. Once you calculate the actual disk service time, you can compare it to the expected level of performance to see whether you have a configuration or tuning problem. In Chapter 5, "Performance Troubleshooting," the steps for relieving a disk bottleneck are discussed in detail. Here the scope of the discussion is limited to the essential background you need to understand and execute those procedures.

Disk Architecture

Figure 1-25 illustrates the architecture of a hard disk. Disks store and retrieve digitally encoded data on *platters*. A disk *spindle* normally consists of several circular platters that rotate continuously. Data is encoded and stored on both sides of each of the disk platters. Bits are stored on the platter magnetically on data *tracks*, arranged in concentric circles on the platter surface. The smallest unit of data transfer to and from the disk is called a *sector*. The capacity of a sector is usually 512 bytes. The *recording density* refers to the number of bits per square inch that can be stored and retrieved. Because the recording density is constant, outer tracks can store much more data than inner tracks. Data on the disk is addressed using a relative sector number. Data is stored and retrieved from the platters using the recording and playback *heads* that are attached at the end of each of the *actuator* arms.

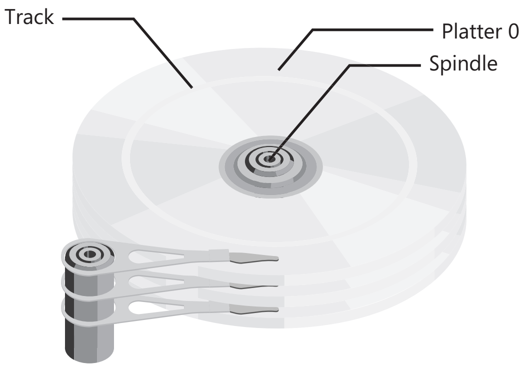


Figure 1-25 The components of a disk drive

An I/O command initiates a request to read or write a designated sector on the disk. The first step is to position the heads over the specific track location where the sector is located. This is known as a *seek*, which can be relatively time-consuming. Seek time is primarily a function of distance. If the heads are already positioned in the right place, there is a *zero motion seek*. Zero seeks occur, for example, when you read consecutive sectors off the disk. This is something that happens when a file is accessed sequentially (unless the file happens to be heavily *fragmented*). A *minimum seek* is the time it takes to reposition the heads from one track to an adjacent track. A *maximum seek* traverses the length of the platter from the first track to the last. An *average seek* is calculated as the time it takes to move across 1/3 of the available tracks.

Following the seek operation, another mechanical delay occurs to wait for the designated sector to rotate under the playback and recording head. This is known as *rotational delay*, or sometimes just *latency*. Device latency is one of those dumb luck operations. If your luck is good, the desired sector is very close to the current position of the recording heads. If your luck is bad, the desired sector is almost a full revolution of the disk away. On average, device latency is 1/2 of the device's rotational speed, usually specified in revolutions per minute (rpm). A disk that is spinning at 7200 rpm revolves 120 times per second, or once every 8.33 milliseconds. The average rotational delay for this disk is 1/2 a revolution, or approximately 4.2 ms.

The third major component of disk I/O service time is *data transfer* time. This is a function of the recording density of the track and the rotational speed of the device. These mechanical characteristics determine the data rate at which bits pass under the read/write heads. Outer tracks with roughly double the number of bits as inner tracks transfer data at twice the data rate. On average, any specific sector could be located anywhere on the disk, so it is customary to calculate an average data rate halfway between the highest and lowest speed tracks. The other variable factor in data transfer is the size of the request, which is normally a function of the block size selected for the file system. Some applications (like the paging subsystem) make bulk requests where they attempt to read or write several logical blocks in a single, physical disk operation.

The service time of a disk request can normally be broken into these three individual components:

disk service time = seek + latency + data transfer

There are device service time components other than these three, including protocol delay time. But because these other components represent only minor delays, they are ignored here for the sake of simplicity.

The specifications for several current, popular disk models are shown in Table 1-3. The service time expectation calculated here is based on the assumption that disks perform zero seeks 50 percent of the time and average seeks the other 50 percent. It also assumes a block size in the range of 4 KB–16 KB, which is typical for NTFS.

Table 1-3 Disk Service Time Expectations for a Representative Sample of Current Disks

Model	Average Seek (ms)	Rpm	Latency (ms)	Transfer Rate (MB/sec)	Service Time (ms)
Desktop disk	9	5400	5.5	45	11
Server disk	7.5	7200	4.2	50	9
Performance disk	6	10,000	3.0	60	7
High performance disk	5	15,000	2.0	75	5

Notice that the worst performing disks in this example are still capable of servicing requests in roughly 10 milliseconds. At 100 percent utilization, these disks are capable of about 100 I/O operations per second (IOPS). More expensive performance disks are capable of performance at about twice that rate.

A simple chart like the one in Table 1-3 should give you some insight into the bandwidth (or I/O capacity) of your disk configuration. Optimizations such as cached disks aside for the moment, the critical factor constraining the performance of your disk configuration is the number of independent physical disk spindles, each with its own finite capacity to execute I/O operations. If your server is configured with a single physical disk, it is probably capable of performing only 100–200 I/Os per second. If five disks are configured, the same computer can normally perform 5 times the number of disk I/Os.

This chart does not say anything about disk response time—in other words, service time plus queue time. Average disk response time is one of the important metrics that the I/O Manager supplies. The counter is called Avg. Disk sec/Transfer and is provided for both Logical and Physical Disks. At high utilization levels, queuing delays waiting for the disk are liable to be substantial. However, the number of concurrent disk requestors, which serves as an upper limit on the disk queue depth, is often small. The number of concurrent disk requestors usually establishes an upper bound on disk queue time that is much lower than a simple queuing model like M/M/1 would predict.

To set reasonable service time expectations for the disks you are using, access the public Web site maintained by the manufacturer that supplies your disks. There you can obtain the specifications for the drives you are running—the average seek time, the rotational speed, and the minimum and maximum data transfer rates the device supports. Failing that, you can also determine for yourself how fast your disks are capable of running by using a stress-testing program.

To utilize the disk service time expectations in Table 1-3 for planning purposes, you need to be able to compare the expected values to actual disk service times in your environment. To understand how to calculate disk service time from these measurements, it will help to learn a little more about the way disk performance statistics are gathered.

Disk Performance Measurements

The Physical Disk and Logical Disk performance statistics in Windows Server 2003 are gathered by the functional layers in the I/O Manager stack. Figure 1-26 shows the volume manager layer underneath the file system layer that is used to gather Logical Disk statistics. The physical disk partition manager, Partmgr.sys, gathers Physical Disk statistics.

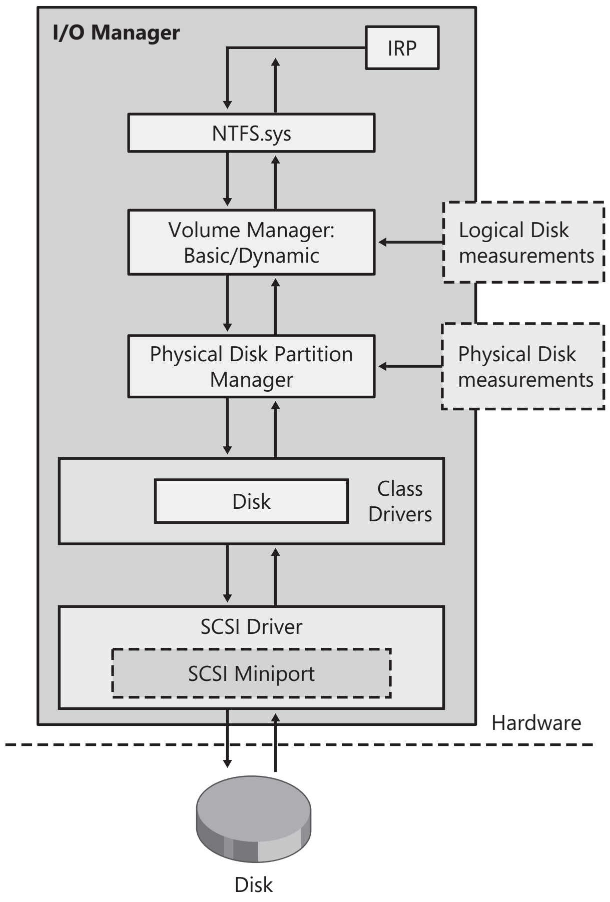

I/O Manager
IRP
NTFS.sys
Volume Manager: Basic/Dynamic
Logical Disk measurements
Physical Disk Partition Manager
Physical Disk measurements
Disk
Class Drivers
SCSI Driver
SCSI Miniport
Hardware
Disk

Figure 1-26 Disk performance statistics are gathered by the I/O Manager stack

Important Both Logical and Physical Disk statistics are enabled by default in Windows Server 2003. This is a change from Windows 2000 and Windows XP where only the Physical Disk statistics were installed by default.

Unlike the % Processor Time measurements that Windows Server 2003 derives using sampling, the disk performance measurements gathered by the I/O Manager reflect precise timings of individual disk I/O requests using the High Precision clock. As each IRP passes through the *measurement layer*, the software gathers information about the request. The DISK_PERFORMANCE structure definition, documented in the platform SDK, shows the metrics that the I/O Manager stack keeps track of for each individual request. They include the metrics shown in Table 1-4.

Table 1-4 Metrics Tracked by the I/O Manager

Metric	Description
BytesRead	Number of bytes read
BytesWritten	Number of bytes written
ReadTime	Time it took to complete the read
WriteTime	Time it took to complete the write
IdleTime	Specifies the idle time
ReadCount	Number of read operations
WriteCount	Number of write operations
QueueDepth	Depth of the queue
SplitCount	Number of split I/Os. Usually an indicator of file fragmentation

Table 1-4 is the complete list of metrics that the I/O Manager measurement layer compiles for each disk I/O request. These are the metrics that are then summarized and reported at the Logical or Physical Disk level. All the disk performance statistics that are available using Performance Monitor are derived from these basic fields. For instance, the Avg. Disk Queue Length counter that is available in Performance Monitor is derived using Little's Law as follows:

$$\textit{Average Disk Queue Length} = (\textit{ReadCount} \times \textit{ReadTime}) + (\textit{WriteCount} \times \textit{WriteTime})$$

Caution The Avg. Disk Queue Length counter is derived using Little's Law and not measured directly. If the Little's Law equilibrium assumption is not valid for the measurement interval, the interpretation of this value is subject to question. Any interval where there is a big difference in the value of the Current Disk Queue Length counter compared to the previous interval is problematic.

Stick with the metrics that *the I/O Manager measurement layer* measures directly and you cannot go wrong!

The I/O Manager measurement layer derives the *ReadTime* and *WriteTime* timing values by saving a clock value when the IRP initially arrives on its way down to the physical device, and then collecting a second timestamp after the disk I/O completes on its return trip back up the I/O Manager stack. The first clock value is subtracted from the second value to calculate *ReadTime* and *WriteTime*. These timings are associated with the Avg. Disk sec/Read and Avg. Disk sec/Write counters that are visible in Performance Monitor. Avg. Disk sec/Transfer is the weighted average of the Read and Write. They are measurements of the round trip time (RTT) of the IRP to the disk device and back.

Avg. Disk sec/Transfer includes any queuing delays at lower levels of the I/O Manager stack and, of course, at the device. To break down disk response time into service time and queue time, it helps to understand how disk Idle time is measured. The *Queue-Depth* is simply an instantaneous count of the number of active IRPs that the filter driver is keeping track of. These are IRPs that the filter driver has passed down on their way to the device, but have not returned yet. It includes any I/O requests that are currently executing at the device, as well as any requests that are queued waiting for service.

When an IRP is being passed upward following I/O completion, the I/O Manager measurement layer checks the current *QueueDepth*. If *QueueDepth* is zero, the I/O Manager measurement layer stores a clock value indicating that an Idle period is beginning. The disk idle period ends when the very next I/O Request Packet arrives at the I/O Manager measurement layer. When there is work to be done, the device is busy, not idle. The I/O Manager measurement layer accesses the current time, subtracts the previous timestamp marking the start of the Idle period, and accumulates the total Idle time measurement over the interval.

Even though Performance Monitor only displays a % Idle Time counter, it is more intuitive to calculate:

disk utilization = 100 –% Idle Time

Applying the Utilization Law, you can then calculate disk service time:

disk service time = disk utilization ÷ Disk Transfers/sec

With disk service time, you can then calculate average disk queue time:

disk queue time = Avg. Disk sec/Transfer –disk service time

Once you calculate the disk service time in this fashion for the devices attached to your machine, you can compare those measurements to your expectations regarding the levels of performance these devices can deliver. If actual disk performance is much worse than expected, you have a disk performance problem worth doing something about. Various configuration and tuning options for improving disk response time and throughput are discussed in Chapter 5, “Performance Troubleshooting.”

Network Interfaces

Networking refers to data communication between two or more computers linked by some transmission medium. Data communication technology is readily broken into *local area network* (LAN) technology, which is designed to link computers over limited distances, and *wide area network* (WAN) technology for communicating over longer distances.

LANs utilize inexpensive wire and wireless protocols suitable for peer-to-peer communication, making it possible to link many computers together cost-effectively. LAN technologies include the popular Fast Ethernet 100baseT standard and Gigabit Ethernet, FDDI, and Token Ring. An unavoidable consideration in wiring your computers together is that LAN technologies have a built-in distance constraint that must be honored—that is why they are referred to as *local* area networks. The Fast Ethernet protocol, for example, cannot be used to connect computers over distances greater than a hundred meters. Wireless LANs also have very stringent distance limitations. The set of localized connections associated with a single Ethernet hub or switch is known as a *network segment*. As you add more connections to a LAN or try to interconnect machines over greater distances, inevitably you will create more network segments that then must be *bridged* or *routed* to form a cohesive network.

WAN connections link networks and network segments over longer distances. Eventually, WANs connect to the backbone of the World Wide Web, which literally interconnect millions of individual computers scattered around the globe. Wide area networking utilizes relatively expensive long distance lines normally provided by telephone companies and other common carriers to connect distant locations. Popular WAN technologies include Frame Relay, ISDN, DSL, T1, T3, and SONET, among others.

The networking services Windows Server 2003 uses are based on prevailing industry standards. The spread of the Internet has led to universal adoption of the Internet communication protocols associated with TCP/IP. Windows Server 2003 is designed to operate with and is fully compliant with the bundle of networking standards associated with the Internet. These Internet protocols include UDP, TCP, IP, ICMP, DNS, DHCP, HTTP, and RPC. Instead of this alphabet soup, the suite of Internet standard protocols is often simply called TCP/IP, the two components that play a central role. The full set of TCP/IP protocols is the native networking language of the Windows Server 2003 operating system.

Packets

Data is transmitted over a communications line in a serial fashion, one bit at a time. Instead of simply sending individual bits between stations across the network, however, data communication is performed using groups of bits organized into distinct datagrams, or *packets*. It is the function of the data communications hardware and software that you run to shape bit streams into standard, recognizable packets. The overall shape of the packets that are being sent and received in Microsoft Windows-based networks is discussed here from a performance and capacity planning perspective.

More Info For more in-depth information about TCP/IP protocols, refer to Microsoft Windows Server 2003 TCP/IP Protocols and Services Technical Reference (Microsoft Press, 2003).

The Network Monitor is the diagnostic tool that allows you to capture packet traces in Windows Server 2003. A Network Monitor example is shown here to illustrate some common types of packets that you can expect to find circulating on your network. Using the Network Monitor to capture and examine network traffic is discussed in more detail in Chapter 2, "Performance Monitoring Tools."

At the heart of any packet is the *payload*, the information that is actually intended to be transmitted between two computers. Networking hardware and software inserts packet *headers* at the front of the data transmission payload to describe the data being transmitted. For instance, the packet header contains a tag that shows the type and format of the packet. The header also contains the *source address* of the station transmitting the data and the *destination address* of the station intended to receive it. In addition, the packet header contains a length code that tells you how much data it contains—remember, coming across the wire, the data appears as a continuous sequence of bits.

Mining these packet header fields, you can calculate who is sending how much data to whom, information that can then be compared to the capacity of the links connecting those stations to determine whether network link capacity is adequate. The information contained in the packet headers forms the basis of the networking performance statistics that you can gather using Performance Monitor.

Understanding the packet-oriented nature of data communication transmissions is very important. The various network *protocols* determine the format of data packets—how many bits in the header, the sequence of header fields, and how error correction code data is created and stored in the packet. Protocols simply represent standard bit formats that packets must conform to. Packets also must conform to some maximum size or maximum transmission unit (MTU). Transmitting blocks of data that are larger than the MTU is also problematic. Large blocks must be broken into packets that will fit within the MTU. Consequently, packet disassembly and reassembly are necessary functions that must be performed by networking hardware and software, too. Packets representing pieces of larger blocks must contain instructions for their reassembly at the receiver. In routing, two packets from the same logical transmission might get sent along different routes and even arrive at their destination out of sequence. Receiving packets out of order naturally complicates the task of reassembling the transmission at the receiver.

Protocol Stack

It is customary to speak of networking hardware and software technology as a series of distinct, well-defined layers. The notion of building networking technology by using layers of hardware and software began with a standardization process that originated in the early 1980s. When the ARPANET, the predecessor of today's Internet, was created, it implemented four standard networking layers. These Internet protocol layers are almost uniformly accepted as standards today, and they form the basis of the networking support for Microsoft Windows Server 2003. This layered architecture of the Internet is depicted in Figure 1-27. These are the standard layers of the TCP/IP protocol stack.

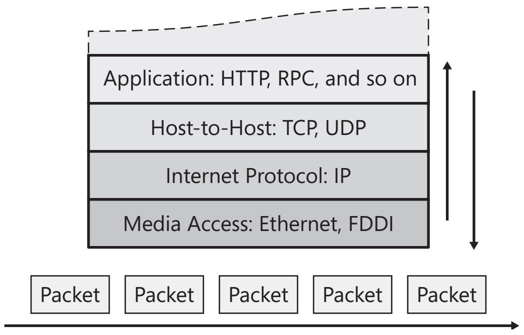


Figure 1-27 Networking protocol stack

The Internet architecture defines the following functional layers:

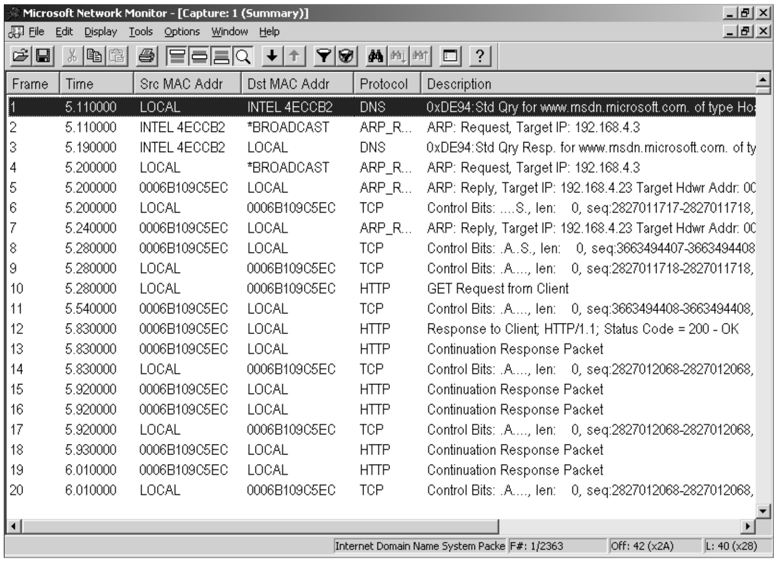
- **Media Access** The lowest level layer is concerned with the physical transmission media and how the signal it carries is used to represent data bits. The MAC layer is also sometimes decomposed further into Physical and Data Link layers. The various forms of Ethernet are the most common implementation of the MAC layer.
- **Internet Protocol (IP)** The IP layer is concerned with packet delivery. IP solves the problem of delivering packets across autonomous network segments. At each network hop, the IP decides the next system to forward the packet to. The packet gets forwarded from neighbor to neighbor in this manner until the payload reaches its intended final destination. The IP layer also includes the Address Resolution Protocol (ARP), the Internet Control Message Protocol

(ICMP), and the Border Gateway Protocol (BGP), which are involved in discovering and maintaining packet delivery routes. The most common version of the protocol deployed today is IP version 4; however the next version, IP version 6, is beginning to be deployed. Most concepts described in this chapter apply to both versions, although most of the discussion is in the context of IP version 4.

- **Host-to-Host** The Host-to-Host layer is concerned with how machines that want to transmit data back and forth can communicate. The Internet protocols define two Host-to-Host implementations—the User Datagram Protocol (UDP) for transmission of simple messages and the Transmission Control Program (TCP) for handling more complex transactions that require communication *sessions*. TCP is the layer that is responsible for ensuring reliable in-order delivery of all packets. It is also responsible for the flow control functions that have major performance implications in WAN communications.
- **Application** The Internet protocols include standard applications for transmitting mail (Simple Mail Transfer Protocol, or SMTP), files (File Transfer Protocol or FTP), Web browser hypertext files (Hypertext Transfer Protocol or HTTP), remote login (Telnet), and others. In addition, Windows Server 2003 supports many additional networking applications that plug into TCP, including Common Internet File System (CIFS), Remote Procedure Call (RPC), Distributed COM (DCOM), Lightweight Directory Access Protocol (LDAP), among others.

Processing of packets The layered architecture of the Internet protocols is much more than a conceptual abstraction. Each layer of networking services operates in succession on individual packets. Consider a datagram originally created by an application layer like the Microsoft Internet Information Services (IIS), which supports HTTP (the Internet's Hypertext Transfer Protocol) for communicating with a Web browser program. As created by IIS, this packet of information contains HTML format text, for example, in Response to a specific HTTP GET Request.

IIS then passes the HTTP Response Message to the next appropriate lower layer, which is TCP, in this case, for transmission to the requesting client program, which is a Web browser program running on a remote computer. The TCP layer of software is responsible for certain control functions such as establishing and maintaining a data communications *session* between the IIS Web server machine and an Internet Explorer Web browser client. TCP ensures that the data sent is reliably received by the remote end. TCP is also responsible for network flow control, which ensures that the sending computer system (IIS Web Server computer) does not flood the Internet routers or your client-side network with data. In addition, TCP is responsible for breaking the data to be sent into maximum sized segments that can be sent across the network to the destination. The TCP layer, in turn, passes the TCP segment to the IP layer, which decides which router or gateway to forward the packet to, such that the IP packet moves closer to its eventual destination. Finally, IP passes each IP packet to the MAC layer, which is actually responsible for placing bits on the wire. The layers in the networking protocol stack each operate in sequence on the packet. As illustrated in Figure 1-28, each layer also contributes some of the packet header control information that is ultimately placed on the wire.


Microsoft Network Monitor - [Capture: 1 (Summary)]
File Edit Display Tools Options Window Help

Frame	Time	Src MAC Addr	Dst MAC Addr	Protocol	Description
1	5.110000	LOCAL	INTEL 4ECCB2	DNS	0xDE94:Std Qry for www.msdn.microsoft.com. of type Ho
2	5.110000	INTEL 4ECCB2	*BROADCAST	ARP_R...	ARP: Request, Target IP: 192.168.4.3
3	5.190000	INTEL 4ECCB2	LOCAL	DNS	0xDE94:Std Qry Resp. for www.msdn.microsoft.com. of ty
4	5.200000	LOCAL	*BROADCAST	ARP_R...	ARP: Request, Target IP: 192.168.4.3
5	5.200000	0006B109C5EC	LOCAL	ARP_R...	ARP: Reply, Target IP: 192.168.4.23 Target Hdwr Addr: 0C
6	5.200000	LOCAL	0006B109C5EC	TCP	Control Bits: ....S., len: 0, seq:2827011717-2827011718,
7	5.240000	0006B109C5EC	LOCAL	ARP_R...	ARP: Reply, Target IP: 192.168.4.23 Target Hdwr Addr: 0C
8	5.280000	0006B109C5EC	LOCAL	TCP	Control Bits: .A..S., len: 0, seq:3663494407-3663494408
9	5.280000	LOCAL	0006B109C5EC	TCP	Control Bits: .A...., len: 0, seq:2827011718-2827011718,
10	5.280000	LOCAL	0006B109C5EC	HTTP	GET Request from Client
11	5.540000	0006B109C5EC	LOCAL	TCP	Control Bits: .A...., len: 0, seq:3663494408-3663494408,
12	5.830000	0006B109C5EC	LOCAL	HTTP	Response to Client; HTTP/1.1; Status Code = 200 - OK
13	5.830000	0006B109C5EC	LOCAL	HTTP	Continuation Response Packet
14	5.830000	LOCAL	0006B109C5EC	TCP	Control Bits: .A...., len: 0, seq:2827012068-2827012068,
15	5.920000	0006B109C5EC	LOCAL	HTTP	Continuation Response Packet
16	5.920000	0006B109C5EC	LOCAL	HTTP	Continuation Response Packet
17	5.920000	LOCAL	0006B109C5EC	TCP	Control Bits: .A...., len: 0, seq:2827012068-2827012068,
18	5.930000	0006B109C5EC	LOCAL	HTTP	Continuation Response Packet
19	6.010000	0006B109C5EC	LOCAL	HTTP	Continuation Response Packet
20	6.010000	LOCAL	0006B109C5EC	TCP	Control Bits: .A...., len: 0, seq:2827012068-2827012068,

Internet Domain Name System Packe F#: 1/2363 Off: 42 (x2A) L: 40 (x28)

Figure 1-28 Network Monitor

The protocol stack functions in reverse order at the receiving station. Each layer processes the packet based on the control information encapsulated in the packet header deposited by the corresponding layer at the sending computer system. If the layer determines that the received packet contains a valid payload, the packet header data originally inserted by that corresponding layer at the sender station is stripped off. The remaining payload data is then passed up to the next higher layer in the stack for processing. In this fashion, the packet is processed at the receiving station by the MAC layer, the IP layer, and the TCP layer in sequence, until it is finally passed to the Web browser application that originally requested the transmission and knows how to format HTML text so that it looks good on your display monitor.

Example packet trace: processing an HTTP GET request Figure 1-28 is a Network Monitor packet capture that illustrates how these processing layers work. In this example, a TCP-mandated SYN request (SYN is short for *synchronize*) in Frame 6 is transmitted from an Internet Explorer Web browser to establish a session with the Web server running at http://www.msn.com. The initial session request packet has a number of performance-oriented parameters including Selective Acknowledgement (SACK) and the TCP Advertised Window. Notice in the Network Monitor's middle frame how the TCP header information is encapsulated inside an IP segment, which is then enclosed in an Ethernet packet for transmission over the wire.

Frame 8 in Figure 1-28 shows a SYN, ACK response frame from the Web server that continues the session negotiation process. Notice that it was received about 80 milliseconds following the initial transmission. This round trip time is a key performance metric in TCP because it governs the Retransmission Time Out (RTO) that the protocol uses to determine when congested routers have dropped packets and data needs to be retransmitted. This aspect of TCP *congestion control* will be discussed in more detail later in this chapter. In this example, the ACK packet in Frame 9 completes the sequence of packets that establishes a session between two TCP host machines.

Frame 10 follows immediately, an HTTP protocol GET Request to the MSN Web site to access the site's home page. This GET Request also passes a cookie containing information to the IIS Web server about the initiator of the request. The TCP running on the Web server acknowledges this packet in Frame 11, some 260 milliseconds later.

An IIS Web server then builds the HTTP Response message, which spans Frames 12, 13, 15, 16, 18, and 19. Here the HTTP Response message is larger than an Ethernet MTU (maximum transmission unit), so it is fragmented into multiple segments. The TCP layer is responsible for breaking up this Response message into MTU-size packets on the Sender side and then reassembling the message at the Receiver. After these frames have been received by the Web browser, Internet Explorer has all the data it needs to render an attractive-looking Web page.

The Network Monitor captures and displays packet headers in a way that lets you easily dig down into the protocol layers to see what is going on. To use the Network Monitor effectively to diagnose performance problems, it helps to understand a little more about these networking protocols and what they do.

Bandwidth

Bandwidth refers to the data rate of the data communications transmission, usually measured in bits per second. It is the capacity of the link to send and receive data. Some authorities suggest visualizing bandwidth as the width of the data pipe connecting two stations. A better analogy is to visualize the rate at which bits arrive at the other end of the pipe. Bandwidth describes the rate at which bits are sent across the link. It tells you nothing about *how long* it takes to transmit those bits. Physically, each bit transmitted across the wire is part of a continuous wave form. The waveform cycles at 100 MHz for 100baseT and at 1 GHz for Gigabit Ethernet. In the time it takes to send 1 bit using 100baseT, you can send 10 bits using the Gigabit Ethernet standard.

Bandwidth is usually the prime performance concern in LANs, only when the network segment is used to move large blocks of data from point to point, as in disk-to-tape backup or video streaming applications. For long distance data communications, especially for organizations attempting to do business on the Web, for example, the long latency, not bandwidth, is the more pressing (and less tractable) performance problem.

Table 1-5 shows the bandwidth rating of a variety of popular link technologies. It also compares the relative bandwidth of the link to a 56 Kbps telephone line. It is more precise, however, to speak of the *effective bandwidth* of a data communications link. Effective bandwidth attempts to factor in the many types of overhead that add bytes to the data payload you are attempting to move over the network. Consider what happens when you transfer a 10-MB (megabyte) file using either FTP or Microsoft's CIFS network file sharing protocol from one machine to another across a 100-Mbps (Megabits per second) switched Ethernet link.

Table 1-5 Connection Speed for a Variety of Networking Links

Circuit	Connection Speed (bps)	Relative Speed
Modem	28,800	0.5
Frame Relay	56,000	1
ISDN	128,000	2
DSL	640,000	12
T1/DS1	1,536,000	28
10 Mb Ethernet	10,000,000	180
11 Mb Wireless	11,000,000	196
T3/DS3	44,736,000	800
OC1	51,844,000	925
100 Mb Fast Ethernet	100,000,000	1800
FDDI	100,000,000	1800
OC3	155,532,000	2800
ATM	155,532,000	2800
OC12	622,128,000	11,120
Gigabit Ethernet	1,000,000,000	18,000

The first overhead of data communication that should be factored into any calculation of effective bandwidth is the packet-header overhead. The 10-MB file that the FTP or SMB protocol transfers must be broken into data packets no larger than Ethernet's 1500-byte MTU. As illustrated in the HTTP packet trace in Figure 1-28, each Ethernet packet also contains IP, TCP, and application headers for this application. The space in the packet that these protocol stack headers occupy reduces effective bandwidth by about 2–3 percent. Plus, there are other protocol-related overheads such as the ACK packets seen in the Figure 1-28 trace that further reduce effective bandwidth. Altogether, the overhead of typical TCP/IP traffic reduces the effective bandwidth of a switched Ethernet link to approximately 95 percent of its rated capacity. If you experiment with transferring this hypothetical 10-MB file across a typical 100-Mbps Ethernet link, you will probably be able to measure only about 95 Mbps of throughput, which for planning purposes, is the effective bandwidth of the link.

The most important measure of bandwidth usage is line utilization. The measurement technique is straightforward. Using MAC layer length fields, a measurement layer inserted into the network protocol stack accumulates the total number of bytes received from packets transferred across the link. Utilization is then calculated as:

network interface utilization = Bytes Total/sec current bandwidth

where both fields are measured in bytes per second. Dividing the Network Interface\Bytes Total/sec counter by the Network Interface\Current Bandwidth counter yields the utilization of the link.

Latency

Latency refers to the delay in sending bits from one location to another. It is the length of time it takes to send a message from one station to another across the link. Electronic signals travel at the speed of light, approximately 300,000 kilometers per second. The physical characteristics of transmission media do have a dampening effect on signal propagation delays, with a corresponding increase in latency. The effective speed of an electronic data transmission wire is only about 1/2 the speed of light, or 150,000 km/second. Optical fiber connections reach fully 2/3 the speed of light, or a latency of 200,000 km/second. The delay involved in sending a message from a location in the eastern United States to a west coast location across a single, continuous optical cable would traverse 5,000 km. At a top speed of 200,000 km/second, the latency for this data transmission is a not insignificant 25 milliseconds. For a rule-of-thumb calculation, allow for at least 5 milliseconds of delay for every 1000 kilometers separating two stations.

Of course, most long distance transmissions do not cross simple, continuous point-to-point links. Over long distances, both electrical and optical signals attenuate and require amplification using *repeaters* to reconstitute the signal and send it further along on its way. These repeaters add latency to the transmission time. Because a long-distance transmission will traverse multiple network segments, additional processing is necessary at every *hop* to route packets to the next hop in the journey between sender and receiver. Processing time at links, including routers and repeaters and amplifiers of various forms, adds significant delays at every network hop. High performance IP packet routers that are designed to move massive amounts of traffic along the Internet backbone, for example, might add 10 µsecs of delay. Slower routers like the ones installed on customer premises could add as much as 50 µsecs of additional latency to the transmission time. This yields a better estimate of long distance data communication latency:

(distance / signal propagation delay) + *(hop count* × *average router latency)*

Because determining network latency across a complex internetworking scheme is so important, the Internet protocols include facilities to measure network packet routing response time. The Internet Control Message Protocol (ICMP), a required component of the TCP/IP standard, supports an Echo Reply command that returns the response time for the request. A simple command-line utility called *ping* that is included with Windows Server 2003 issues several ICMP Echo Reply commands and displays the response time as reported by the destination node. A slightly more sophisticated utility called *tracert* decomposes the response time to a remote IP destination by calculating the time spent traversing every hop in the route. These utilities are discussed in more detail later in this chapter.

A related measure of latency is the *round trip time* (RTT). RTT is defined as the time it takes for a message to get to its destination and back (usually the sum of the latency in the forward direction and the latency in the backward direction). In typical client/server transactions, network RTT corresponds closely to the response time that the user of the application perceives. As discussed earlier, this perceived application response time is the most important performance metric because of its correlation with user satisfaction. RTT is also used by TCP to track whether a particular data packet has been lost; the performance of TCP on *lossy* links depends on RTT.

Ethernet

In Ethernet, the network connections are *peer-to-peer*, meaning there is no master controller. Because there is no master controller in Ethernet peer-to-peer connections and the link is a shared transmission medium, it is possible for *collisions* to occur when two stations attempt to use the shared link at the same time. The performance impact of Ethernet collisions is discussed later.

Fault tolerance, price, and performance are the main considerations that determine the choice of a local area network configuration. As the price of Ethernet switches has dropped, switched network segments have become more common. Unfortunately, many people are then disappointed when a bandwidth-hogging application like tape backup does not run any faster when a switch is configured instead of a hub. Because the underlying protocol is unchanged, point-to-point data transmissions that proceed in a serial fashion cannot run any faster.

Another caution is not to get confused by the terminology that is used to identify hubs and switches. Physically, hubs are wiring hubs that function logically as rings where collisions occur whenever two stations attempt to send data concurrently. Switches create network segments that function logically like spoke and hub configurations where multiple transmissions can be in progress simultaneously on what are, in effect, dedicated links operating at the interface's full rated bandwidth. Collisions occur only in switched networks when two (or more) stations A and B attempt to send data to the same station C concurrently.

The Ethernet protocol is peer-to-peer, requiring no master controller of any kind. Among other things, this makes an Ethernet network very easy to configure—you can just continue to extend the wire and add links, up to the physical limitations of the protocol in terms of the number of stations and the length of the wiring loop. Unlike the SCSI protocol used to talk to a computer's disk, tape, and other peripherals, for example, the Ethernet standard has no provision for time-consuming and complex bus arbitration. An Ethernet station that has data that it wants to send to another session does not face any sort of arbitration. A station simply waits until the transmission medium appears to be free and then starts transmitting data.

This simple approach to peer-to-peer communication works best on relatively lightly used network segments where stations looking to transmit data seldom encounter a busy link. The rationale behind keeping the Ethernet protocol simple suggests that it is not worth bothering about something that rarely happens anyway. The unhappy result of having no bus arbitration is that in busier network segments, multiple stations can and do try to access the same communications link at the same time. This leads to *collisions*, which are disrupted data transmissions that then must be retried.

A station with data to transmit waits until the wire appears free before attempting to transmit. Each transmission begins with a characteristic preamble of alternating 0 and 1 bits of proscribed length. (The network interface card discards this preamble so that it is not visible in a Network Monitor packet trace.) The preamble is followed by a 1-byte start delimiter that contains the bit sequence 10101011 designed to distinguish the preamble from the beginning of the real data to be transmitted.

The station then continues with the transmission, always sending an entire packet, or *frame*, of information. Each Ethernet frame begins with the 48-bit destination address, followed by the 48-bit source address. These 48-bit MAC addresses uniquely identify the Ethernet source and destination addresses—this is accomplished by giving every hardware manufacturer a distinct range of addresses that only it can use. These unique MAC addresses are also called *unicast* addresses. Ethernet also supports *broadcast* addresses where the address field is set to binary 1s to indicate that it should be processed by all LAN cards on the segment. Broadcast messages are used, for example, to pass configuration and control information around the network.

Maximum transmission unit The length of the frame, including the header, is encoded in the frame header immediately following the addressing fields. For historical reasons, Ethernet frames are limited to no more than 1514 bytes (1518 bytes, if you include the required postamble bits) to keep any one station from monopolizing a shared data link for too long. Assuming that successive Ethernet, IP, and TCP headers occupy a minimum of 54 bytes, the data payload in an Ethernet packet is limited to about 1460 bytes. As the speed of Ethernet links has increased, the small frame size that the protocol supports has emerged as a serious performance limitation. For example, CIFS access to remote files must conform to the Ethernet MTU, causing blocks from large files to be fragmented into multiple packets. This slows down network throughput considerably because each station must wait a predetermined interval before transmitting its next packet. Consequently, some Gigabit Ethernet implementations across 1 Gb/sec high-speed fiber optics links optionally create so-called *jumbo frames*. Windows Server 2003 support for the Gigabit Ethernet standard is discussed in Chapter 6, “Advanced Performance Topics.”

Following the actual data payload, each Ethernet frame is delimited at the end by a Frame Check Sequence, a 32-bit number that is calculated from the entire frame contents (excluding the preamble) as a cyclic redundancy check (CRC). A receiving station calculates its own version of the CRC as it takes data off the wire and compares it to the CRC embedded in the frame. If they do not match, it is an error condition and the frame is rejected.

Collision detection When two (or more) stations have data to transmit and they both attempt to put data on the wire at the same time, this creates an error condition called a *collision*. What happens is that each station independently senses that the wire is free and begins transmitting its preamble, destination address, source address, and other header fields, data payload, and CRC. If more than one station attempts to transmit data on the wire, the sequence of bits from two different frames becomes hopelessly intermixed. The sequence of bits received at the destination is disrupted, and, consequently, the frame is rejected.

A sending station detects that a collision has occurred because it also receives a copy of the disrupted frame, which no longer matches the original. The frame must be long enough so that the original station can detect the fact that the collision has occurred before it attempts to transmit its next packet. This key requirement in the protocol specification determines a minimum sized packet that must be issued. Transmissions smaller than the minimum size are automatically padded with zeros to reach the required length.

The latency (or transmission delay) for a maximum extent Ethernet segment determines the minimum packet size that can be sent across an Ethernet network. The propagation delay for a maximum extent network segment in 10BaseT, for example, is 28.8 µsecs, according to the standards specification. The sending station must send data to the distant node and back to detect that a collision has occurred. The round trip time for the maximum extent network is 2 × 28.8 µsecs, or 57.6 µsecs. The sending station must get a complete frame header, data payload, and CRC back to detect the collision. At a data rate of 10 Mb/sec, a station could expect to send 576 bits, or 72 bytes in 57.6 µsecs. Requiring each station to send at least 72 bytes means that collisions can be detected across a maximum extent network. Ethernet pads messages smaller than 72 bytes with zeros to achieve the minimum length frames required.

With wiring hubs, most Ethernet segments that are in use today do not approach anywhere near the maximum extent limits. With hubs, maximum distances in the range of 200–500 meters are typical. At the 512 meters limit of a single segment, Ethernet latency is closer to 5 µsecs, and with shorter segments the latency is proportionally less. So you can see that with wiring hubs, Ethernet transmission latency is seldom a grave performance concern.

Switches help to minimize collisions on a busy network because stations receive only packets encoded with their source or destination address. Collisions can and still do occur on switched segments, however, and might even be prevalent when network traffic all tends to be directed at one or more Windows Server 2003 machines configured on the segment. When there are two network clients that both want to send data to the same server at the same time, a collision on the link to the server can and will occur on most types of switches.

Back-off and retry Collisions disrupt the flow of traffic across an Ethernet network segment. The data Sender A intended to transmit is not received properly at the receiving station C. This causes an error condition that must be corrected. Sender station B also trying to send data to C also detects that a collision has occurred. Both Senders must resend the frames that were not transmitted correctly.

The thought might occur to you that if Sender A and Sender B *both* detect a collision and *both* try to resend data again at the same time, it is highly likely that the datagrams will collide again. In fact this is exactly what happens on an Ethernet segment. Following a collision, Ethernet executes its *exponential back-off and retry* algorithm to try to avoid potential future collisions. Each station waits a random period of time before resending data to recover from the collision. If a collision reoccurs the second time (the probability of another collision on the first retry remains high), each station doubles the potential delay interval and tries again. If a collision happens *again*, each station doubles the potential delay interval again, and so on, until the transmission finally succeeds. As the potential interval between retries lengthens, one of the stations will gain enough of a staggered start that eventually its transmission will succeed.

In summary, the Ethernet protocol avoids the overhead of a shared bus arbitration scheme to resolve conflicts when more than one station needs access to the bus. The rationale is, "Let's keep things simple." This approach has much to commend it. As long as the network is not heavily utilized, there is little reason to worry about bus contention.

When conflicts do arise, which they inevitably do on busier networks, Ethernet stations detect that the collisions have occurred and attempt to recover by retrying the transmissions until they succeed. Notice that each station executes the exponential back-off algorithm independently until the transmissions finally succeed. No master controller is ever required to intervene to bring order to the environment. Moreover, no priority scheme is involved in sharing the common transmission medium.

Performance monitoring So long as network utilization remains relatively low, the performance characteristics of Ethernet are excellent. For a nonswitched LAN, as network utilization begins to increase above 20–30 percent busy with multiple stations attempting to transmit data on the segment, collisions begin to occur. Because of retries, utilization of the segment increases sharply, doubling from 30–35 percent busy to 60–70 percent, as depicted in Figure 1-29.

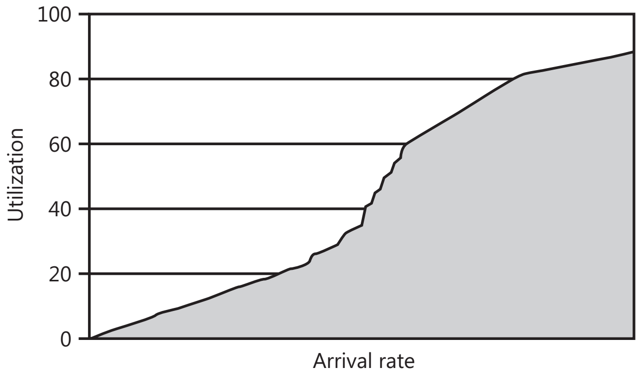


Figure 1-29 The characteristic utilization bulge in the utilization of a shared Ethernet transmission medium because of collisions

Figure 1-29 illustrates the bulge in utilization that can be expected on an Ethernet segment once collisions begin to occur. This characteristic bulge leads many authorities to recommend that you try to keep the utilization of Ethernet segments below 30–40 percent busy for nonswitched segments. Switched segments can typically sustain much higher throughput levels without collisions, but remember that most switches do not eliminate all collisions. Because collision detection and retries are a Network Interface hardware function, performance monitoring from inside a machine running Windows Server 2003 cannot detect that collisions and retries are occurring. Only packets sent and delivered successfully are visible to the network interface measurement software.

Lacking direct measurement data, you must resort to assembling a case for collisions occurring indirectly. From the standpoint of performance monitoring, utilization levels on the link of less than the 100 percent theoretical maximum utilization may represent the effective saturation point, given the way Ethernet behaves. Remember, however, that the condition that causes collision is contention for the transmission link. If the only activity on a segment consists of station A sending data to station B (and station B acknowledging receipt of that data with transmissions back to A) during an application like network backup, there is no contention for the link. Under those circumstances, you can drive utilization of an Ethernet link to 100 percent without collisions.

Warning Switched networks provide significant relief from performance problems related to Ethernet collisions, but they do not eliminate them completely. A switch provides a dedicated virtual circuit to and from every station on the segment. With a switch, station A can send data to B while station C sends data to D concurrently without a collision. However, collisions can still occur on a switched network if two stations both try to send data to a third station concurrently, which is frequently what happens in typical client/server networking configurations.

You might have access to other network performance statistics from your switches that are available through RMON or SNMP interfaces that report the rate of collisions directly.

IP Routing

The Internet Protocol layer, also known as layer 3 (with the physical and data link layers associated with the MAC layer being layers 1 and 2, respectively) is primarily concerned with delivering packets from one location to another. On the sender, the IP layer decides which gateway or router the outgoing packet must be sent to. On the receiver, it is IP that makes sure that the incoming packet has been sent by the expected router or gateway. On intermediate links, IP picks the incoming packet and then decides which network-segment the packet should go out on, and which router it should be sent to, so that the packet moves closer to its eventual destination. This technology is called *routing*. Routing is associated with a bundle of standards that include IP itself, ICMP, ARP, BGP, and others. This section introduces the key aspects of IP routing technology that most impact network performance and capacity planning. Note that this section discusses IP-layer concepts in the context of IP version 4, the dominant version of IP on the Internet today. However, most of the discussion applies to IP version 6 also.

The basic technology used in IP routing is deceptively simple. What makes IP routing such a difficult topic from a performance perspective is the complicated, interconnected network infrastructures and superstructures that organizations have erected to manage masses of unscheduled IP traffic. That the Internet works as well as it does is phenomenal, given the complexity of the underlying network of networks that it supports.

Routing Routing is the process where packets are forwarded from one network segment to the next until they reach their final destination. These network segments can span organizations, regions, and countries, with the result that IP is used to interconnect a vast worldwide network of computers. IP is the set of routing standards that ties computers on both private intranets and the public Internet together so that they can communicate with each other to send mail, messages, files, and other types of digital information back and forth.

Devices called *routers* serve as gateways, interconnecting different network segments. They implement layer 3 packet *forwarding*. Routers are connected to one or more local LAN segments and then connected via WAN links to other routers located on external networks. Windows Server 2003 machines can be configured to serve as routers by enabling the IP Forwarding function. However, the more common practice is to use dedicated devices that are designed specifically to perform layer 3 switching. Routers are basically responsible for forwarding packets on to the next hop in their journey toward their ultimate destination. Routers serve as gateways connecting separate and distinct network segments. They recognize packets that have arrived at the network junction that are intended for internal locations. They place these packets on the LAN, where they circulate until they reach the desired MAC address. Routers also initiate messages (encapsulated as packets, naturally) intended for other routers that are used to exchange information about routes.

Although the IP layer is responsible for moving packets toward their eventual destination, IP does not guarantee delivery of those packets. Moreover, once a packet is entrusted to IP for delivery, there is no mechanism within IP to confirm the delivery of that packet as instructed. IP was designed around a "best effort" service model that is both *unreliable* and *connectionless*. It is the Host-to-Host connection layer above IP that is responsible for maintaining reliable, in-order delivery of packets. That component, of course, is TCP.

Being a “best effort” service model, IP works hard to deliver the packets entrusted to it. Using IP, if there is a serviceable route between two addresses on the Internet, no matter how convoluted, IP will find it and use it to deliver the datagram payload. As you can probably imagine, route availability across a large system of interconnected public and private networks is subject to constant change. For example, it is a good practice for critical locations on your private network to be accessible by two or more connections or paths. If one of these links goes down, IP will still be able to deliver packets through an alternate route.

Determining which path among many possible choices that exist is one of the responsibilities of Internet Protocol layer 3 routers. Undoubtedly, some routes are better than others because they can deliver traffic faster to a destination, more reliably, or with less cost. Some routers implement the simple Routing Information Protocol (RIP), which selects routes primarily based on the number of hops involved. More powerful routers usually implement the more robust Open Shortest Path First (OSPF) protocol, which attempts to assess both route availability and performance in making decisions. The popularity of the public access Internet has recently generated interest in having routers use policy-oriented Quality of Service (QoS) metrics to select among packets that arrive from different sources and different applications. An in-depth discussion comparing and contrasting these routing methods is beyond the scope of this chapter, but it is a good discussion to have with your networking provider.

Routing tables The dynamic aspects of routing create a big problem: specifically, how to store all that information about route availability and performance and keep it up-to-date. The Internet consists of thousands and thousands of separate autonomous network segments. They are interconnected in myriad ways. The route from your workstation to some location like http://www.microsoft.com is not predetermined. There is no way to know beforehand that such a route even exists.

IP solves the problem of how to store information about route availability in an interesting way. The IP internetworking environment does not store a complete set of routing information in any one, centralized location that might be either vulnerable to failure or be subject to becoming a performance bottleneck.

Instead, information about route availability is distributed across the network, maintained in *routing tables* stored in individual routers. These routing tables list the specific network addresses to which the individual router can deliver packets. In addition, there is a default location—usually another router—where the router forwards any packets with an indeterminate destination for further address resolution. The *route* command-line utility can be used to display the contents of a machine’s routing table:

```
C:\>route print
===========================================================================
Interface List
0x1 ........................... MS TCP Loopback interface
0x2000002 ...00 00 86 38 39 5a ...... 3Com Megahertz LAN + 56K
===========================================================================
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      24.10.211.1    24.10.211.47       1
      24.10.211.0    255.255.255.0     24.10.211.47    24.10.211.47       1
     24.10.211.47  255.255.255.255        127.0.0.1       127.0.0.1       1
   24.255.255.255  255.255.255.255     24.10.211.47    24.10.211.47       1
        127.0.0.0        255.0.0.0        127.0.0.1       127.0.0.1       1
    192.168.247.0    255.255.255.0    192.168.247.1   192.168.247.1       1
    192.168.247.1  255.255.255.255        127.0.0.1       127.0.0.1       1
    200.200.200.0    255.255.255.0    200.200.200.1   200.200.200.1       1
    200.200.200.1  255.255.255.255        127.0.0.1       127.0.0.1       1
        224.0.0.0        224.0.0.0    200.200.200.1   200.200.200.1       1
        224.0.0.0        224.0.0.0     24.10.211.47    24.10.211.47       1
        224.0.0.0        224.0.0.0    192.168.247.1   192.168.247.1       1
  255.255.255.255  255.255.255.255    192.168.247.1         0.0.0.0       1


===========================================================================
Persistent Routes:
  None
```

This sample route table is for a Windows Server 2003 machine at address 24.10.211.47 serving as a router. This table marks addresses within the 24.10.211.0 Class C network range (with a subnet mask of 255.255.255.0) for local delivery. It also shows two external router connections at locations 200.200.200.1 and 192.168.247.1. Packets intended for IP addresses that this machine has no direct knowledge of are routed to the 24.10.211.1 gateway by default.

The set of all IP addresses that an organization's routers manage directly defines the boundaries of what is known in routing as an *autonomous system* (AS). For the Internet to work, routers in one autonomous system need to interchange routing information with the routers they are connected to in other autonomous systems. This is accomplished using the Border Gateway Protocol (BGP). Using BGP, routers exchange information with other routers about the IP addresses that they are capable of delivering packets to.

As the status of links within some autonomous network configuration changes, the routers at the borders of the network communicate these changes to the routers they are attached to at other external networks. Routers use Border Gateway Protocol (BGP) messages to exchange information about routes with routers they are connected to across autonomous systems. BGP is mainly something that the Internet Service Providers worry about. Naturally, ISP routers maintain extensive routing tables about the subnetworks the Providers manage.

One of the key ingredients that makes IP routing across a vast network like the Internet work is that each IP packet that routers operate on is self-contained. Each packet contains all the information that the layer 3 switching devices need to decide where to deliver the packet and what kind of service it requires. The IP header contains the addresses of both the sender and the intended receiver. Another feature of IP is that routers operate on each self-contained IP packet individually. It is quite possible for one packet intended for B that is sent by A to get delivered to its destination following one route while the next packet is delivered by an entirely different route.

When a router receives a packet across an external link that is destined for delivery locally, the router is responsible for delivering that packet to the correct station on the LAN. That means the router sends this packet to the MAC layer interface, plugging in the correct destination address. (The router leaves the Source address unchanged so that you can always tell where the packet originated.) The Address Resolution Protocol (ARP) is used to maintain a current list of local IP addresses and their associated MAC addresses.

Router performance The fact that IP does not guarantee the delivery of packets does have some interesting performance consequences. When IP networks get congested, routers can either queue packets for later processing or drop the excess load. By design, most high-performance routers do the latter. They are capable of keeping only a very small queue of packets. If additional requests to forward packets are received and the queue is full, most routers simply drop incoming packets. Dropping packets is acceptable behavior in IP, and even something that is to be expected. After all, IP never guaranteed that it would deliver those packets in the first place. The protocol is designed to make only its “best effort” to deliver them.

The justification for this strategy is directly related to performance. The original designers of the Internet understood that persistent bottlenecks in the Internet infrastructure would exist whenever some customer’s equipment plugged into the network was hopelessly undersized. In fact, given the range of organizations that could connect to the Internet, it is inevitable that some routes are not adequately sized. Therefore, the Internet packet delivery mechanism needs to be resilient in the face of persistent, mismatched speeds of some of the components of a route.

In the face of inevitable speed mismatches, what degree of queuing should a router serving as the gateway between two networks attached to the Internet support? Consider that queuing at an undersize component during peak loads could lead to unbounded queuing delays whenever requests started to arrive faster than the router could service them. It is also inevitable that whatever queue depth a bottlenecked router was designed to support could readily be exceeded at some point. When a router finally exhausts the buffer space it has available to queue incoming packets, it would be necessary to discard packets anyway. With this basic understanding of the fundamental problem in mind, it makes sense to start to discard packets before the queue of deferred packets grows large and begins, for example, to require extensive resources to manage.

Routers have a specific processing capacity, rated in terms of packets/second. When packets arrive at a router faster than the router can deliver them, excess packets are dropped. Because most routers are designed to support only minimal levels of queuing, the response times for the packets they deliver is very consistent, never subject to degradation when the network is busy. The price that is paid for this consistent response time is that some packets might not be delivered at all.

Obviously, you need to know when the rate of network traffic exceeds capacity because that is when routers begin dropping packets. You can then upgrade these routers or replace them with faster units. Alternatively, you might need to add network capacity in the form of both additional routes and routers. Performance statistics are available from most routers using either SNMP or RMON interfaces. In addition, some routers return ICMP Source Quench messages when they reach saturation and need to begin dropping packets.

Tip The IP statistics that are available in System Monitor count packets received and processed and are mainly useful for network capacity planning. These include IP\Datagrams Received/sec, IP\Datagrams Received/sec, and the total IP\Datagrams/sec. Counters for both the IP version 4 (IPv4) and IP version 6 (IPv6) versions of the protocol are available.

Obviously, dropping packets at a busy router has dire consequences for someone somewhere, namely the originator of the request that failed. Indeed it does. Although IP is not concerned about what happens to a few packets here and there that might have gotten dropped, this is a concern at the next higher level in the protocol stack, namely in the TCP Host-to-Host connection layer 4. TCP will eventually notice that a packet is missing and attempt some form of error recovery, which involves resending the original packet. If TCP cannot recover from the error, it will eventually notify the application that issued the request. This leads to the familiar "Request Timed Out" error message in your Web browser that prods you into retrying the entire request.

The structural problem of having an overloaded component on the internetwork somewhere is something that also must be systematically addressed. In networking design, this is known as the problem of *flow control*, for example, what to do about a sender computing system that is overloading some underpowered router installed on a customer's premises. Again, flow control is not a concern at the IP level, but at the next higher level, TCP does provide a suitable flow control and a congestion control mechanism, which is discussed later.

Packet headers IP packet headers contain the familiar source and destination IP addresses, fragmentation instructions, and a hop count called TTL, or Time To Live. Three IP header fields provide the instructions used in packet fragmentation and reassembly. These are the Identification, Flags, and Offset fields that make up the second 32-bit word in the IP packet header. The IP layer at the packet's destination is responsible for reassembling the message from a series of fragmented packets and returning data to the application in its original format.

The Time To Live (TTL) field is used to ensure that packets cannot circulate from router to router around the network forever. Each router that operates on a packet decrements the TTL field before sending it on its way. If a router finds a packet with a TTL value of zero, that packet is discarded on the assumption that it is circulating in an infinite loop.

Note As originally specified, the 1-byte TTL field in the IP header was supposed to represent the number of seconds a packet could survive on the Internet before it was discarded. Each IP router in the destination path was supposed to subtract the number of seconds that the packet resided at the router. But because packets tended to spend most of their time in transit *between* routers connected by long distance WAN links, this scheme proved unworkable. The TTL field was then reinterpreted to mean the number of path components that a packet travels on the way to its destination. Today, when the TTL hop count gets to zero, the packet is discarded.

Subtracting the final TTL value observed at the destination from the initial value yields the *hop count*, which is the number of links traversed before the packet reached its final destination. TTL is a 1-byte field with a maximum possible link count of 255. Windows Server 2003 sets TTL to 128 by default. Because IP packets typically can span the globe in less than 20 hops, a default TTL value of 128 is generous.

ICMP The Internet Control Message Protocol (ICMP) is used to generate informational error messages on behalf of IP. Even though ICMP does not serve to make IP reliable, it certainly makes IP easier to manage. In addition to its role in generating error messages, ICMP messages are used as the basis for several interesting utilities, including ping and tracert.

Ping is a standard command-line utility that utilizes ICMP messaging. The most common use of the ping command is simply to verify that one IP address can be reached from another. The ping command sends an ICMP type 0 Echo Reply message and expects a type 8 Echo Request message in reply. The ping utility calculates the round trip time for the request and the TTL value for a one-way trip. (Note that ping sets TTL to a value of 255 initially.) By default, ping sends Echo Reply messages four times so that you can see representative RTT and hop count values. Because different packets can arrive at the destination IP address through different routes, it is not unusual to observe variability in the four measurements.

A more elaborate diagnostic tool is the *tracert* utility that determines the complete path to the destination, router by router. Typical output from the tracert command follows:

```
C:\>tracert 207.46.155.17

Tracing route to 207.46.155.17 over a maximum of 30 hops

  1     1 ms     1 ms     1 ms  192.168.0.101
  2    14 ms    15 ms    13 ms  12-208-96-1.client.attbi.com [12.208.96.1]
  3    13 ms    13 ms    11 ms  12.244.104.97
  4    13 ms    15 ms    16 ms  12.244.72.230
  5    17 ms    15 ms    14 ms  gbr6-p90.cgcil.ip.att.net [12.123.6.6]
  6    19 ms    15 ms    13 ms  tbr2-p013601.cgcil.ip.att.net [12.122.11.61]
  7    56 ms    53 ms    53 ms  gbr4-p20.st6wa.ip.att.net [12.122.10.62]
  8    58 ms    59 ms    56 ms  gar1-p370.stwwa.ip.att.net [12.123.203.177]
  9    56 ms    59 ms    58 ms  12.127.70.6
 10    59 ms    57 ms    58 ms  207.46.33.225
 11    59 ms    60 ms    57 ms  207.46.36.66
 12    54 ms    57 ms    56 ms  207.46.155.17

Trace complete.
```

The way the tracert command works is that it begins by sending ICMP Echo Reply type 0 messages with a TTL of 1, then increments TTL until the message is successfully received at the destination. In this fashion, it traces at least one likely route of a packet and calculates the cumulative amount of time it took to reach each intermediate link. (This is why tracert sometimes reports that it takes less time to travel further along the route—the response times displayed represent different ICMP packets that were issued.) The tracert command also issues a DNS reverse query to determine the DNS name of each node in the path.

TCP

The Transmission Control Protocol (TCP) is the Layer 4 protocol that provides a reliable, peer-to-peer delivery service. TCP sets up point-to-point, connection-oriented sessions to guarantee reliable in-order delivery of application transmission requests.

TCP sessions are full duplex, capable of sending and receiving data between two locations concurrently. This section reviews the way the TCP protocol works. The most important TCP tuning parameters are discussed in Chapter 5, "Performance Troubleshooting."

TCP sessions, or *connections*, are application-oriented. TCP port numbers uniquely identify applications that plug into TCP. Familiar Internet applications like HTTP, FTP, SMTP, and Telnet all plug into TCP *sockets*. Microsoft networking applications like DCOM, RPC, and the SMB server and redirector functions also utilize TCP. They use the NBT interface that allows NetBEUI services to run over TCP/IP.

TCP connection-oriented behavior plays an important role in network performance because the TCP layer is responsible for both flow control and congestion control. The flow control problem was cited earlier: how to keep a powerful sender from overwhelming an undersized link. Congestion control deals with a performance problem that arises in IP routing where busy routers drop packets instead of queuing them. The TCP layer that is responsible for in-order, reliable receipt of all data, detects that packets are being dropped. Because the likely cause of packets being dropped is router congestion, TCP senders recognize this and back off to lower transmission rates.

TCP advertises a *sliding window* that limits the amount of data that one host application can send to another without receiving an Acknowledgement. Once the Advertised Window is full, the sender must wait for an ACK before it can send any additional packets. This is the flow control mechanism that TCP uses to ensure that a powerful sender does not overwhelm the limited capacity of a receiver.

Being unable to measure network link capacity directly, TCP instead detects that network congestion is occurring and backs off sharply from sending data. TCP recognizes two congestion signals: a window-full condition indicating that the receiver is backed up; and an unacknowledged packet that is presumed lost because of an overloaded router. In both cases, the TCP sender reacts by reducing its network transmission rates.

These two important TCP performance-oriented functions are tied to the round trip time (RTT) of connections, which TCP calculates. Together, the RTT and the size of the TCP sliding data window determine the throughput capability of a TCP connection. RTT also figures into TCP congestion control. If a sender fails to receive a timely Acknowledgement message that a packet was delivered successfully, TCP ultimately retransmits the datagram. The amount of time TCP waits for an Acknowledgement to be delivered before it retransmits the data is based on the connection RTT. Important TCP tuning options include setting the size of the Advertised Window and the method used to calculate RTT.

Session connections Before any data can be transferred between two TCP peers, these peers first must establish a connection. In the setup phase of a connection, two TCP peers go through a handshaking process where they exchange information about each other. Because TCP cares about delivering data in the proper sequence, the hosts initiating a session need to establish common sequence numbers to use when they want to begin transferring data. The peers also negotiate to set various options associated with the session, including establishing the size of the data transfer window, the use of selective acknowledgement (SACK), the maximum segment size the parties can use to send data, and other high-speed options like timestamps and window scaling.

To initiate a connection, TCP peer 1 sends a Synchronize Sequence Number (SYN) message that contains a starting sequence number, a designated port number to send the reply to, and the various proposed option settings. This initial message is posted to a standard application port destination at the receiver's IP address—for example, Port 80 for an HTTP session between a Web browser and a Web server. Then TCP peer 2 acknowledges the original SYN message with a SYN-ACK, which returns the receiver's starting sequence number to identify the packets that it will initiate. Peer 2 also replies with its AdvertisedWindow size recommendation. Peer 1 naturally must acknowledge receipt of Peer 2's follow-up SYN-ACK. When TCP Peer 2 receives Peer 1's acknowledgement (ACK) message referencing its SYN-ACK message number and the agreed-upon AdvertisedWindow size, the session is established. Frames 9–11 in the packet trace illustrated in Figure 1-28 are a typical sequence of messages exchanged by two hosts to establish a TCP connection.

Byte sequence numbers The TCP packet header references two 32-bit sequence numbers, a Sequence Number and the Acknowledgement field. These are the relative byte number offsets of the current transmission streams that are being sent back and forth between the two host applications that are in session. At the start of each TCP session as part of establishing the connection, the peers exchange initial byte sequence numbers in their respective SYN messages. Because TCP supports full duplex connections, both host applications send SYN messages initially.

The Sequence Number field in the header is the relative byte offset of the first data byte in the current transmission. This Sequence Number field allows the receiver to slot an IP packet received out of order into the correct sequence. Because the sequence numbers are 32 bits wide, it is safe for TCP to assume any packets received with identical sequence numbers are duplicates because of retransmission. In case this assumption is not true for certain high-latency high-speed connections, the endpoints should turn on the timestamp option in TCP. Duplicates can safely be discarded by the receiver.

The Acknowledgment field acknowledges receipt of all bytes up to (but not including) the current byte offset. It is interpreted as the Next Byte a TCP peer expects to receive in this session. The receiver matches the Acknowledgement ID against the SequenceNumber field in the next message received. If the SequenceNumber is higher, the current message block is interpreted as being out of sequence, and the Acknowledgement field of the ACK message returned is unchanged. The Acknowledgement field is cumulative, specifically acknowledging receipt of all bytes from the Initial Sequence Number (ISN) +1 to the current Acknowledgement byte number –1. A receiver can acknowledge an out-of-sequence packet only when the SACK (Selective Acknowledgement) option is enabled.

Sliding window TCP provides a flow control mechanism called the *sliding window*, which determines the maximum amount of data a peer can transmit before receiving a specific acknowledgement from the receiver. This mechanism can be viewed as window that slides forward across the byte transmission stream. The AdvertisedWindow is the maximum size of the sliding window. The current Send Window is the AdvertisedWindow minus any transmitted bytes that the receiver has not yet Acknowledged. Once the window is filled and no Acknowledgement is forthcoming, the sender is forced to wait before sending any more data on that connection.

The AdvertisedWindow field in the TCP header is 16 bits wide, making 64 KB the largest possible window size. However, an optional Window Scale factor can also be specified, which is used to scale up the AdvertisedWindow field to support larger windows. The combination of the two fields allows TCP to support a sliding data window up to 1 GB wide.

TCP's sliding window mechanism has network capacity planning implications. Together, the Advertised Window size and the RTT establish an upper limit to the effective throughput of a TCP session. The default Advertised Window Windows Server 2003 uses for Ethernet connections is about 17,520 bytes. TCP session management can send only one 17.5 KB window's worth of data before stopping to wait for an ACK from the receiver. If the RTT of the session is 100 milliseconds (ms) for a long distance connection, the TCP session will be able to send a maximum of only 1/RTT windows per second, in this case, just 10 windows per second. The maximum throughput of that connection is effectively limited to

Max throughput = AdvertisedWindow / RTT

which in this case is 175 KB/sec, *independent of the link bandwidth.*

Consider a Fast Ethernet link at 100 Mb/sec, where the effective throughput capacity of the link is roughly 12 MB/sec (12,000,000 bytes/sec). RTT and the default Windows

Server 2003 TCP Advertised Window begin to limit the effective capacity of a Fast Ethernet link once RTT increases above 1 millisecond, as illustrated in Table 1-6.

Table 1-6 How Various RTT Values Reduce the Effective Capacity of a 100BaseT Link

RTT (ms)	Max Windows/sec	Max Throughput/sec
0.001	1,000,000	12,000,000
0.010	100,000	12,000,000
0.100	10,000	12,000,000
1	1,000	12,000,000
10	100	1,750,000
100	10	175,000
1000	1	17,500

Because the latency of a LAN connection is normally less than a millisecond (which is what ping on a LAN will tell you), RTT will not limit the effective capacity of a LAN session. However, it will have a serious impact on the capacity of a long distance connection to a remote Web server, where you can expect the RTT to be 10–100 milliseconds, depending on the distances involved. Note that this is the effective link capacity of a single TCP connection. On IIS Web servers with many connections active concurrently, the communications link can still saturate.

Windows Server 2003 defaults to using a larger window of about 65,535 bytes for a Gigabit Ethernet link. Assuming a window of exactly 65,000 bytes leads to the behavior illustrated in Table 1-7.

Table 1-7 How Various Values Reduce the Effective Capacity of a 1000BaseT Link

RTT (ms)	Max Windows/sec	Max Throughput/sec
0.001	1,000,000	125,000,000
0.010	100,000	125,000,000
0.100	10,000	125,000,000
1	1,000	65,000,000
10	100	6,500,000
100	10	650,000
1000	1	65,000

An Advertised Window value larger than 65535 bytes can be set for Gigabit Ethernet links using the Window scaling option. This is one of the networking performance options discussed in more detail in Chapter 6, “Advanced Performance Topics.” in this book.

Congestion window As part of congestion control, a TCP sender paces its rate of data transmission, slowly increasing it until a congestion signal is recognized. When a congestion signal is received, the sender backs off sharply. TCP uses *Additive increase/ Multiplicative decrease* to open and close the Send Window. It starts a session by sending two packets at a time and waiting for an ACK. TCP slowly increases its connection Send Window one packet at a time until it receives a congestion signal. When it recognizes a congestion signal, TCP cuts the current Send Window in half and then resumes additive increase. The operation of these two congestion mechanisms produces a Send Window that tends to oscillate, as illustrated in Figure 1-30, reducing the effective capacity of a TCP connection accordingly.

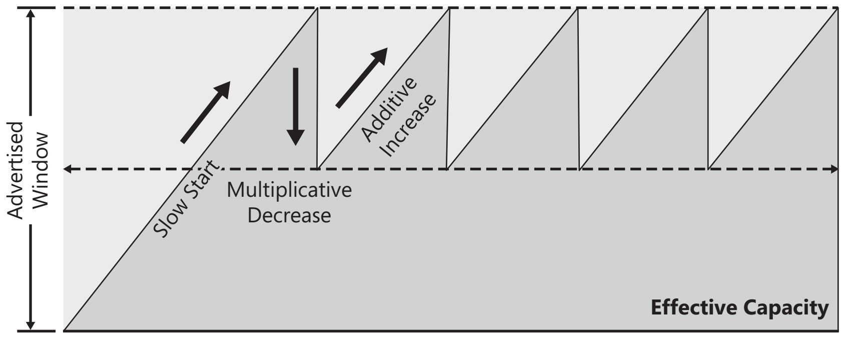


Figure 1-30 The TCP congestion window reducing effective network capacity

The impact of the TCP congestion window on effective network capacity is explored in more detail in Chapter 5, "Performance Troubleshooting."

Summary

This chapter introduced the performance monitoring concepts that are used throughout the remaining chapters of this book to describe the scalability and provisioning of Windows Server 2003 machines for optimal performance. It provided definitions for key computer performance concepts, including utilization, response time, service time, and queue time. It then discussed several mathematical relations from the queuing models that are often applied to problems related to computer capacity planning, including Utilization Law and Little's Law. The insights of mathematical Queuing Theory imply that it is difficult to balance efficient utilization of hardware resources with optimal response time. This tension between these two goals arises because, as utilization at a resource increases, so does the potential for services requests to encounter queue time delays at a busy device.

This chapter also discussed an approach to computer performance and tuning that is commonly known as *bottleneck analysis*. Bottleneck analysis decomposes a response time–oriented transaction into a series of service requests to computer subcomponents such as processors, disks, and network adaptors, that are interconnected in a network of servers and their queues. Bottleneck analysis then seeks to determine which congested resource adds the largest amount of queue time delay to the transaction as it traverses this set of interconnected devices. Once they have been identified, bottlenecks can be relieved by adding processing capacity, by spreading the load over multiple servers, or by optimizing the application's use of the saturated resource in a variety of straightforward ways.

The bulk of this chapter discussed the key aspects of computer systems architecture that are most important to performance and scalability. This included a discussion of the basic performance capabilities of the processor, memory, disk, and networking hardware that is most widely in use today. The key role that the Windows Server 2003 operating system plays in supporting these devices was also highlighted, including a discussion of the basic physical and virtual memory management algorithms that the operating system implements. The techniques used by the operating system to gather performance data on the utilization of these resources were also discussed to provide insight into the manner in which the key measurement data that supports performance troubleshooting and capacity planning is derived.

Chapter 2

Performance Monitoring Tools

In this chapter:

Summary of Monitoring Tools. 126
Performance Monitoring Statistics 133
System Monitor 144
Task Manager 154
Automated Performance Monitoring. 168
Managing Performance Logs. 191
Windows Performance Monitoring Architecture 207
Event Tracing for Windows 212
Alerts 225
Windows System Resource Manager 233
Network Monitor. 233

Microsoft® Windows Server™ 2003 provides a comprehensive set of tools to help with the collection of useful performance data. Four general classes of tools are available: performance statistics gathering and reporting tools, event tracing tools, load generating tools, and administration tools. The statistical tools that you will use regularly are the Performance Monitor and Task Manager, but you will have occasion to use other more specialized statistical tools to investigate specific performance problems.

Event tracing tools gather data about key, predefined system and application events. The event traces that you can gather document the sequence of events that take place on your Windows Server 2003 machine in great detail. Event tracing reports can be extremely useful in diagnosing performance problems, and they can also be used to augment performance management reporting in some critical areas.

This chapter discusses the use of these tools to gather performance statistics and diagnostic performance events. Specific performance statistics that should be gathered are discussed in detail in Chapter 3, "Measuring Server Performance." Performance monitoring procedures designed to support both problem diagnosis and longer term capacity planning are described in Chapter 4, "Performance Monitoroing Procedures."

Chapter 5, "Performance Troubleshooting," and Chapter 6, "Advanced Performance Topics," discuss the role that other key diagnostic tools can play in troubleshooting some of the difficult performance problems that you can encounter.

In addition to these essential performance monitoring tools, load testing should be used in conjunction with capacity planning to determine the capacity limits of your applications and hardware. How to use application load testing tools is beyond the scope of this chapter.

Summary of Monitoring Tools

There are three primary sources for the monitoring and diagnostic tools that are available for Windows Server 2003:

- Windows Server 2003 tools installed as part of the operating system
- The Windows Server 2003 Support Tools from the operating system installation CD
- The Windows Server 2003 Resource Kit tools

This chapter is mainly concerned with the monitoring and diagnostic tools that are automatically installed alongside the operating system.

Performance Statistics

The first group of tools to be discussed gathers and displays statistical data on an interval basis. These tools use a variety of sampling techniques to generate interval performance monitoring data that is extremely useful in diagnosing performance problems. The statistical tools in Windows Server 2003 are designed to be efficient enough that you can run them continuously with minimal impact. Using the tools that are supplied with Windows Server 2003, you should be able to establish automated data collection procedures. The performance statistics you gather using the Performance Monitor can also be summarized over longer periods of time to assist you in capacity planning and forecasting. Specific procedures designed to help you accomplish this are described in Chapter 5 of this book.

You will find that the same set of performance counters described here are available in many other tools. Other applications that access the same performance statistics include the Microsoft Operations Manager (MOM) as well as applications that have been developed by third parties. All these applications that gather Windows Server 2003 performance measurements share a common measurement interface—a performance monitoring application programming interface (API). The performance monitoring API is the common source of all the performance statistics these tools gather.

Event Traces

A comprehensive trace facility called Microsoft Event Tracing for Windows (ETW) gathers detailed event traces from operating system providers. Using ETW, you can determine precisely when context switches, file operations, network commands, page faults, or other system events occur. Event trace providers are also available for many server applications, including File Server, Internet Information Services (IIS), and Active Directory. ETW traces capture a complete sequence of events that allow you to reconstruct precisely what is occurring on your Windows Server 2003 machine and when. Event traces can also illuminate key aspects of application performance for those server applications like IIS and Active Directory that have also been instrumented. Event traces are one of the most important tools available for diagnosing performance problems.

Unlike statistical tools, event tracing is not designed to be executed continuously or to run unattended for long periods of time. Depending on what events you are tracing, ETW traces can easily generate significant overhead to the extent that they might interfere with normal systems operation. Consequently, event traces are a diagnostic tool normally reserved for gathering data about performance problems that require more detailed information than statistical tools can provide. Also, there is no easy way to summarize the contents of an event trace log, although standard reports are provided. In addition, it is possible to generate very large files in a relatively short time period. So long as you are careful about the way you use them, event traces are a valuable diagnostic tool that you will find invaluable in a variety of circumstances.

Another important trace tool is Network Monitor. With Network Monitor, you can determine the precise sequence of events that occur when your computer is communicating with the outside world. Network Monitor traces are often used to diagnose network connectivity and performance problems. But they are also useful to document the current baseline level at which your systems are operating so that you can recognize how your systems are growing and changing. Using Network Monitor to diagnose network performance problems is discussed in Chapter 5, “Performance Troubleshooting” in this book.

Finally, one more event-oriented administrative tool should be mentioned in this context. The event logging facility of the operating system can often play an important role in diagnosing problems. Using Event Viewer, you can access event logs that record a variety of events that can be meaningful when you are analyzing a performance problem. For example, one of the easiest ways to determine when processes begin and end in Windows Server 2003 is to enable audit tracking of processes and then examine process start and end events in the Security log.

Load Generating and Testing

Load-generating tools of all kinds are useful for stress testing specific hardware and software configurations. Windows Server 2003 includes several load- testing tools, including some for specific applications like Web-based transaction processing and Microsoft Exchange. Load-testing tools simulate the behavior of representative application users. As you increase the number of simulated users, you can also gather performance statistics to determine the impact on system resource utilization. These tools help you determine the capacity limits of the hardware you have selected because they allow you to measure application response time as the number of users increases. As discussed in Chapter 1, "Performance Monitoring Overview," application response time can be expected to increase as a function of load, and these tools allow you to characterize that nonlinear function very accurately for specific application workloads.

Load testing your workload precisely can become very complicated. A key consideration with load-testing tools is developing an artificial workload that parallels the real-life behavior of the application you are interested in stress testing. The fact that a particular hardware or software bottleneck arises under load might be interesting, but unless the simulated workload bears some relationship to the real workload, the test results might not be very applicable. Further discussion of the use of load-testing tools is beyond the scope of this chapter.

Administrative Controls

Finally, there are tools that provide system administrators with controls that allow you to manage complex Windows Server 2003 environments. Administrative tools are especially useful when you have two or more major applications running on the same server that are vying for the same, potentially overloaded resources. The Windows System Resources Manager (WSRM) is the most comprehensive administrative tool available for the Windows Server 2003 environment, but other administrative tools can also prove helpful. WSRM is discussed briefly in Chapter 6, "Advanced Performance Topics," in this book.

More Info Complete documentation about the use of WSRM is available at http://www.microsoft.com/windowsserver2003/technologies/management/wsrm/default.mspx.

Required Security for Tool Usage

Many monitoring tools will only work provided you have the appropriate security access. In previous versions of the operating system, access was controlled by setting security on the appropriate registry keys. Windows Server 2003 comes with two predefined security groups—Performance Monitor Users and Performance Log Users—that already have the prerequisite security rights for accessing the registry keys required by Performance Monitor.

You can add a designated support person into an appropriate performance security group by using the Active Directory Users and Computers tool when working in a domain. These two security groups are found in the Builtin container for the domain and are the following:

- **Performance Monitor Users** To allow designated support personnel access to remote viewing of system performance through the System Monitor tool, you will need to make each person a member of the Performance Monitor Users group.
- **Performance Log Users** To allow designated support personnel access to remote configuring of and using the Performance Logs and Alerts tool, you will need to make each person a member of the Performance Log Users group.

More Info For information about how to add or remove users from groups, see "Changing group memberships" in the Windows Server 2003 Help.

Table 2-1 contains a list of Windows Server 2003 operating system tools for monitoring performance.

More Info For more information about operating system tools, in Help and Support Center for Microsoft® Windows Server™ 2003, click **Tools**, and then click **Command-Line Reference A–Z**.

Table 2-1 Performance-Related Operating System Tools

Tool	Description
Freedisk.exe	Checks for a specified amount of free disk space, returning a 0 if there is enough space for an operation and a 1 if there isn't. If no values are specified, the tool tells you how much hard disk space you have left. Useful when performance monitoring disk usage.
Lodctr.exe	Loads performance counters. Especially useful for saving and restoring a set of registry enabled performance counters.
Logman.exe	Powerful command-line tool for collecting event trace and performance information in log files. Useful for scripting performance monitoring. Using Logman is discussed later in this chapter. Comprehensive performance monitoring procedures that incorporate Logman are described in Chapter 4, "Performance Monitoring Procedures," later in this book.

Table 2-1 **Performance-Related Operating System Tools**

Tool	Description
Msinfo32.exe	Provides information about the resources used by your system. Useful when requiring a snapshot of information about processes and the resources they are consuming.
Network Monitor	Comprehensive graphical tool for monitoring network traffic to and from the local server. Must be installed manually after the base operating system is loaded. Use of Network Monitor is discussed in Chapter 5, "Performance Troubleshooting," later in this book.
Performance Logs and Alerts	Creates a manually defined set of logs that can be used as baselines for your servers. The Alerts component allows you to take a specified action when specific conditions are encountered. Using Performance Logs and Alerts is discussed later in this chapter.
Relog.exe	Command-line tool that creates new performance logs from existing log files by varying the sample rate. Using Relog is discussed later in this chapter. Comprehensive performance monitoring procedures that incorporate Relog are described in Chapter 4, "Performance Monitoring Procedures," later in this book.
Systeminfo.exe	Provides detailed support information about computers
System Monitor	Main graphical tool used for monitoring system performance. Using System Monitor is discussed extensively throughout this book.
Taskkill.exe	Kills a specified process on the specified computer.
Tasklist.exe	Displays the process ID and memory information about all running processes.
Task Manager	Graphical tool that offers an immediate overview of the system and network performance of the local server only. Using Task Manager is discussed later in this chapter.
Tracerpt.exe	Command-line tool useful for converting binary log event trace files into a report or comma-separated value (CSV) files for importing into spreadsheet products such as Microsoft Excel. Using Tracerpt is discussed later in this chapter. Using Tracerpt in conjunction with troubleshooting performance problems is discussed in Chapter 5, "Performance Troubleshooting."
Typeperf.exe	Command-line tool that writes performance data to a log file. Useful for automating the performance monitoring process. Using Typeperf is discussed later in this chapter. Comprehensive performance monitoring procedures that incorporate Typeperf are described in Chapter 4, "Performance Monitoring Procedures."
Unlodctr.exe	Unloads performance counters. Especially useful for restoring a set of registry-enabled performance counters.

Table 2-2 contains a list of Windows Server 2003 Support Tools for monitoring performance. For more information about Windows Support Tools, in Help and Support

Center for Microsoft® Windows Server™ 2003, click **Tools**, and then click **Windows Support Tools**.

Table 2-2 Performance-Related Support Tools

Tool	Description
Depends.exe	The Depency Walker tool scans any 32-bit or 64-bit Windows module (including .exe, .dll, .ocx, and .sys, among others) and builds a hierarchical tree diagram of all dependent modules. Useful when knowledge of all files used by a monitored process is required.
Devcon.exe	The Device Configuration Utility displays all device configuration information and their current status. Useful when monitoring hardware performance.
Diruse.exe	Directory Usage will scan a directory tree and report the amount of space used by each user. Useful when tracking disk space issues.
Exctrlst.exe	Extensible Counter List is a graphical tool that displays all counter .dll files that are running and provides the capability to disable them. Using Exctrlst is discussed later in this chapter.
Health_chk.cmd	This script uses Ntfrsutl.exe to gather data from FRS on the target computer for later analysis. Useful when scripting remote monitoring.
Memsnap.exe	This memory-profiling tool takes a snapshot of the memory resources being consumed by all running processes and writes this information to a log file. Useful when performance monitoring a system's memory.
Netcap.exe	Command-line sniffer tool used to capture network packets. Useful when monitoring network performance.
Poolmon.exe	The Memory Pool Monitor tool monitors memory tags, including total paged and non-paged pool bytes. Poolmon is often used to help detect memory leaks. Using Poolmon is discussed in Chapter 5, "Performance Troubleshooting."
Pviewer.exe	Process Viewer is a graphical tool that displays information about a running process and allows you to stop (kill) processes and change process priority.
Replmon.exe	The Replication Monitor tool enables administrators to monitor Active Directory replication, synchronization, and topology.

Table 2-3 contains a list of Windows Server 2003 Resource Kit Tools for monitoring performance.

More Info For more information about Resource Kit tools, in Help and Support Center for Microsoft Windows Server 2003, click **Tools**, and then click **Windows Resource Kit Tools**. You can also download these tools directly from the Microsoft Download Center by going to http://www.microsoft.com/downloads and searching for "Windows Server 2003 Resource Kit Tools".

Table 2-3 Performance-Related Windows Resource Kit Tools

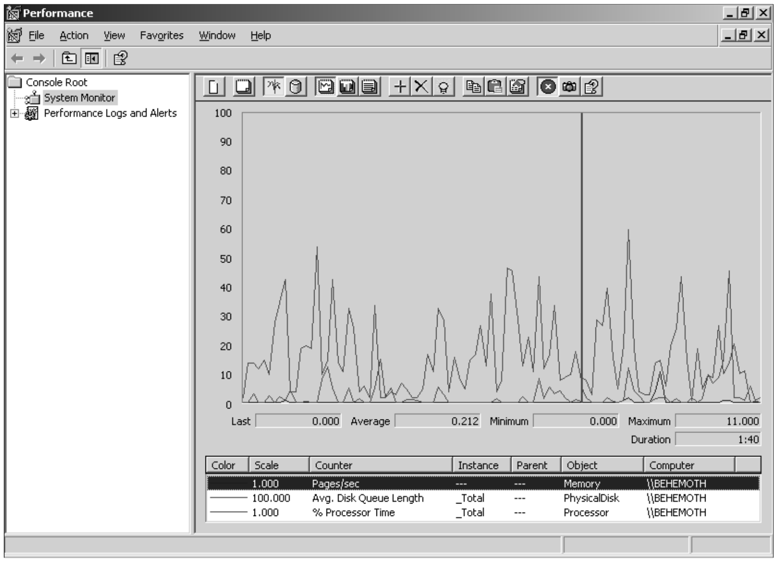
Tool	Description
Adlb.exe	Active Directory Load Balancing tool that balances the load imposed by Active Directory connection objects across multiple servers.
Checkrepl.vbs	Script to monitor replication and enumerate the replication topology for a given domain controller. Useful when monitoring specific network performance.
Clearmem.exe	Forces pages out of memory. Useful when testing and tracking memory performance issues.
Consume.exe	Consumes resources for stress testing performance such as low memory situations. The resources that can be appropriated include physical memory, page file memory, disk space, CPU time, and kernel pool memory. Examples of uses of Consume.exe are found in Chapter 5, "Performance Troubleshooting."
Custreasonedit.exe	Command-line and GUI tool that allows users to add, modify, and delete custom reasons used by the Shutdown Event Tracker on the Windows Server 2003 operating system.
DH.exe	Display Heap shows information about usage in a User mode process, or about pool usage in Kernel mode memory. A heap is a region of one or more pages that can be subdivided and allocated in smaller chunks. This is normally done by the heap manager, whose job is to allocate and deallocate variable amounts of memory.
Empty.exe	Frees the working set memory of a specified task or process.
Intfiltr.exe	The Interrupt-Affinity Filter (IntFiltr) allows a user to change the CPU affinity for hardware components that generate interrupts in a computer. See Chapter 6, "Advanced Performance Topics," for a discussion of the benefits this tool can provide on a large-scale multiprocessor machine.
Kernrate.exe	Kernrate is a sample-profiling tool meant primarily to help identify where CPU time is being spent. Both Kernel and User mode processes can be profiled separately or simultaneously. Useful when monitoring the performance of a process or device driver that is consuming excessive CPU time. See Chapter 5, "Performance Troubleshooting," for an example of how to use it to resolve performance problems involving excessive CPU usage.
Memtriage.exe	Detects a possible resource leak on a running system. Useful for monitoring memory leak, memory fragmentation, heap fragmentation, pool leaks, and handle leaks.
Pfmon.exe	Page Fault Monitor lists the source and number of page faults generated by an application's function calls. Useful when monitoring memory and disk performance. See Chapter 5, "Performance Troubleshooting," for an example of how to use it to resolve performance problems involving excessive application paging.
Pmon.exe	Process Resource Monitor shows each process and its processor and memory usage.

Table 2-3 **Performance-Related Windows Resource Kit Tools**

Tool	Description
Showperf.exe	Performance Data Block Dump Utility is a graphical tool that creates a dump of the contents of the Performance Data block so that you can view and debug the raw data structure of all loaded performance objects.
Splinfo.exe	Displays print spooler performance information on the screen. Useful when taking a snapshot of print spooler performance.
Srvinfo.exe	Displays a summary of information about a remote computer and the current services running on the computer.
TSSCalling.exe	Series of applets that can be used to create automated scripts to simulate interactive remote users for performance stress testing of Terminal Services environments.
Vadump	Details the current amount of virtual memory allocated by a process.
Volperf.dll	Enables administrators to use Performance Monitor to monitor shadow copies.

Performance Monitoring Statistics

The easiest way to get started with performance monitoring in Windows Server 2003 is to click **Performance** on the Administrative Tools menu and begin a real-time monitoring session using the Performance Monitor console, as illustrated in Figure 2-1.



Last 0.000 Average 0.212 Minimum 0.000 Maximum 11.000
Duration 1:40

Color	Scale	Counter	Instance	Parent	Object	Computer
	1.000	Pages/sec	---	---	Memory	\\BEHEMOTH
	100.000	Avg. Disk Queue Length	_Total	---	PhysicalDisk	\\BEHEMOTH
	1.000	% Processor Time	_Total	---	Processor	\\BEHEMOTH

Figure 2-1 The Performance Monitor console

When you first launch Performance Monitor, a System Monitor Chart View is activated with a default set of counters loaded for monitoring your local computer, as illustrated in Figure 2-1. The default display shows three of the potentially thousands of performance counter values that System Monitor can report on.

Performance Objects

Related performance statistics are organized into objects. For example, measurements related to overall processor usage, like Interrupts/sec and % User Time, are available in the Processor object.

Multiple Instances of Objects

There may be one or more instances of a performance object, where each instance is named so that it is uniquely identified. For example, on a machine with more than one processor, there is more than one instance of each set of processor measurements. Each processor performance counter is associated with a specific named instance of the Processor object. A Processor object has instances 0 and 1 for a 2-way multiprocessor that uniquely identify the processor instance. The instance name is a unique identifier for the set of counters related to that instance, as illustrated in Figure 2-2.

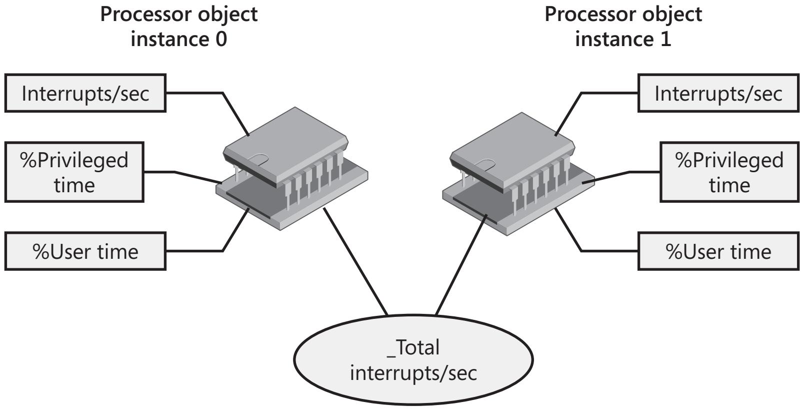


Figure 2-2 Objects, counters, and instances

Figure 2-2 gives an example of a computer containing two Processor objects. In the example, statistics from three counters (Interrupts/sec, %Privileged time, and %User Time) are being monitored for each instance of the Processor object. Also, the total number of Interrupts/sec on all processors in the system is being monitored.

Similarly, for each running process, a unique set of related performance counters are associated with that process instance. The instance name for a process has an additional index component whenever multiple instances of a process have the same process name. For example, you will see instances named svchost#1, svchost#2, and svchost#3 to distinguish separate instances of the svchost process.

The best way to visualize the relationships among object instances is to access the Add Counters dialog box by clicking the Plus Sign (+) button on the toolbar. Select the Thread object, and you will see something like the form illustrated in Figure 2-3. There are many instances of the Thread object, each of which corresponds to a process program execution thread that is currently active.

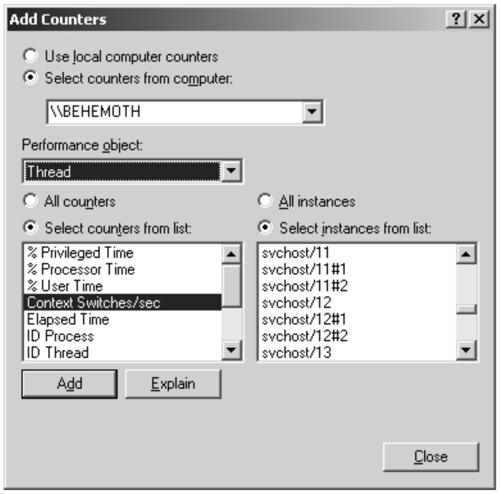


Figure 2-3 Thread object instances

Two objects can also have a parent-child relationship, which is illustrated in Figure 2-3. For example, all the threads of a single process are related. So thread instances, which are numbered to identify each uniquely, are all children of some parent process. The thread parent instance name is the process name that the thread is associated with. In Figure 2-3, thread 11 of the svchost#3 process identifies a specific thread from a specific instance of the svchost process.

Many objects that contain multiple instances also provide a _Total instance that conveniently summarizes the performance statistics associated with multiple instances.

Types of Performance Objects

Table 2-4 shows a list of the performance objects corresponding to hardware, operating system services, and other resources that are installed with Windows Server 2003. These objects and their counters can be viewed using the Add Counters dialog box of the Performance Monitor console. This is neither a default list nor a definitive guide. You might not see some of the performance objects mentioned here on your machine because these objects are associated with hardware, applications, or services that are not installed. You are also likely to see many additional performance objects that are not mentioned here but that you do have installed, such as those for measuring other applications such as Microsoft SQL Server.

Tip For a more comprehensive list of Windows Server 2003 performance objects and counters along with a list of which objects are associated with optional services and features, see the Windows Server 2003 Performance Counters Reference in the Windows Server 2003 Deployment Kit. You can view this Reference online at http://www.microsoft.com/resources/documentation/WindowsServ/2003/all/deployguide/en-us/counters_overview.asp. And for information about performance objects and counters for other Windows Server System products like Exchange Server and SQL Server, see the documentation for these products.

Table 2-4 **Windows Server 2003 Performance Objects**

Object Name	Description
ACS Policy	Provides policy-based Quality of Service (QoS) admission control data.
ACS/RSVP Interfaces	Reports the Resource Reservation Protocol (RSVP) or Admission Control Service (ACS) Interfaces performance counters.
ACS/RSVP Service	Reports the activity of the Quality of Service Admission Control Service (QoS ACS), which manages the priority use of network resources (bandwidth) at the subnet level.
Active Server Pages	Monitors errors, requests, sessions, and other activity data from Active Server Pages (ASP).
ASP .NET	Monitors errors, requests, sessions, and other activity data from ASP.NET requests.
AppleTalk	Monitors traffic on the AppleTalk network.
Browser	Reports the activity of the Browser service that lists computers sharing resources in a domain and other domain and workgroup names across the network. The Browser service provides backward compatibility with clients that are running Microsoft Windows 95, Microsoft Windows 98, Microsoft Windows 3.x, and Microsoft Windows NT.
Cache	Reports activity for the file system cache, an area of physical memory that holds recently used data from open files.
Client Service for Netware	Reports packet transmission rates, logon attempts, and connections to Netware servers.
Database	Reports statistics regarding the Active Directory database cache, files, and tables.

Table 2-4 **Windows Server 2003 Performance Objects**

Object Name	Description
Database Instances	Reports statistics regarding access to the Active Directory database and associated files.
DHCP Server	Provides counters for monitoring Dynamic Host Configuration Protocol (DHCP) service activity.
Distributed Transaction Coordinator	Reports statistics about the activity of the Microsoft Distributed Transaction Coordinator, which is a part of Component Services (formerly known as Transaction Server) and which coordinates two-phase transactions by Message Queuing.
DNS	Provides counters for monitoring various areas of the Domain Name System (DNS) to find and access resources offered by other computers.
Fax Service	Displays fax activity.
FileReplicaConn	Monitors performance of replica connections to the Distributed File System service.
FileReplicaSet	Monitors the performance of file replication service.
FTP Service	Includes counters specific to the File Transfer Protocol (FTP) Publishing Service.
HTTP Indexing Service	Reports statistics regarding queries that are run by the Indexing Service, a service that builds and maintains catalogs of the contents of local and remote disk drives to support powerful document searches.
IAS Accounting Clients	Reports the activity of Internet Authentication Service (IAS) as it centrally manages remote client accounting.
IAS Accounting Proxy	Reports the activity of the accounting proxy for Remote Authentication Dial-In User Service (RADIUS).
IAS Accounting Server	Reports the activity of Internet Authentication Service (IAS) as it centrally manages remote server accounting.
IAS Authentication Clients	Reports the activity of Internet Authentication Service (IAS) as it centrally manages remote client authentication.
IAS Authentication Proxy	Reports the activity of the RADIUS authentication proxy.
IAS Authentication Server	Reports the activity of Internet Authentication Service (IAS) as it centrally manages remote server authentication.
IAS Remote Accounting Server	Reports the activity of the RADIUS accounting server where the proxy shares a secret.
IAS Remote Authentication Server	Reports the activity of the RADIUS authentication server where the proxy shares a secret.
ICMP and ICMPv6	Reports the rate at which messages are sent and received by using Internet Control Message Protocol (ICMP), which provides error correction and other packet information.

Table 2-4 Windows Server 2003 Performance Objects

Object Name	Description
Indexing Service	Reports statistics pertaining to the creation of indexes and the merging of indexes by Indexing Service. Indexing Service indexes documents and document properties on your disks and stores the information in a catalog. You can use Indexing Service to search for documents, either by using the Search command on the Start menu or by using a Web browser.
Indexing Service Filter	Reports the filtering activity of Indexing Service. Indexing Service indexes documents and document properties on your disks and stores the information in a catalog. You can use Indexing Service to search for documents, either by using the Search command on the Start menu or by using a Web browser.
Internet Information Services Global	Includes counters that monitor Internet Information Services (IIS), which includes the Web service and the FTP service.
IPSec v4 Driver	Reports activity about encrypted.IPSec network traffic
IPSec v4 IKE	Reports IPSec security association information.
IPv4 and Ipv6	Reports activity at the Internet Protocol (IP) layer of Transmission Control Protocol/Internet Protocol (TCP/IP).
Job Object	Reports the data for accounting and processor use that is collected by each active, named job object.
Job Object Details	Reports detailed performance information about the active processes that make up a job object.
Logical Disk	Reports activity rates and allocation statistics associated with a Logical Disk file system.
Macfile Server	Provides information about a system that is running File Server for Macintosh.
Memory	Reports on the overall use of both physical (RAM) and virtual memory, including paging statistics.
MSMQ Queue	Monitors message statistics for selected queues.
MSMQ Service	Monitors session and message statistics.
MSMQ Session	Monitors statistics about active sessions.
NBT Connection	Reports the rate at which bytes are sent and received over connections that use the NetBIOS over TCP/IP (NetBT) protocol, which provides network basic input/output system (NetBIOS) support for TCP/IP between the local computer and a remote computer.
NetBEUI	Measures NetBIOS Enhanced User Interface (NetBEUI) data transmission.
NetBEUI Resource	Tracks the use of buffers by the NetBEUI protocol.

Table 2-4 **Windows Server 2003 Performance Objects**

Object Name	Description
Network Interface	Reports the rate at which bytes and packets are sent and received over a TCP/IP connection by means of network adapters.
NNTP Commands	Includes counters for all Network News Transfer Protocol (NNTP) commands processed by the NNTP service.
NNTP Server	Monitors posting, authentication, and connection activity on an NNTP server.
NTDS	Handles the Windows NT directory service on your system.
NWLink IPX	Measures datagram network traffic between computers that use the Internetwork Packet Exchange (IPX) protocol.
NWLink NetBIOS	Monitors IPX transport rates and connections.
NWLink SPX	Measures network traffic between computers that use the Sequenced Packet Exchange (SPX) protocol.
Objects	Reports data about system software objects such as events.
Paging File	Reports the current allocation of each paging file, which is used to back virtual memory allocations.
Pbserver Monitor	Reports activity on a phone book server.
Physical Disk	Reports activity on hard or fixed disk drives.
Print Queue	Reports statistics for print jobs in the queue of the print server. This object is new in Windows Server 2003.
Process	Reports the activity of the process, which is a software object that represents a running program.
Process Address Space	Monitors memory allocation and use for a selected process.
Processor	Reports the activity for each instance of the processor.
ProcessorPerformance	Reports on the activity of variable speed processors.
PSched Flow	Monitors flow statistics from the packet scheduler.
PSched Pipe	Monitors pipe statistics from the packet scheduler.
RAS Port	Monitors individual ports of the remote access device on your system.
RAS Total	Monitors all combined ports of the remote access device on your system.
Redirector	Reports activity for the redirector, which diverts file requests to network servers.
Server	Reports activity for the server file system, which responds to file requests from network clients.
Server Work Queues	Reports the length of queues and number of objects in the file server queues.

Table 2-4 **Windows Server 2003 Performance Objects**

Object Name	Description
SMTP NTFS Store Driver	Monitors Simple Mail Transport Protocol message activity that is associated with an MS Exchange client.
SMTP Server	Monitors message activity generated by the Simple Mail Transport Protocol (SMTP) service.
System	Reports overall statistics for system counters that track system up time, file operations, the processor queue length, and so on.
TCPv4 and TCPv6	Reports the rate at which Transmission Control Protocol (TCP) segments are sent and received.
Telephony	Reports the activity for telephony devices and connections.
Terminal Services	Provides Terminal Services summary information.
Terminal Services Session	Provides resource monitoring for individual Terminal sessions.
Thread	Reports the activity for a thread, which is the part of a process that uses the processor.
UDPv4 and UDPv6	Reports the rate at which datagrams are sent and received by using the User Datagram Protocol (UDP).
Web Service	Includes counters specific to the Web publishing service that is part of Internet Information Services.
Web Service Cache	Provides statistics on the Kernel mode and User mode caches that are used in IIS 6.0.
Windows Media Station Service	Provides statistics about the Windows Media Station service, which provides multicasting, distribution, and storage functions for Windows Media streams.
Windows Media Unicast Service	Provides statistics about the Windows Media Unicast service that provides unicasting functions for Advanced Streaming Format (ASF) streams.
WMI Objects	Reports the available classes of WMI objects

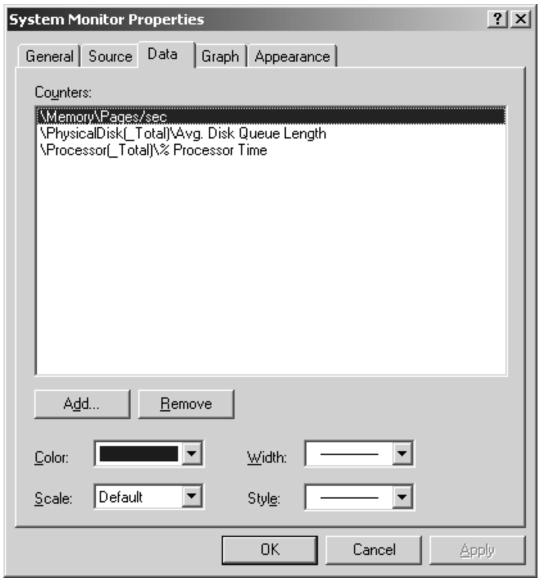
Performance Counters

The individual performance statistics that are available for each measurement interval are numeric counters. You can obtain an explanation about the meaning of a counter by clicking the Explain button shown in Figure 2-3.

Performance Counter Path

Each performance counter you select is uniquely identified by its path.

If you right-click **Chart View** in the System Monitor control of the Performance Monitor console to access the Properties of your console session, you will see listed on the Data tab the counters that are selected to be displayed, as shown in Figure 2-4. Each counter selected is identified by its path.



```
\Memory\Pages/sec
\PhysicalDisk(_Total)\Avg. Disk Queue Length
\Processor(_Total)\% Processor Time
```

Scale: Default

Figure 2-4 Data tab for System Monitor Properties

The following syntax is used to describe the path to a specified counter:

```
\\Computer_name\Object(Parent/Instance#Index)\Counter
```

The same syntax is also used consistently to identify the counters you want to gather using the Logman, Relog, and Typeperf command-line tools.

For a simple object like System or Memory that has only a single object instance associated with it, the following syntax will suffice:

```
\Object\Counter
```

For example, \Memory\Pages/sec identifies the Pages/sec counter in the Memory object.

The Computer_name portion of the path is optional; by default, the local computer name is assumed. However, you can specify the computer by name so that you can access counters from a remote machine.

The parent, instance, index, and counter components of the path can contain either a valid name or a wildcard character. For example, to specify all the counters associated with the Winlogon process, you can specify the counters individually or use a wildcard character (*):

```
\Process(winlogon)\*
```

Only some objects have parent instances, instance names, and index numbers that need to be used to identify them uniquely. You need to specify these components of the path only when they are necessary to identify the object instance you are interested in. Where a parent instance, instance name, or instance index is necessary to identify the counter, you can specify either each individual path or use a wildcard character (*) instead. This allows you to identify all the instances with a common path identification, without having to enumerate each individual counter path.

For example, the LogicalDisk object has an instance name, so you must provide either the name or a wildcard. Use the following format to identify all instances of the Logical Disk object:

```
\LogicalDisk(*)\*
```

To specify Logical Disk instances separately, use the following paths:

```
\LogicalDisk(C:)\*
\LogicalDisk(D:)\*
```

It is easy to think of the instance name, using this notation, as an index into an array of object instances that uniquely identifies a specific set of counters.

The Process object has an additional path component because the process instance name is not guaranteed to be unique. You would use the following format to collect the ID Process counter for each running process:

```
\Process(*)\ID Process
```

When there are multiple processes with the same name running that you need to distinguish, use the #Index identifier. For example, multiple instances of the svchost process would be identified as follows:

```
\Process(svchost)\% Processor Time
\Process(svchost#1)\% Processor Time
\Process(svchost#2)\% Processor Time
\Process(svchost#3)\% Processor Time
```

Notice that the first unique instance of a process instance name does not require an #Index identifier. The instance index 0 is hidden so that the numbering of additional instances starts with 1. You cannot identify multiple instances of the same process for monitoring unless you display instance indexes.

For the Thread object, which has a parent instance of the process to help identify it, the parent instance is also part of the path. For example, the following is the path that identifies counters associated with Thread 11 of the third instance of the svchost process.

```
\Process(svchost11#2)\Context switches/sec
```

If a wildcard character is specified in the parent name, all instances of the specified object that match the specified instance and counter fields will be returned. If a wildcard character is specified in the instance name, all instances of the specified object will be returned. If a wildcard character is specified in the counter name, all counters of the specified object are returned.

Partial counter path string matches (for example, svc*) are not supported.

Types of Counters

Each counter has a counter type. System Monitor (and similar applications) uses the counter type to calculate and present the counter value correctly. Knowing the counter type is also useful because it indicates how the performance statistic was derived. The counter type also defines the summarization rule that will be used to summarize the performance statistic over longer intervals using the Relog command-line tool.

The performance monitor API defines more than 20 specific counter types, some of which are highly specialized. The many different counter types fall into a few general categories, depending on how they are derived and how they can be summarized. Five major categories of counters are:

- **Instantaneous counters that display a simple numeric value of the most recent measurement** An instantaneous counter is a single observation or sample of the value of a performance counter right now. Instantaneous counters are always reported as an integer value.
  Instantaneous counters tell you something about the state of the machine right now. They do not tell you anything about what happened during the interval between two measurements. An example of an instantaneous counter is Memory\Available Bytes, which reports the current number of RAM pages in physical memory that is available for immediate use. Most queue length measurements are also instantaneous counters since they represent a single observation of the current value of the measurement. You can summarize such counters over longer intervals by calculating average values, minimums, maximums, and other summary statistics.
- **Interval counters that display an activity rate over time** Interval counters are derived from an underlying measurement mechanism that counts continuously the number of times some particular event occurs. The System Monitor retrieves the current value of this counter every measurement interval.
  Interval counters can also be thought of as difference counters because the underlying counter reports the current value of a continuously measured event. System Monitor retains the previous interval value and calculates the difference

between these two values. The difference is then usually expressed as a rate per second. Examples of this type of counter include Processor\Interrupts/sec and Logical and Physical Disk\Disk Transfers/sec. Some interval counters count timer ticks instead of events. These interval counters are transformed into % busy processor time measurements by dividing the timer tick difference by the total number of timer ticks in the interval. Interval counters can be summarized readily, reflecting average rates over longer periods of time.

- **Elapsed time counters** There are a few important elapsed time counters that measure System Up Time and Process\Elapsed time. These counters are gathered on an interval basis and cannot be summarized.
- **Averaging counters that provide average values derived for the interval** Examples of averaging counters include the hit rate % counters in the Cache object and the average disk I/O response time counters in the Logical and Physical Disk objects. These counters must be summarized carefully; make sure you avoid improperly calculating the average of a series of computed average values. You must calculate a weighted average over the summarization interval instead.
- **Miscellaneous complex counters including specialized counters that do not readily fall into any of the other categories** Similar to instantaneous counters, the complex counters are also single observations. Examples include Logical Disk\% Free Space, which is calculated by subtracting the number of allocated disk bytes from the total number of bytes available in the file system and expressing the result as a percentage of the whole. They also must be summarized carefully.

For more information about the way the Relog tool summarizes counter types, see “Summarizing Log Files Using Relog.”

Note Instantaneous counters like System\Processor Queue Length are always reported as integer values. They are properly viewed as single instances of a sample. You can summarize them by calculating average values and other summary statistics over longer intervals.

System Monitor

System Monitor is the main graphical tool used for real-time monitoring and analysis of logged data. The most common method of accessing System Monitor is by loading the Performance Monitor console.

To see procedures and a brief overview of the Performance Monitor console, click **Help** on the Performance Monitor toolbar. For information about remote monitoring, see Chapter 4, “Performance Monitoring Procedures.”

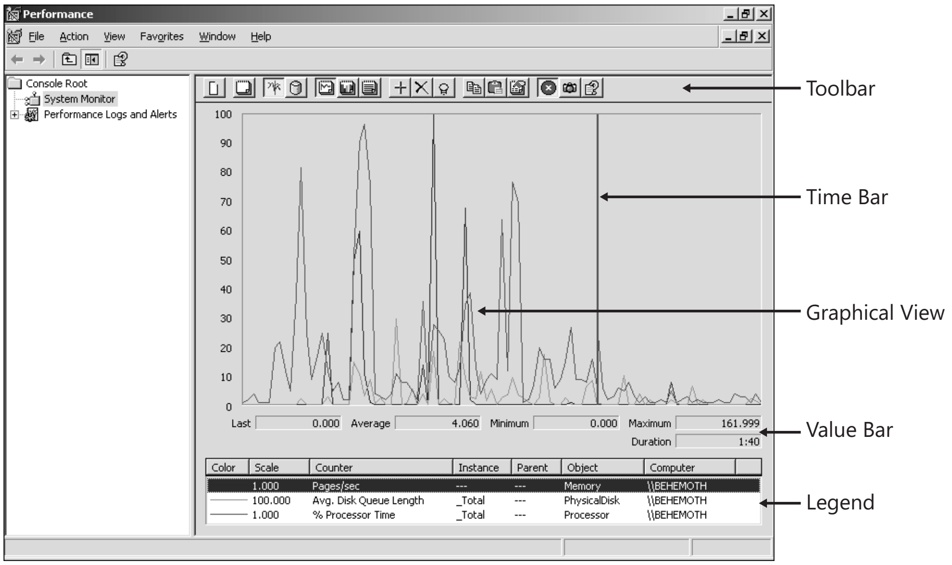
Warning Monitoring large numbers of counters can generate high overhead, potentially making the system unresponsive to keyboard or mouse input and impacting the performance of important server application processes. To reduce performance monitoring overhead, delete some of the counters you are collecting, reduce the sampling interval, or switch to a background data logging session using a binary logging file. For more information about background data logging, see "Performance Logs and Alerts" in this chapter.

Viewing a Chart in Real Time

To access the Performance Monitor console, select Performance from the Administrative Tools menu; or click **Start**, **Run**, and type **Perfmon.exe**.

When you start the Performance Monitor console, by default a System Monitor graph displays a default set of basic counters that monitor processor, disk, and virtual memory activity. These counters give you immediate information about the health of a system you are monitoring. The Chart View is displayed by default, but you can also create bar charts (histograms) and tabular reports of performance counter data using the Performance Monitor console.

When you run the Performance Monitor console, the Performance Logs and Alerts snap-in also appears beneath System Monitor in the console tree, as shown in Figure 2-5.



Last 0.000 Average 4.060 Minimum 0.000 Maximum 161.999
Duration 1:40

Color	Scale	Counter	Instance	Parent	Object	Computer
—	1.000	Pages/sec	---	---	Memory	\\BEHEMOTH
—	100.000	Avg. Disk Queue Length	_Total	---	PhysicalDisk	\\BEHEMOTH
—	1.000	% Processor Time	_Total	---	Processor	\\BEHEMOTH

Figure 2-5 System Monitor in the Performance console

The System Monitor display shown in Figure 2-5 contains the elements listed in Table 2-5.

Table 2-5 System Monitor Elements

Element	Description
Optional toolbar	This toolbar provides capabilities for adding and deleting counters from a graph. The toolbar buttons provide a quick way to configure the monitoring console display. You can also right-click the display to access a shortcut menu to add counters and configure your monitoring session Properties.
Graphical View	This displays the current values for selected counters. You can vary the line style, width, and color. You can customize the color of the window and the line chart itself, add a descriptive title, display chart gridlines, and change the display font, among other graphical options.
Value Bar	This displays the Last, Average, Minimum, and Maximum values for the counter value that is currently selected in the Legend. The value bar also shows a Duration value that indicates the total elapsed time displayed in the graph (based on the current update interval).
Legend	The Legend displays the counters selected for viewing, identified by the computer name, object, and parent and instance name. The line graph color and the scaling factor used to graph the counter value against the y-axis are also shown.
Time Bar	In real-time mode, this Time Bar moves across the graph from right to left to indicate the passing of each update interval.

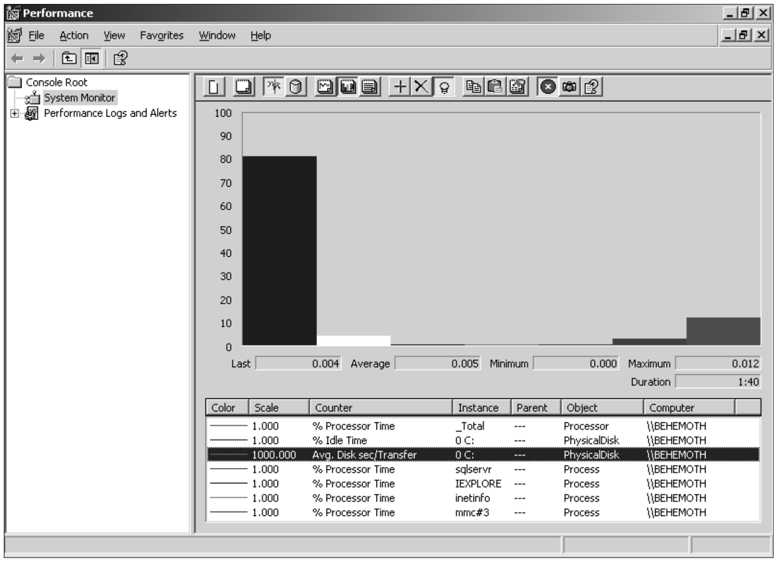
Changing the Sampling Interval

The default interval for a line graph in Chart View is once per second. You might find that using a 1-second sampling rate generates measurements too quickly to analyze many longer-term conditions. Change the sampling interval by accessing the General tab on the System Monitor Properties.

In a real-time monitoring session, the Chart View displays the last 100 observations for each counter. If your monitoring session has not yet accumulated 100 samples, the line charts you see will be truncated.

Caution Regardless of the update interval, a real-time Chart View can display no more than the last 100 observations for each counter value. When the Time Bar reaches the right margin of the display, the display wraps around to the beginning and starts overwriting data. If you change the sampling interval on the General Properties tab, the Duration field in the Chart View is updated immediately to reflect the change. However, until the Time Bar makes one complete revolution, the time range of the current display is a combination of the previous and current interval values.

A Histogram, or Report View, can display the last (or Current) value of each counter being monitored, or one of the following statistics: the minimum, maximum, or average value of the last 100 observations. Use the Legend in the Histogram View to identify what counter each bar in the graph corresponds to. Figure 2-6 illustrates the System Monitor Histogram View, which is an effective way to monitor a large number of counters at any one time.



Last 0.004 Average 0.005 Minimum 0.000 Maximum 0.012
Duration 1:40

Color	Scale	Counter	Instance	Parent	Object	Computer
———	1.000	% Processor Time	_Total	---	Processor	\\BEHEMOTH
———	1.000	% Idle Time	0 C:	---	PhysicalDisk	\\BEHEMOTH
———	1000.000	Avg. Disk sec/Transfer	0 C:	---	PhysicalDisk	\\BEHEMOTH
———	1.000	% Processor Time	sqlservr	---	Process	\\BEHEMOTH
———	1.000	% Processor Time	IEXPLORE	---	Process	\\BEHEMOTH
———	1.000	% Processor Time	inetinfo	---	Process	\\BEHEMOTH
———	1.000	% Processor Time	mmc#3	---	Process	\\BEHEMOTH

Figure 2-6 Histogram View helps montior many counters simultaneously

Creating a Custom Monitoring Configuration

After you select the counters you want to view and customize the System Monitor display the way you like it, you can save the System Monitor configuration for reuse at a later time.

To create a simple real-time monitoring configuration

1. Click the **New** button.
2. Click the Plus Sign (**+**) button.
3. In the **Add Counters** dialog box, select the performance objects and counters that you want to monitor.
4. Click **Add** for each one you want to add to the chart.
5. After selecting your counter set, click **Close** and watch the graph.
6. Click the **File** menu and click **Save As** to save the chart settings in a folder where you can easily reload your graph without having to reconfigure your counters.

You can also use the optional toolbar that appears by default to help you reconfigure your customized settings. Resting your mouse over any toolbar button will tell you the operation or action the button performs. Table 2-6 outlines the purpose of the tasks associated with each button more fully.

Table 2-6 System Monitor Toolbar Buttons

Task	Which Buttons To Use
Create a new chart	Use the New Counter Set button to create a new chart.
Refresh the data	Click Clear Display to clear the displayed data and obtain a fresh data sample for existing counters.
Conduct manual or automatic updates	Click Freeze Display to suspend data collection. Click Freeze Display again to resume data collection. Alternatively use the Update Data button for manual snapshots of data.
Select display options	Use the View Graph, View Histogram, or View Report option button. When you want to review data in a time series, the Chart View line graphs display a range of measured values continuously. Histograms and reports are useful for viewing a large number of Counter values. However, they display only a single value at a time, reflecting either current activity or a range of statistics.
Select a data source	Use the View Current Activity button for real-time data, or the View Log File Data button for data from either a completed or a currently running log.
Add, delete, reset, and get more information about counters	Use the Add or Delete buttons as needed. You can also use the New Counter Set button to reset the display and select new counters. Clicking the Add button displays the Add Counters dialog box, as shown in Figure 2-3. You can also press the Delete key to delete a counter that you select in the list box. When you are adding Counters, click the Explain button to read a more detailed explanation of a counter.
Highlighting	To accentuate the chart line or histogram bar for a selected counter with white (default) or black (for light backgrounds), click Highlight on the toolbar.
Import or export counter settings	To save the displayed configuration to the Clipboard for insertion into a Web page, click Copy Properties. To import counter settings from the Clipboard to the current System Monitor display, click Paste Counter List.
Configure other System Monitor properties	To access colors, fonts, or other settings that have no corresponding button on the toolbar, click the Properties button.

Saving Real-Time Data

If something of interest is in your real-time graph that you would like to spend more time investigating, you can save a copy of the current display by using the following steps:

1. Right-click your chart and click **Save As**.
2. Type the name of the file you want to create.
3. Select the format (chose HTML if you would like to view the full-time range of the counter values that are being displayed).
4. Click **Save**.

Note Because this is a real-time graph, you must click **Save** before the data you are interested in gets overwritten as the Time Bar advances. If you are likely to need access to performance data from either the recent period or historical periods, use Performance Logs and Alerts to write counter values to a log file.

The other commands available in the shortcut menu are listed in Table 2-7.

Table 2-7 **System Monitor Shortcut Menu Commands**

Command	Description
Add Counters	Use this command in the same way you use the Add button on the System Monitor toolbar.
Save As	Use this command if you want to save a snapshot of the current performance monitor statistics either in HTML format for viewing using a browser or as a .tsv file for incorporation into a report using a program such as Excel.
Save Data As	Use this command if you want to save log data in a different format or reduce the log size by using a greater sampling rate. This shortcut command is unavailable if you are collecting real-time data.
Properties	Click this command to access the property tabs that provide options for controlling all aspects of System Monitor data collection and display.

Customizing How Data Is Viewed

By default, real-time data is displayed in Chart View. Chart View is useful for watching trends. You can configure the elements shown such as the scale used, and the vertical and horizontal axis; and you can also change how the data is graphed.

The full range of configurable properties can be accessed by using the System Monitor Properties sheets. To open the System Monitor Properties dialog box, right-click on the right pane and select **Properties**, or click the **Properties** button on the toolbar.

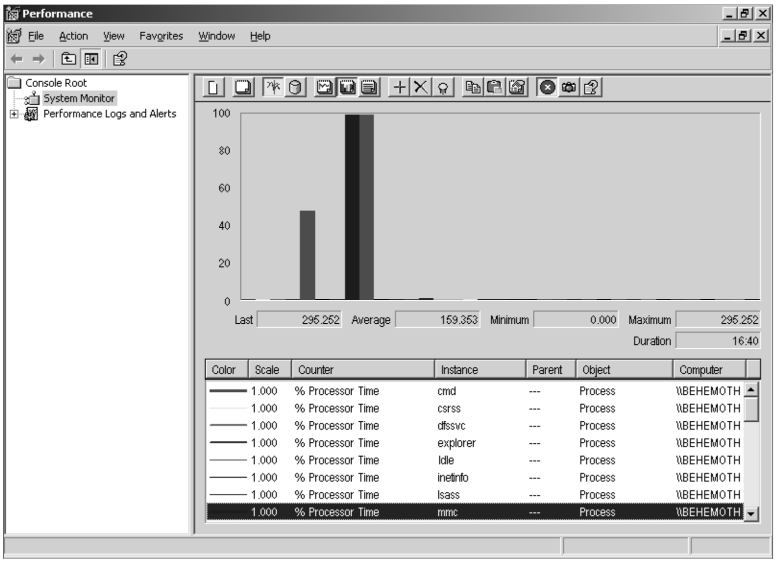
Table 2-8 lists the property tabs in the dialog box, along with the attributes they control.

Table 2-8 **System Monitor Properties**

Use This Tab	To Add or Change This
General	**View**: Choose between a graph (chart), histogram, or report.
	Display Elements: Show/hide the following: ■ Counter legend ■ Value bar: Displays the last, minimum, and maximum values for a selected counter ■ Toolbar
	Report and Histogram Data: Choose between default, minimum, average, current, and maximum report values.
	Appearance and **Border**: Change the appearance of the view. You can configure three-dimensional or flat effects, or include or omit a border for the window.
	Sample Automatically Every: Update sample rates. Specify sample rates in seconds.
	Allow Duplicate Counter Instances: Displays instance indexes (for monitoring multiple instances of a counter). The first instance (instance number 0) displays no index. System Monitor numbers subsequent instances starting with 1.
Source	**Data Source**: Select the source of data to display. **Current Activity**: Display the current data for the graph. **Log Files**: Display archived data from one or more log files. **Database**: Display current data from a Structured Query Language (SQL) database. **Time Range Button**: Display any subset of the time range for a log or database file.
Data	**Counters**: Add or remove objects, counters, and instances. **Color**: Change the color assigned to a counter. **Width**: Change the thickness of the line representing a counter **Scale**: Change the scale of the value represented by a counter. **Style**: Change the pattern of the line representing a counter.
Graph	**Title**: Type a title for the graph. **Vertical Axis**: Type a label for the vertical axis. **Show**: Select any of the following to show: ■ Vertical grid lines ■ Horizontal grid lines ■ Vertical scale numbers **Vertical Scale**: Specify a maximum and minimum (upper and lower limits, respectively) for the graph axes.
Appearance	**Color**: Specify color for Graph Background, Control Background, Text, Grid, and Time Bar when you click Change. **Font**: Specify Font, Font Style, and Size when you click Change.

In addition to using the Chart properties to change the view of the data, you can use the toolbar buttons:

- **View Histogram button** Click this button to change from the line graph Chart View to a bar chart Histogram View. Histogram View is useful at a deeper level when, for example, you are trying to track the application that is using most of the CPU time on a system.



Last 295.252 Average 159.353 Minimum 0.000 Maximum 295.252
Duration 16:40

Color	Scale	Counter	Instance	Parent	Object	Computer
	1.000	% Processor Time	cmd	---	Process	\\BEHEMOTH
	1.000	% Processor Time	csrss	---	Process	\\BEHEMOTH
	1.000	% Processor Time	dfssvc	---	Process	\\BEHEMOTH
	1.000	% Processor Time	explorer	---	Process	\\BEHEMOTH
	1.000	% Processor Time	Idle	---	Process	\\BEHEMOTH
	1.000	% Processor Time	inetinfo	---	Process	\\BEHEMOTH
	1.000	% Processor Time	lsass	---	Process	\\BEHEMOTH
	1.000	% Processor Time	mmc	---	Process	\\BEHEMOTH

Figure 2-7 Histogram View

- **View Report button** Click this button when you want to view data values only.

Tips for Working with System Monitor

Windows Server 2003 Help for Performance Monitoring explains how to perform common tasks, such as how to:

- Work with Counters
- Work with Data
- Work with Settings

Some useful tips for using the System Monitor are provided in this section.

Simplifying Detailed Charts

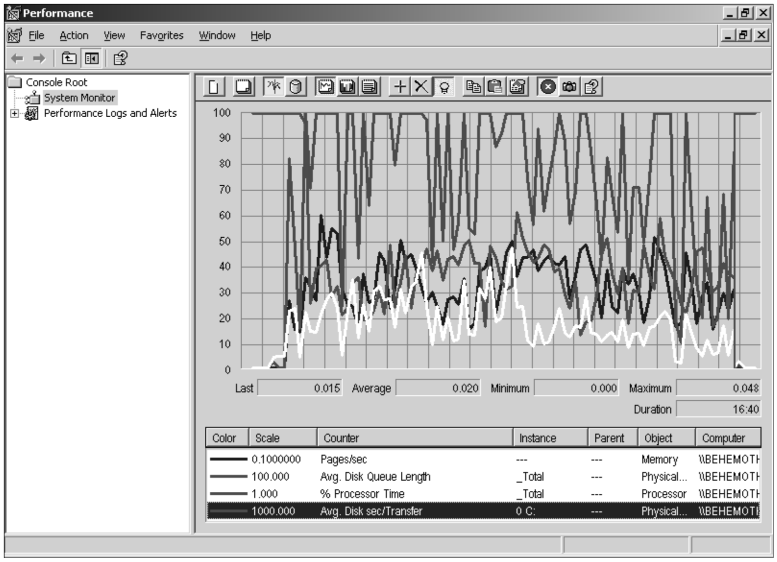
When working with Chart Views, you will find that simpler charts are usually easier to understand and interpret. If you have several counters to graph in Chart View, it can lead to a display that is jumbled and difficult to untangle. Some tips for simplifying complex charts to make them easier to understand include:

- In Chart View, use the Highlight button while you scroll up and down in the Legend to locate interesting counter values.
- Switch to the Report View or Histogram View to analyze data from many counters. When you switch back to Chart View, delete the counters from the chart that are cluttering up the display.
- Double-click an individual line graph in Chart View or a bar in Histogram View to identify that counter in the Legend.
- Widen the default width of line graphs that display the "interesting" counters that remain for better visibility.
- Run multiple copies of the Performance Monitor console if you need to watch more counters than are readily displayed on a single chart or to group them into categories for easier viewing.

Scaling the Y-Axis

All the counter values in a Chart View or Histogram View are graphed against a single y-axis. The y-axis displays values from 0 through 100 by default. The default y-axis scale works best for counter values that ordinarily range from 0 through 100. You can change the default minimum and maximum y-axis values using the Graph tab of the System Monitor Properties dialog. At the same time, you can also turn on horizontal and vertical grid lines to make the graph easier to read.

You might also need to adjust the scaling factor for some of the counters so that all counters you selected are visible in the graph area. For example, select the Avg. Disk sec/Transfer, as illustrated in Figure 2-8.



Last 0.015 Average 0.020 Minimum 0.000 Maximum 0.048
Duration 16:40

Color	Scale	Counter	Instance	Parent	Object	Computer
━	0.1000000	Pages/sec	---	---	Memory	\\BEHEMOTH
━	100.000	Avg. Disk Queue Length	_Total	---	Physical...	\\BEHEMOTH
━	1.000	% Processor Time	_Total	---	Processor	\\BEHEMOTH
━	1000.000	Avg. Disk sec/Transfer	0 C:	---	Physical...	\\BEHEMOTH

Figure 2-8 Scaling counter data to fit on the y-axis

Notice that the Avg. Disk sec/Transfer counter, which measures the average response time for an I/O to a physical disk, uses a default scaling factor of 1000. System Monitor multiplies the counter values by the scaling factor before it displays them on a line chart or histogram. Multiplying counter values for Avg. Disk sec/Transfer by 1000 normally allows disk response time in the range of 0–100 milliseconds to display against a default y-axis scale of 0–100.

On the Data tab of the System Monitor Properties dialog, you will see the Default scaling factor that was defined for each counter. If you click on this drop-down list, you will see a list of the available scaling factors. Select a scaling factor that will allow the counters to be visible within the display. If the counter values are too small to be displayed, use a higher number scaling factor. If the counter values are too large to be displayed, use a smaller scaling factor.

You might have to experiment using trial and error to arrive at the best scaling factor to use for each counter selected.

Sorting the Legend

The Counter, Instance, Parent, Object, and Computer columns in the Legend can be sorted. Click on the column heading in the Legend to sort the legend by the values in that column. Click on the column header to re-sort the display in the opposite direction.

Printing System Monitor Displays

You cannot print System Monitor charts and tabular reports directly from the Performance Monitor console. There are a number of other ways that you can print a System Monitor report, including:

- Ensure the System Monitor window is the active window. Copy the active window to the Clipboard by pressing ALT+PRINT SCREEN. Then you can open a Paint program, paste the image from the Clipboard, and print it.
- Add the System Monitor control to a Microsoft Office application, such as Microsoft Word or Microsoft Excel. Configure it to display data, and then print from that program.
- Save the System Monitor control as an HTML file by right-clicking the details pane of System Monitor, clicking Save As, and typing a file name. You can then open the HTML file and print it from Microsoft Internet Explorer or another program.

Table 2-9 provides a set of tips that you can use when working with system monitor.

Table 2-9 Tips for Working with System Monitor

Tip	Do This
Learn about individual counters	When adding counters, if you click Explain in the Add Counters dialog box in System Monitor, Counter Logs or Alerts, you can view counter descriptions.
Vary the data displayed in a report	By default, reports display only one value for each counter. This is the current data if the data source is real-time activity, or averaged data if the source is a log. However, on the General tab, you can configure the report display to show different values, such as the maximum, minimum, and average.
Select a group of counters or counter instances to monitor	In the Add Counters dialog box in System Monitor, you can do the following: ■ To select all counters or instances, click All Counters or All Instances. ■ To select specific counters or instances, click Select Counters from the list or Select Instances from the list. ■ To monitor a group of consecutive counters or instances in a list box, hold down the Shift key and click the first counter. Scroll down through the items in the list box and click the last counter you require. ■ To select multiple, nonconsecutive counters or instances, click each counter and press CTRL. It is a good idea to keep the Ctrl key pressed throughout this operation; if you do not, you can inadvertently lose previously selected counters.
Track totals for all instances of a counter	Instead of monitoring individual instances for a selected counter, you can use the _Total instance, which sums all instance values and reports them in System Monitor.
Pinpoint a specific counter from lines in a graph	To match a line in a graph with the counter for which it is charting values, double-click a position in the line. If chart lines are close together, try to find a point in the graph where they diverge.
Highlight the data for a specific counter	To highlight the data for a specific counter, use the highlighting feature. To do so, press CTRL+H or click Highlight on the toolbar. For the counter selected, a thick line replaces the colored chart line. For white or light-colored backgrounds (defined by the *Graph Background* property), this line is black; for other backgrounds, this line is white.

Task Manager

Sometimes a quick look at how the system is performing is more practical than setting up a complicated monitoring session. Task Manager is ideal for taking a quick look at how your system is performing. You can use it to verify whether there are any pressing performance issues and to obtain an overview of how key system components are functioning. You can also use Task Manager to take action. For example, after you determine that a runaway process is causing the system to become unresponsive, you can use Task Manager to end the process that is behaving badly.

Although Task Manager lacks the breadth of information available from the System Monitor application, it is useful as a quick reference to system operation and performance. Task Manager does provide several administrative capabilities not available with the System Monitor console, including the ability to:

- Stop running processes
- Change the base priority of a process
- Set the processor affinity of a process so that it can be dispatched only on particular processors in a multiprocessor computer
- Disconnect or log off a connected user session

Unlike the Performance Monitor application, Task Manager can only run interactively. Also, you are permitted to run only one instance of Task Manager at a time. You can use Task Manager only to view current performance statistics.

Most of the performance statistics that Task Manager displays correspond to counters you can also gather and view using the Performance Monitor, even though Task Manager sometimes calls similar values by slightly different names. Later in this section, Tables 2-10, 2-11, and 2-12 explain each Task Manager field and which Performance Monitor counter it corresponds to.

Warning If you are comparing statistics being gathered by both a Performance Monitor console session and Task Manager, don't expect these applications to report exactly the same values. At the very least, they are gathering performance measurement data at slightly different times, which helps explain some of the discrepancy when you compare two views of the same measurement.

Working with Task Manager

To start Task Manager, press CTRL+SHIFT+ESC or right-click the Taskbar and select Task Manager.

Task Manager displays different types of system information using various tabs. The display will open to the tab that was selected when Task Manager was last closed. Task Manager has five tabs: Applications, Processes, Performance, Networking, and Users. The status bar, at the bottom of the Task Manager window, shows a record of the number of Processes open, the CPU Usage, and the amount of virtual memory committed compared to the current Commit Limit.

The Task Manager display allows you to:

- Select Always On Top from the Options menu to keep the window in view as you switch between applications. The same menu option is also available by using the context menu on the CPU usage gauge in the notification area of the Taskbar.
- Press CTRL+TAB to toggle between tabs, or click the tab.
- Resize all Task Manager columns shown in the Networking, Processes, and Users tabs.
- Click a column in any view other than the Performance tab to sort its entries in ascending or descending order.

While Task Manager is running, a miniature CPU Usage gauge appears in the notification area of the Taskbar. When you point to this icon, the icon displays the percentage of processor use in text format. The miniature gauge, illustrated in Figure 2-9, matches the CPU Usage History chart on the Performance tab.

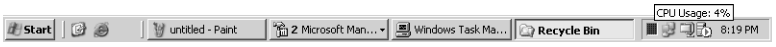


Figure 2-9 Task Manager CPU gauge shown in the notification area

If you would like to run Task Manager without its application window taking up space on the Taskbar, click **Hide When Minimized** on the Options menu. To open an instance of Task Manager when it is hidden, double-click the Task Manager CPU gauge in the notification area, press CTRL+SHIFT+ESC, or right-click the **Taskbar** and select **Task Manager**.

You can control the rate at which Task Manager updates its counts by setting the Update Speed option on the View menu. These are the possible speeds:

- **High** Updates every half-second.
- **Normal** Updates once per second. This is the default sampling rate.
- **Low** Updates every 4 seconds.
- **Paused** Does not update automatically. Press F5 to update.

A lower update speed reduces Task Manager overhead and thus lowers the sampling rate, which might cause some periodic spikes to go unnoticed. You can manually force an update at any time by clicking Refresh Now on the View menu or by pressing F5.

Monitoring Applications

The Applications tab lists the applications currently running on your desktop. You can also use this tab to start and end applications. Ending the application is just like clicking the X in the title bar of an application. The application then has a chance to prompt you to save information and clean up (flush buffers, close handles, cancel database transactions, close database connections, and so on). If you don't respond immediately to the prompt generated by the application when you try to end it, Task Manager prompts you to either cancel the operation or terminate the application immediately with possible loss of data.

Click the **Applications** tab to view the status of each open desktop application. The Applications tab includes the following additional features:

- To end an application, highlight the application and then click **End Task**.
- To switch to another application, highlight the application and then click **Switch To**.
- To start a new task, click **New Task**. In the Open text box that appears, type the name of the program you want to run.
- To determine what executable is associated with an application, right-click the task you want, and then click **Go To Process**.

Monitoring Processes

The Processes tab provides a tabular presentation of current performance statistics of all processes currently running on the system. You can select which statistics are displayed on this tab by clicking Select Columns on the View menu. You can also terminate processes from this tab. Terminating a process is much different from clicking the X in the application's title bar—the application is marked for termination. The application will not have a chance to save information or clean up before it is terminated. When you terminate an application process in this fashion, you can corrupt the files or other data that the application is currently manipulating and prevent that application from being restarted successfully. Verify that the process you are terminating is causing a serious performance problem by either monopolozing the processor or leaking virtual memory before you terminate it using this display.

Warning You should use the End Process function in Task Manager only to terminate a runaway process that is threatening the stability of the entire machine.

Note that you can terminate only those processes that you have the appropriate security access rights for. The attempt to terminate a service process running under a different security context from your desktop Logon ID will be denied for security reasons.

In Task Manager, click the **Processes** tab to see a list of running processes and their resource usage. Figure 2-10 is an example of how Task Manager displays process information. It shows some additional columns that have been turned on.

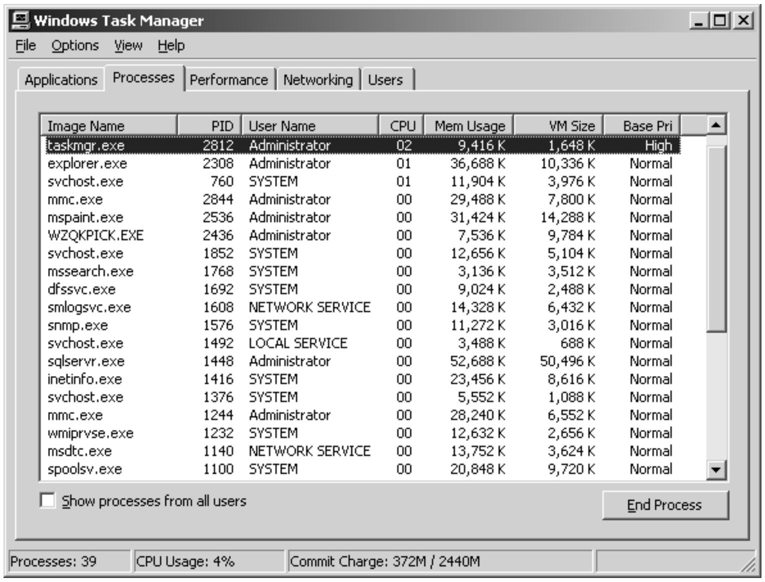


Image Name	PID	User Name	CPU	Mem Usage	VM Size	Base Pri
taskmgr.exe	2812	Administrator	02	9,416 K	1,648 K	High
explorer.exe	2308	Administrator	01	36,688 K	10,336 K	Normal
svchost.exe	760	SYSTEM	01	11,904 K	3,976 K	Normal
mmc.exe	2844	Administrator	00	29,488 K	7,800 K	Normal
mspaint.exe	2536	Administrator	00	31,424 K	14,288 K	Normal
WZQKPICK.EXE	2436	Administrator	00	7,536 K	9,784 K	Normal
svchost.exe	1852	SYSTEM	00	12,656 K	5,104 K	Normal
mssearch.exe	1768	SYSTEM	00	3,136 K	3,512 K	Normal
dfssvc.exe	1692	SYSTEM	00	9,024 K	2,488 K	Normal
smlogsvc.exe	1608	NETWORK SERVICE	00	14,328 K	6,432 K	Normal
snmp.exe	1576	SYSTEM	00	11,272 K	3,016 K	Normal
svchost.exe	1492	LOCAL SERVICE	00	3,488 K	688 K	Normal
sqlservr.exe	1448	Administrator	00	52,688 K	50,496 K	Normal
inetinfo.exe	1416	SYSTEM	00	23,456 K	8,616 K	Normal
svchost.exe	1376	SYSTEM	00	5,552 K	1,088 K	Normal
mmc.exe	1244	Administrator	00	28,240 K	6,552 K	Normal
wmiprvse.exe	1232	SYSTEM	00	12,632 K	2,656 K	Normal
msdtc.exe	1140	NETWORK SERVICE	00	13,752 K	3,624 K	Normal
spoolsv.exe	1100	SYSTEM	00	20,848 K	9,720 K	Normal

Processes: 39 | CPU Usage: 4% | Commit Charge: 372M / 2440M

Figure 2-10 Processes tab in Task Manager

> **Note** System Monitor displays memory allocation values in bytes, but Task Manager displays its values in kilobytes, which are units of 1,024 bytes. When you compare System Monitor and Task Manager values, divide System Monitor values by 1,024.

To include 16-bit applications in the display, on the Options menu, click **Show 16-Bit Tasks**.

Table 2-10 shows how the data displayed on the Task Manager Processes tab compares to performance counters displayed in System Monitor in the Performance console. Not all of these columns are displayed by default; use the Select Columns command on the View menu to add columns to or remove columns from the Task Manager display.

Table 2-10 Task Manager Processes Tab Compared to the Performance Console

Task Manager Column	System Monitor Counter	Description
PID (Process Identifier)	Process(*)\ID Process	The numerical identifier assigned to the process when it is created. This number is unique to a process at any point in time. However, after a process has been terminated, the process ID it was assigned might be reused and assigned to another process.
CPU Usage (CPU)	Process(*)\% Processor Time	The percentage of time the processor or processors spent executing the threads in this process.

Table 2-10 **Task Manager Processes Tab Compared to the Performance Console**

Task Manager Column	System Monitor Counter	Description
CPU Time	Not available	The total time the threads of this process used the processor since the process was started.
Memory Usage	Process(*)\Working Set	The amount of physical memory currently allocated to this process. This counter includes memory that is allocated specifically to this process in addition to memory that is shared with other processes.
Memory Usage Delta	Not available	The change in memory usage since the last update.
Peak Memory Usage	Process(*)\ Working Set Peak	The maximum amount of physical memory used by this process since it was started.
Page Faults	Not available	The number of page faults caused by this process since it was started.
Page Faults Delta	Process(*)\ Page Faults/sec	The change in the value of the Page Faults counter since the display was last updated.
USER Objects	Not available	The number of objects supplied to this process by the User subsystem. Some examples of USER objects include windows, menus, cursors, icons, and so on.
I/O Reads	Not available	The number of read operations performed by this process since it was started. Similar to the Process(*)\IO Read Operations/sec counter value, except Task Manager reports only a cumulative value.
I/O Read Bytes	Not available	The number of bytes read by read operations performed by this process since it was started. Similar to the Process(*)\IO Read Bytes/sec counter value, except Task Manager reports only a cumulative value.
Session ID	Terminal Services Session instance name	This counter is meaningful only when Terminal Services is installed. When installed, the counter displays the Terminal Services session ID that is running this process. This number is unique to the session at any point in time; however, after a session has been terminated, the session ID it was originally assigned might be reused and assigned to another process.

Table 2-10 Task Manager Processes Tab Compared to the Performance Console

Task Manager Column	System Monitor Counter	Description
User Name	Not available	The account the process was started under.
Virtual Memory Size	Process(*)\Virtual Bytes	The amount of virtual memory allocated to this program.
Paged Pool	Process(*)\Pool Paged Bytes	The amount of pageable system memory allocated to this process.
Non-pages Pool	Process(*)\Pool Nonpages Bytes	The amount of nonpageable system memory allocated to this process.
Base Priority	Process(*)\Priority Base	The base priority assigned to this process. Performance Monitor displays priority as a number, whereas Task Manager uses a name. The names used by Task Manager are Low, Below Normal, Normal, Above Normal, High, and Realtime. These correspond to priority levels 4, 6, 8, 10, 13, and 24, respectively.
Handle Count	Process(*)\Handle Count	The number of open handles in this process.
Thread Count	Process(*)\Thread Count	The number of threads that make up this process.
GDI Objects	Not available	The number of Graphics Device Interface (GDI) objects in use by this process. A GDI object is an item provided by the GDI library for graphics devices.
I/O Writes	Not available	The number of write operations performed by this process since it was started. Similar to the Process(*)\IO Write Operations/sec counter value, except Task Manager reports only a cumulative value.
I/O Write Bytes	Not available	The number of bytes written by write operations performed by this process since it was started. Similar to the Process(*)\IO Write Bytes/sec counter value, except Task Manager reports only a cumulative value.

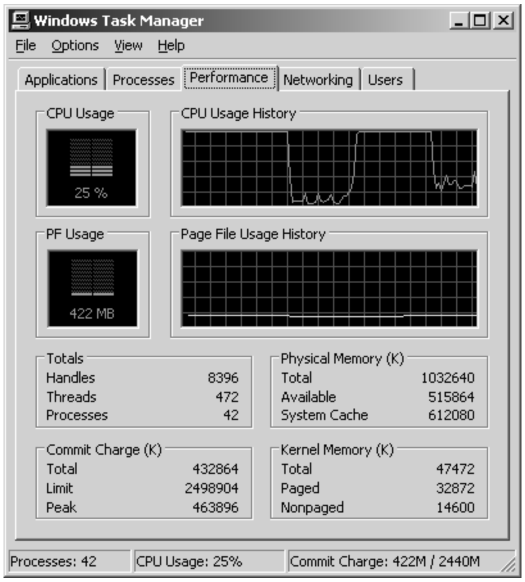
Table 2-10 Task Manager Processes Tab Compared to the Performance Console

Task Manager Column	System Monitor Counter	Description
I/O Other	Not available	The number of input/output operations that are neither read nor write. An example of this would be a command input to a part of the running process. Similar to the Process(*)\IO Other Operations/sec counter value, except Task Manager reports only a cumulative value.
I/O Other Bytes	Not available	The number of bytes used in other I/O operations performed by this process since it was started. Similar to the Process(*)\IO Other Bytes/sec counter value, except Task Manager reports only a cumulative value.

Monitoring Performance

The Performance tab displays a real-time view of performance data collected from the local computer, including a graph and numeric display of processor and memory usage. Most of the data displayed on this tab can also be displayed using Performance Monitor.

In Task Manager, to see a dynamic overview of system performance, click the **Performance** tab, as shown in Figure 2-11.



```
Totals                     Physical Memory (K)
Handles        8396        Total          1032640
Threads         472        Available       515864
Processes        42        System Cache    612080

Commit Charge (K)          Kernel Memory (K)
Total        432864        Total            47472
Limit       2498904        Paged            32872
Peak         463896        Nonpaged         14600

Processes: 42   CPU Usage: 25%   Commit Charge: 422M / 2440M
```

Figure 2-11 Performance tab in Task Manager

The following additional display options are available on the Performance tab:

- Double-clicking Task Manager will display a window dedicated to the CPU usage gauges. Using this mode on the Performance tab, you can resize and change the location of the CPU usage gauges. To move and resize the chart, click the edge of the gauge and adjust the size, and then drag the window to the new location. To return to Task Manager, double-click anywhere on the gauge.
- To graph the percentage of processor time in privileged or Kernel mode, click Show Kernel Times on the View menu. This is a measure of the time that applications are using operating system services. The difference between the time spent in Kernel mode and the overall CPU usage represents time spent by threads executing in User mode.

Table 2-11 shows how the data displayed on the Task Manager Performance tab compares to performance counters displayed in System Monitor.

Table 2-11 Task Manager Performance Tab Compared to the Performance Console

Task Manager Field	System Monitor Counter	Description
CPU Usage (bar graph and chart)	Processor(_Total)\% Processor Time	The percentage of time the processor or processors were busy executing application or operating system instructions. This counter provides a general indication of how busy the computer is.
PF Usage (bar graph) Page File Usage History	Memory\Committed Bytes	A graphical representation of the Commit Charge Total. This is the total amount of virtual memory in use at that instant.
Handles	Process(_Total)\Handle Count	The total number of open handles in all processes currently running on the system.
Threads	System\Threads	The total number of threads currently running on the system.
Processes	System\Processes	The number of running processes.
Physical Memory - Total	Not available	The total amount of physical memory installed and recognized by the operating system.
Physical Memory - Available	Memory\Available KBytes	The amount of physical memory that can be allocated to a process immediately.

Table 2-11 Task Manager Performance Tab Compared to the Performance Console

Task Manager Field	System Monitor Counter	Description
Physical Memory - System Cache	Memory\ Cache Bytes	The amount of physical memory allocated to the system working set; includes System Cache Resident Bytes, Pool Paged Resident Bytes, System Driver Resident Bytes, and System Code Resident Bytes.
Commit Charge - Total	Memory\ Committed Bytes	The amount of virtual memory allocated and in use (that is, committed) by all processes on the system.
Commit Charge - Limit	Memory\Commit Limit	The maximum amount of virtual memory that can be committed without enlarging the paging file.
Commit Charge - Peak	Not available	The most virtual memory that has been committed since system startup.
Kernel Memory - Total	Not available	The sum of nonpaged and paged system memory from the Nonpaged and paged memory pools that is currently in use.
Kernel Memory - Paged	Memory\Pool Paged Resident Byes	The amount of system memory from the paged pool that is currently resident.
Kernel Memory - Nonpaged	Memory\Pool Nonpaged Bytes	The amount of system memory from the Nonpaged pool that is currently in use.

Monitoring the Network

The Networking tab displays the network traffic to and from the computer by network adapter. Network usage is displayed as a percentage of theoretical total available network capacity for that network adapter, RAS connection, established VPN, or other network connection.

Click the Networking tab to see the status of the network. Table 2-12 shows how the data displayed on the Task Manager Networking tab compares to performance counters in System Monitor. Not all of these columns are displayed by default; use the Select Columns command on the View menu to add columns to the Task Manager display.

Table 2-12 **Comparing the Task Manager Networking Tab with the Performance Console**

Task Manager Column	System Monitor Counter	Description
Adapter Description	Network Interface(instance)	The name of the network connection being monitored. This is used as the instance name of the performance object for this network adapter.
Network Adapter Name	Not available	The name of the Network Connection.
Network Utilization	Not available	The Network Interface(*)\Bytes Total/sec counter divided by the Network Interface(*)\Current Bandwidth counter and displayed as a percentage.
Link Speed	Network Interface(*)\Current Bandwidth	The theoretical maximum bandwidth of this network adapter.
State	Not available	The current operational state of this network adapter.
Bytes Sent Throughput	Not available	The rate that bytes were sent from this computer using this network adapter, divided by the Link Speed and displayed as a percentage. This value is based on the same data that is used by the Network Interface(*)\Bytes Sent/sec performance counter.
Bytes Received Throughput	Not available	The rate that bytes were received by this computer using this network adapter, divided by the Link Speed and displayed as a percentage. This value is based on the same data that is used by the Network Interface(*)\Bytes Received/sec performance counter.
Bytes Throughput	Not available	The rate that bytes were sent from or received by this computer using this network adapter divided by the Link Speed and displayed as a percentage. This value is based on the same data that is used by the Network Interface(*)\Bytes Total/sec performance counter.

Table 2-12 **Comparing the Task Manager Networking Tab with the Performance Console**

Task Manager Column	System Monitor Counter	Description
Bytes Sent	Not available	The total number of bytes sent from this computer since the computer was started. This value is based on the same data that is used by the Network Interface(*)\Bytes Sent/sec performance counter.
Bytes Received	Not available	The total number of bytes received by this computer since the computer was started. This value is based on the same data that is used by the Network Interface(*)\Bytes Received/sec performance counter.
Bytes	Not available	The total number of bytes sent from or received by this computer since the computer was started. This value is based on the same data that is used by the Network Interface(*)\Bytes Total/sec performance counter.
Bytes Sent/Interval	Network Interface(*)\Bytes Sent/sec	The number of bytes sent from this computer during the sample interval.
Bytes Received/Interval	Network Interface(*)\Bytes Received/sec	The number of bytes received by this computer during the sample interval.
Bytes/Interval	Network Interface(*)\Bytes Total/sec	The number of bytes sent from or received by this computer during the sample interval.
Unicasts Sent	Not available	The total number of unicast packets sent from this computer since the computer was started. This value is based on the same data that is used by the Network Interface(*)\Packets Sent Unicast/sec performance counter.

Table 2-12 **Comparing the Task Manager Networking Tab with the Performance Console**

Task Manager Column	System Monitor Counter	Description
Unicasts Received	Not available	The total number of unicast packets received by this computer since the computer was started. This value is based on the same data that is used by the Network Interface(*)\Packets Received Unicast/sec performance counter.
Unicasts	Not available	The total number of unicast packets sent from or received by this computer since the computer was started.
Unicasts Sent/ Interval	Network Interface(*)\Packets Sent Unicast/sec	The number of unicast packets sent from this computer during the sample interval.
Unicasts Received/ Interval	Network Interface(*)\Packets Received Unicast /sec	The number of unicast packets received by this computer during the sample interval. This value is based on the same data that is used by the performance counter.
Unicasts/Interval	Not available	The number of unicast packets sent from or received by this computer during the sample interval.
Nonunicasts Sent	Not available	The total number of nonunicast packets sent from this computer since the computer was started. This value is based on the same data that is used by the Network Interface(*)\Packets Sent Non-Unicast/sec performance counter.
Nonunicasts Received	Not available	The total number of nonunicast packets received by this computer since the computer was started. This value is based on the same data that is used by the Network Interface(*)\Packets Received Non-Unicast/sec performance counter.
Nonunicasts	Not available	The total number of nonunicast packets sent from or received by this computer since the computer was started.
Nonunicasts Sent/ Interval	Network Interface(*)\Packets Sent Non-Unicast/sec	The number of nonunicast packets sent from this computer during the sample interval.

Table 2-12 Comparing the Task Manager Networking Tab with the Performance Console

Task Manager Column	System Monitor Counter	Description
Nonunicasts Received/Interval	Network Interface(*)\Packets Received Non-Unicast / sec	The number of nonunicast packets received by this computer during the sample interval.
Nonunicasts/Interval	Not available	The number of nonunicast packets sent from or received by this computer during the sample interval.

Monitoring Users

The Users tab displays those users currently logged on to the computer. You can use this display to quickly identify the users logged on to that system.

Click the Users tab to see the status of all users currently logged on. To select the columns you want to view, on the View menu, click Select Columns. Table 2-13 gives a description of each of these columns.

Table 2-13 Task Manager Columns for Users

Task Manager Column	Description
User	Name of person logged on.
ID	Number that the user session is identified by.
Status	Active or Disconnected.
Client Name	Name of computer using the session. If it is a local session, this field will appear blank.
Session	Console or Terminal Services.

Using Task Manager you can perform a number of user-related tasks. These include the ability to:

- End a User session (Console session or Terminal session). Select the session, and then click the Disconnect button. This allows you to disconnect a user while the user's session continues to run the applications.
- Close a session and all applications that are running on that session. Select the session and click Log Off.

Warning If the user has any unsaved data, that data might be lost.

- Send a message to a specific user. Select the user, and then click Send Message.
- Take remote control of a Terminal Services session. Right-click a user, and then select remote control.

Automated Performance Monitoring

Performance problems do not occur only when you have the opportunity to observe them. Background performance monitoring procedures provide you with a way to diagnose performance problems that occurred in the recent past while you were not looking. This section documents the two automated tools in Windows Server 2003 that allow you to establish background performance monitoring procedures.

The first method uses the Counter Logs facility in the Performance Logs and Alerts section of the Performance Monitor console, which has a graphical user interface. The second method uses the Logman, Typeperf, and Relog command-line tools. The Counter Logs facility in the Performance Logs and Alerts section of the Performance Monitor console are discussed first. Then, performance monitoring command-line utilities and some of their additional capabilities will be discussed.

These are the keys to establishing effective automated performance monitoring procedures:

- Knowing which current performance statistics you want to collect on a regular basis
- Knowing in what form and how frequently you want to collect the data
- Knowing how much historical performance data you need to keep to go back in time and resolve a problem that occurred in the recent past or to observe historical trends

This section examines the data logging facilities you will use to gather performance statistics automatically. Chapter 3, "Measuring Server Performance," provides advice about what metrics to gather and how to interpret them. Chapter 4, "Peformance Monitoring Procedures," offers recommendations for setting up automated performance logging procedures.

Performance Logs and Alerts

When you open the Performance Monitor, you will notice the Performance Logs and Alerts function in the left tree view. There are three components to Performance Logs and Alerts: counter logs, trace logs, and alerts. This section discusses only the use of counter logs. Trace logs and alerts are discussed later in this chapter in sections entitled "Event Tracing for Windows" and "Alerts," respectively.

You create counter logs using the Performance Logs and Alerts tool whenever you require detailed analysis of performance statistics for your servers. Retaining, summarizing, and analyzing counter log data collected over a period of several months is beneficial for capacity planning and deployment. Using the Performance Logs and Alerts tool, designated support personnel can:

- Manage multiple logging sessions from a single console window.
- Start and stop logging manually, on demand, or automatically, at scheduled times for each log.
- Stop each log based on the elapsed time or the current file size.
- Specify automatic naming schemes and stipulate that a program be run when a log is stopped.

The Performance Logs and Alerts service process, Smlogsvc.exe, is responsible for executing the performance logging functions you have defined. Comparable performance data logging capabilities are also available using the command-line tools that also interface with the Performance Logs and Alerts service process. These command-line tools are discussed in "Creating Performance Logs Using Logman" in this chapter.

Counter Logs

Counter logs record to a log file the same performance statistics on hardware resource usage and system services that you can gather and view interactively in the System Monitor. Both facilities gather performance objects and counters through a common performance monitoring API. Counter logs are suited for gathering much more information than an interactive System Monitor console session. With the performance statistics stored in a log file, you no longer have to worry about newer counter values replacing older performance data values in an interactive data gathering session. All the interval performance data that was gathered for the duration of the counter logging session is available for viewing and reporting.

After you create a counter log file, you can use the System Monitor to view and analyze the performance data you collected. To access counter data from a log instead of viewing counters interactively, use the View Log Data button in System Monitor. You can use the System Monitor both during and after log file data collection to view the performance data you have collected.

Counter Log File Formats

Data in counter logs can be saved in the file formats shown in Table 2-14.

Table 2-14 Counter Log File Formats

File Format	File Type	Description
Binary	.blg	Binary log format. This format is used if another format is not specified. When a binary log file reaches its maximum size, data collection stops.
Binary circular	.blg	Circular binary format. This file format is the same as binary format. However, when a circular binary log file reaches its maximum size, data collection continues, wrapping around to the beginning of the file. Once the file is filled with data, the oldest data in the file is overwritten as new data is recorded.
Text (comma-separated)	.csv	Comma-separated values. This format is suitable for creating performance data in a format compatible with spreadsheet programs like Excel.
Text (tab-separated)	.tsv	Tab-separated values. This format is also suitable for creating performance data in a format compatible with spreadsheet programs like Excel.
SQL database	System DSN	SQL database format. This is valid only if data is being logged to a SQL database.

Binary log files are the recommended file format for most counter logging, especially any logging sessions in which you expect a sizable amount of data. They are the most efficient way to gather large amounts of performance data, and they store counter data more concisely than any other format. Because binary log files are readily converted into the other formats that are available, there is very little reason not to use binary log files initially. Use the Relog command-line tool to convert binary format log files to other formats as needed.

Note Binary file format is designed to be used by System Monitor. To interchange performance data with other applications like Microsoft Excel, use text file format. The precise format of a binary log file is open and documented. You could write a program to read a binary log file using the Performance Data Helper interface.

Binary circular files have the same format as binary linear files. They record data until they reach a user-defined maximum size. Once they reach this size, they will overwrite existing log data starting from the beginning of the file. Use this format if you want to ensure that the most current performance data is always accessible and you do not need to keep older sampled data once the log file maximum is reached.

Structured Query Language (SQL) database format allows all data to be quickly and easily imported into a SQL database. This format is ideal for archiving summarized data, especially if you are responsible for tracking performance on multiple computers. Chapter 4, "Performance Monitoring Procedures," illustrates the use of a SQL database for longer term archival and retrieval of counter log data to support capacity planning.

Scheduling Collection Periods

You can start and stop counter logs manually or schedule them to run automatically. You can define an automatic start and end time for a counter logging session, or specify that the logging session gather performance data continuously. At the end of a designated logging period, you can start a new logging session and run a command to process the counter log file that has just been completed.

File Management

You can generate unique names for counter log files that you create. You can choose to number log files sequentially or append a time or date stamp that identifies when the counter log file was created. You can also set a limit on the size that any log file can grow to. Using the circular binary file format, you can ensure that your counter log file will never consume more than the designated amount of disk space.

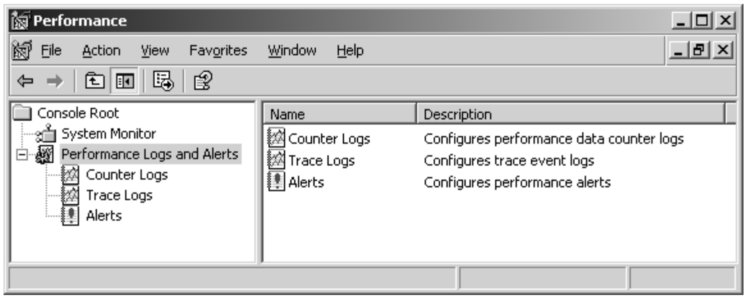
Working with Counter Logs

You can find Performance Logs and Alerts in the Performance Monitor console and the Computer Management console. The following procedure describes how to start them.

To start Performance Logs and Alerts from the Performance console

1. Click Start, point to Run, and then type **Perfmon**.
2. Press the ENTER key.
3. Double-click Performance Logs and Alerts to display the available tools.

Figure 2-12 shows the Performance console tree.



Name	Description
Counter Logs	Configures performance data counter logs
Trace Logs	Configures trace event logs
Alerts	Configures performance alerts

Figure 2-12 Performance Logs and Alerts console tree

After you load the Performance Logs and Alerts console, you will need to configure the Counter logs.

To configure counter logs

1. Click the Counter Logs entry to select it.

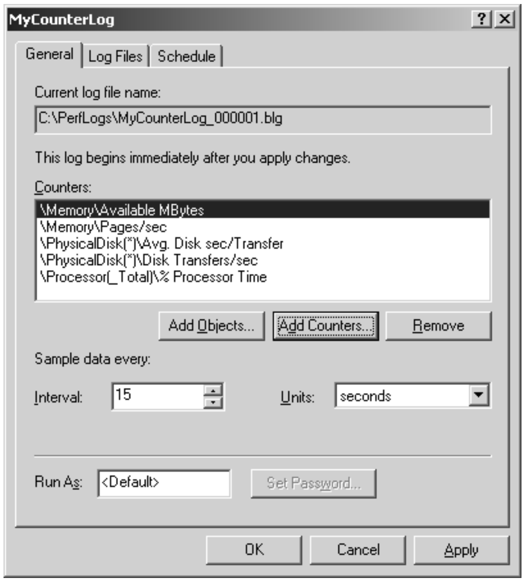
   Previously defined logs and alerts appear in the appropriate node of the details pane. A sample settings file for a counter log named System Overview in <system drive>:\perflogs\system_overview.blg is included with Windows Server 2003. These counter log settings allow you to monitor basic counters from the memory, disk, and processor objects.

2. Right-click the details pane to create a new log. You can also use settings from an existing HTML file as a template.

> **Note** To run the Performance Logs and Alerts service, you must be a member of the Performance Log Users or Administrators security groups. These groups have special security access to a subkey in the registry to create or modify a log configuration.

3. In the New Log Settings box, type the name of your counter log session and click OK.

   Figure 2-13 shows the General tab for the Properties of a new counter log after you have entered the counter log name.



Current log file name:
C:\PerfLogs\MyCounterLog_000001.blg

This log begins immediately after you apply changes.

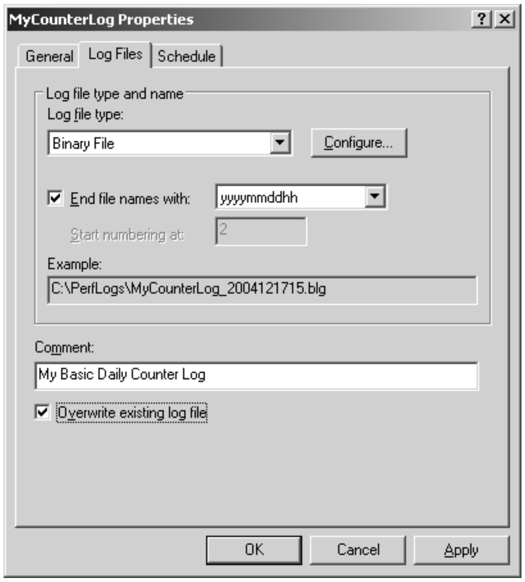
Counters:
\Memory\Available MBytes
\Memory\Pages/sec
\PhysicalDisk(*)\Avg. Disk sec/Transfer
\PhysicalDisk(*)\Disk Transfers/sec
\Processor(_Total)\% Processor Time

Sample data every:
Interval: 15
Units: seconds

Run As: <Default>

Figure 2-13 General tab for a counter log

4. To configure a counter log, click Add Objects or Add Counters to specify objects, counters, and instances. You also set the data collection interval on the General tab. Use the Add Objects button to log all counters from all instances of a performance object that you have selected. Use the Add Counters button to see the same familiar dialog box that is used to add counters to an interactive System Monitor data collection session.
5. Click the Log Files tab, shown in Figure 2-14, to set the file type, the file naming convention, and other file management options.

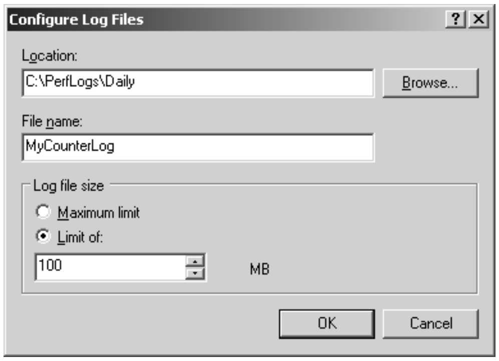


Log file type and name
Log file type: Binary File
[✓] End file names with: yyyymmddhh
Start numbering at: 2
Example: C:\PerfLogs\MyCounterLog_2004121715.blg
Comment: My Basic Daily Counter Log
[✓] Overwrite existing log file

Figure 2-14 Configuring counter log file properties

The Log Files tab is used to select the file type and automatic file naming options. You can generate unique log file names that are numbered consecutively as an option, or you can add a date and timestamp to the file name automatically. Or you can choose to write all performance data to same log file where you specify that current performance data is used to overwrite any older data in the file. After you set the appropriate option, the Log Files tab displays an example of the automatic file names that will be generated for you.

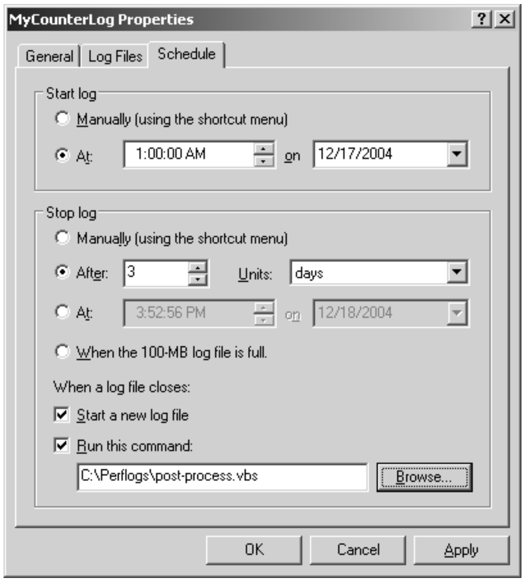
6. Click the Configure button to set the file name, location, and a file size limit in the Configure Log Files dialog box, as shown in Figure 2-15. Click OK to close the dialog box.



Location: C:\PerfLogs\Daily
File name: MyCounterLog
Log file size
○ Maximum limit
● Limit of: 100 MB

Figure 2-15 Dialog box for configuring log file size limits

7. Click the Schedule tab (Figure 2-16) to choose manual or automatic startup options. You can then set the time you want the logging session to end using an explicit end time; a duration value in seconds, minutes, hours, or days; or when the log file reaches its designated size limit.



Start log
- Manually (using the shortcut menu)
- [✓ At:] 1:00:00 AM on 12/17/2004

Stop log
- Manually (using the shortcut menu)
- [✓ After:] 3 Units: days
- At: 3:52:56 PM on 12/18/2004
- When the 100-MB log file is full.

When a log file closes:
- [✓] Start a new log file
- [✓] Run this command: C:\Perflogs\post-process.vbs

Figure 2-16 Setting startup options on the Schedule tab

The Counter log properties that allow you to establish automated performance monitoring procedures are summarized in the Table 2-15.

Table 2-15 Summary of Counter Log Properties

Tab	Settings to Configure	Notes
General	Add objects, or add counters	You can collect performance data from the local computer and from remote computers.
	Sample interval	Defaults to once every 15 seconds.
	Account and password	You can use Run As to provide the logon account and password for data collection on remote computers.
Log Files	File type	Counter logs can be defined as comma-delimited or tab-delimited text files, as binary or binary circular files, or as SQL database files. For database files, use Configure to enter a repository name and data set size. For all other files, use Configure to enter location, file name, and log file size.
	Automatic file naming	You can choose to add unique file sequence numbers to the file name or append a time and date stamp to identify it.

Table 2-15 **Summary of Counter Log Properties**

Tab	Settings to Configure	Notes
Schedule	Manual or automated start and stop methods and schedule	You can specify that the log stop collecting data when the log file is full.
	Automated start and stop times	Start and stop by time of day or specify the log start time and duration.
	Stop when the file is full	Automatically stop data collection when the file reaches its maximum size.
	Processing when the log file closes	For continuous data collection, start a new log file when the log file closes. You can also initiate automatic log file processing by running a designated command when the log file closes.

Note Sequential counter log files can grow larger than the maximum file size specified. This occurs for counter logs because the log service waits until after the last data sample was gathered and written to the file before checking the size of the log file. At this point, the file size might already exceed the defined limit.

Analyzing Counter Logs

Once you have created a counter log, you can use the System Monitor to analyze the performance data it contains. If you need to manipulate the measurement data further, run the Relog tool to create a text format file that you can read using a spreadsheet application like Excel, as discussed further in the section entitled "Managing Performance Logs."

To analyze counter logs using the System Monitor

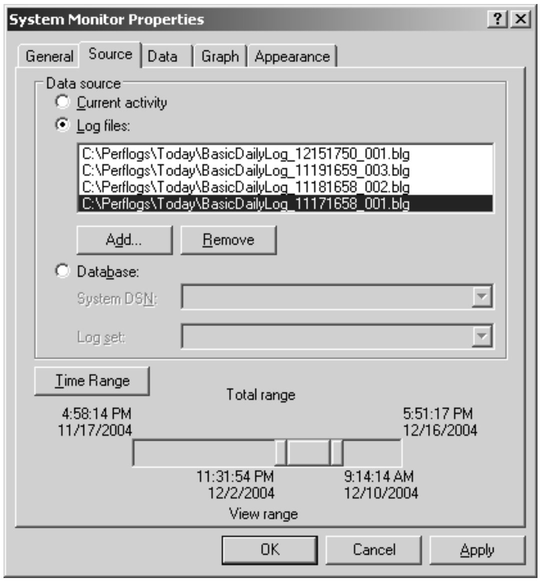
1. Open System Monitor and click the View Log Data button.
2. On the Source tab (shown in Figure 2-17), click Log Files as the data source and use the Add button to open the log file you want to analyze. You can specify one or more log files or a SQL database.
3. Click the Time Range button to set the start and end time of the interval you want to analyze using the slider control.

   Remember that a System Monitor Chart View can display only 100 data points at a time. When you are reporting on data from a log file, the Chart View automatically summarizes sequences longer than 100 measurement intervals to fit the display. If the counter contains 600 measurement samples, for example, the Chart View line graphs will show the average of every six data points. The Duration field in the Value bar will display the duration of the time interval you have selected to view. You can also use Report View and Histogram View to display numeric averages, minimum values, and maximum values for all the counter values stored in the log file.

> **Note** When you are working with a log file, you can only add counters that are available in the counter file to a chart or report.

4. Right-click the display to relog a binary data file to a text file format using the Save Data As function. Relogging binary data to text format allows you to analyze the counter log data using a spreadsheet application like Excel. To convert a binary data file into a SQL database format, you must run the Relog command-line tool instead.

> **Tip** You also summarize a counter log when you relog it by setting the summarization interval. For example, if the original counter log contains measurement data gathered every 5 minutes, if you reduce the log file size by writing data only once every 12 intervals, you will create a counter log that contains hourly data.


System Monitor Properties

Data source: [✓ Log files:]

```
C:\Perflogs\Today\BasicDailyLog_12151750_001.blg
C:\Perflogs\Today\BasicDailyLog_11191659_003.blg
C:\Perflogs\Today\BasicDailyLog_11181658_002.blg
C:\Perflogs\Today\BasicDailyLog_11171658_001.blg
```

Total range: 4:58:14 PM 11/17/2004 – 5:51:17 PM 12/16/2004

View range: 11:31:54 PM 12/2/2004 – 9:14:14 AM 12/10/2004

Figure 2-17 Changing the time range to view a subset of the counter log data

Tips for Working with Performance Logs and Alerts

Windows Server 2003 Help for Performance Logs and Alerts describes performing the most common tasks with logs and alerts, including how to create and configure a counter log.

Table 2-16 gives tips that you can use when working with the Performance Logs and Alerts snap-in.

Table 2-16 Tips for Working with Performance Logs and Alerts

Task	Do This
Export log data to a spreadsheet for reporting purposes	Exporting log data to a spreadsheet program such as Excel offers easy sorting and filtering of data. For best results, Relog binary counter log files to text file format, either CSV or TSV.
Record intermittent data to a log	Not all counter log file formats can accommodate data that is not present at the start of a logging session. For example, if you want to record data for a process that begins after you start the log, select one of the binary (.blg) formats on the Log Files tab.
Limit log file size to avoid disk-space problems	If you choose automated counter logging with no scheduled stop time, the file can grow to the maximum size allowed based on available space. When you set this option, consider your available disk space and any disk quotas that are in place. Change the file path from the default (the Perflogs folder on the local computer) to a location with adequate space if appropriate. During a logging session, if the Counter Logs service cannot update the file because of lack of disk space, an event is written to the Application Event Log showing an error status of "Error disk full."
Name files for easy identification	Use File Name (in the Configure Log Files box) and End File Names With (on the Log Files tab) to make it easy to find specific log files. For example, you can set up periodic logging, such as a log for every day of the week. Then you can develop different naming schemes with the base name being the computer where the log was run, or the type of data being logged, followed by the date as the suffix. For example, a naming scheme that generates a file named ServerRed1_050212.blg was created on a computer named ServerRed1 on May 2nd at noon, assuming the End File Name With entry was set at mmddhh.

Creating Performance Logs Using Logman

Suppose you want to monitor the amount of network traffic generated by your backup program, an application that runs at 3:00 each morning? The Counter Log facility in Performance Logs and Alerts allows you to record performance data to a log file, thus providing for automated and unattended performance monitoring. But even though the Performance Logs and Alerts snap-in is an excellent tool for carrying out these tasks, it does have some limitations:

- Each counter log setting must be created using the graphical user interface. Although this makes it possible for novice users to create performance monitors, experienced users might find the process slower and less efficient than creating performance monitors from the command line.
- Performance log settings created with the snap-in cannot be accessed from batch files or scripts.

The Logman command-line tool helps overcome these limitations. In addition, Logman is designed to work with other command-line tools like Relog and Typeperf to allow you to build reliable automated performance monitoring procedures. This section documents the use of Logman to create and manage counter log files. For more information about Logman, in Help and Support Center for Microsoft Windows Server 2003, click **Tools**, and then click **Command-Line Reference A–Z**. The Logman facilities related to creating trace logs are discussed in "Event Tracing for Windows" later in this chapter.

Log Manager Overview

Log Manager (Logman.exe) is a command-line tool that complements the Performance Logs and Alerts snap-in. Log Manager replicates the functionality of Performance Logs and Alerts in a simple-to-use command-line tool. Among other things, Log Manager enables you to:

- Quickly create performance log settings from the command line.
- Create and use customized settings files that allow you to copy the monitoring configuration and reuse it on other computers.
- Call performance logging within batch files or scripts. These batch files or scripts can then be copied and used on other computers in your organization.
- Simultaneously collect data from multiple computers.

The data that Log Manager collects is recorded in a performance counter log file using the format you specify. You can use the System Monitor to view and analyze any counter log file that Log Manager creates.

Command Syntax

Logman operates in one of two modes. In Interactive mode, you can run Logman from a command-line prompt and interact with the logging session. For example, you can control the start and stop of a logging session interactively. In Background mode, Logman creates Counter log configurations that are scheduled and processed by the same Performance Logs and Alerts service that is used with the Performance Monitor console. For more information about Performance Logs and Alerts, see "Performance Logs and Alerts" in this chapter.

Table 2-17 summarizes the six basic Logman subcommands.

Table 2-17 Logman Subcommands

Subcommand	Function
Create counter *CollectionName*	Creates collection queries for counter data collection sessions.
Update *CollectionName*	Updates an existing collection query to modify the collection parameters.
Delete *CollectionName*	Deletes an existing collection query.
Query *CollectionName*	Lists the collection queries that are defined and their status. Use query *CollectionName* to display the properties of a specific collection. To display the properties on remote computers, use the *-s RemoteComputer* option on the command line.
Start *CollectionName*	Starts a logging session manually.
Stop *CollectionName*	Stops a logging session manually.

Collection queries created using Log Manager contain properties settings identical to the Counter Logs created using the Performance Logs and Alerts snap-in. If you open the Performance Logs and Alerts snap-in, you will see any collection queries you created previously using Log Manager. Likewise, if you use the *-query* command-line parameter in Log Manager to view a list of collection queries on a computer, you will also see any counter logs created using the Performance Logs and Alerts snap-in.

Table 2-18 summarizes the Logman command-line parameters that are used to set the properties of a logging session.

Table 2-18 Logman Counter Logging Session Parameters

Parameter	Syntax	Function	Notes
Settings file	*-config FileName*	Use the logging parameters defined in this setting file.	
Computer	*-s ComputerName*	Specify the computer you want to gather the performance counters from.	If no computer name is provided, the local computer is assumed.
Counters	*-c {Path [Path ...]*	Specify the counters that you want to gather.	Required. Use *-cf FileName* to use counter settings from an existing log file.
Sample interval	*-si [[HH:]MM:]SS*	Specify the interval between data collection samples.	Defaults to 15 seconds.
Output file name	*-o {Path | DSN!CounterLog}*	Specify the output file name. If the file does not exist, Logman will create it.	Required. Use *-v* option to generate unique file names.

Table 2-18 Logman Counter Logging Session Parameters

Parameter	Syntax	Function	Notes
File versioning	*-v {NNNNNN | MMDDHHMM}*	Generate unique file names, either by numbering them consecutively or by adding a time and date stamp to the file name.	
Log file format	*-f bin | bincirc | csv | tsv | SQL}*	Choose the format of the output counter log file.	Defaults to binary format.
File size limit	*-max Value*	Specify the maximum log file or database size in MB.	Logging ends when the file size limit is reached.
Create New log file at session end	*-cnf [[HH:]MM:]SS*	Create a new log file when the file size limit or logging duration is exceeded.	Requires that *-v* versioning be specified to generate unique file names.
Run command at session end	*-rc FileName*	Run this command at session end.	Use in conjunction with *-cnf* and *-v* options.
Append	*-a*	Append the output from this logging session to an existing file.	
Begin logging	*-b M/D/YYYY H:MM:SS [{AM | PM}]*	Begin a logging session automatically at the designated date and time.	
End logging	*-e M/D/YYYY H:MM:SS [{AM | PM}]*	End a logging session automatically at the designated date and time.	Or use *-rf* to specify the duration of a logging session.
Log duration	*-rf [[HH:]MM:]SS*	End a logging session after this amount of elapsed time.	Or use *-e* to specify a log end date and time.
Repeat	*-r*	Repeats the collection every day at the same time. The time period is based on either the *-b* and *-rf* options, or the *-b* and *-e* options.	Use in conjunction with *-cnf*, *-v*, and *-rc* options.
Start and stop data collection.	*-m [start] [stop]*	Start and stop an interactive logging session manually.	
User name and password.	*-u UserName Password*	Specify user name and password for remote computer access.	The User account must be a member of the Performance Log Users Group. Specify * to be prompted for the password at the command line.

Creating Log Manager Collection Queries

Before you can use Log Manager to start logging performance data, you must create a collection query, which is a set of instructions that specifies which computer to monitor, which performance counters to gather, how often to gather those counters, and other performance data collection parameters.

Collection queries are created using the *create counter* parameter, followed by the name to give to the collection query, and then a list of performance counters to be monitored. For example, the following command creates a collection query called *web_server_log*:

```
logman create counter web_server_log
```

Although the preceding command creates *web_server_log*, it does not specify any performance counters. To create a collection query that actually collects data, you need to use the *-c* parameter to indicate the performance counters to be sampled. For example, the following command creates a collection query called *web_server_log* that can be used to monitor and log available bytes of memory:

```
logman create counter web_server_log -c "\Memory\Available Bytes"
```

If you want to monitor multiple performance counters with a single collection query, list each of these counters after the *-c* parameter. The following example creates a collection query that measures three performance counters:

```
logman create counter web_server_log -c "\Memory\Available Bytes" "\Memory\Pages/sec"
"\Memory\Cache Bytes"
```

Using a Settings File with Log Manager

When creating a collection query that monitors a large number of performance counters, placing those counters in a counter settings file is easier than typing them as part of the command-line string. For example, to monitor disk drive performance, you could use a command string similar to this:

```
logman -create counter web_server_log -c "\PhysicalDisk\Avg. Disk Bytes/Read"
"\PhysicalDisk\Avg. % Disk Read Time" "\PhysicalDisk\Avg. Split IOs/sec"
"\PhysicalDisk\Avg. Current Disk Queue Length" "\Avg. PhysicalDisk\Avg. Disk Bytes/
Read"
```

Alternatively, you could list the performance counters in a text file (one counter per line) and then reference this file when creating the collection query. A sample settings file is shown in Listing 2-1.

Listing 2-1 Log Manager Settings File

```
"\PhysicalDisk\Avg. Disk Bytes/Read"
"\PhysicalDisk\Avg. % Disk Read Time"
"\PhysicalDisk\Avg. Split IOs/sec"
"\PhysicalDisk\Avg. Current Disk Queue Length"
"\Avg. PhysicalDisk\Avg. Disk Bytes/Read"
```

To create a collection query that reads a settings file, use the *-cf* parameter followed by the path to the file. For example, if your settings file is stored in C:\Scripts\Counters.txt, use the following command string:

```
logman create counter web_server_log –cf c:\scripts\counters.txt
```

Settings files enable you to create query collections that can be used consistently on many computers. For example, if you plan to monitor disk drive performance on 20 different computers, you can use a common settings file to create and distribute consistent collection queries

> **Tip** Use the query *(-q)* option of the Typeperf tool to list counter paths and save them in a text file, which you can then edit and use as a Logman settings file to minimize typing errors.

For more information about how to specify the counter path correctly, see "Performance Counter Path" earlier in this chapter.

Keep in mind that the Logman tool was designed for scripting. One way to create the same query collections on multiple computers is to create a batch file with the command string. That batch file can then be copied and run on multiple computers, resulting in the exact same query collection being created on each computer.

Monitoring Remote Computers Using Log Manager

Log Manager can be used to monitor remote computers, and, if you choose, to consolidate the performance statistics from each of these computers into the same log file. To monitor a remote computer, add the *-s* parameter followed by the computer name.

For example, the following collection query monitors available bytes of memory on the remote computer DatabaseServer:

```
logman create counter web_server_log –c "Memory\Available bytes" –s DatabaseServer
```

To carry out performance monitoring on remote computers, you must have the necessary permissions. If the account from which you are working does not have the required permissions, you can specify a user name and password for an account that does by using the *-u* (user name and password) parameters. For example, this command creates a collection query that runs under the user name *jones*, with the password *mypassword*:

```
logman create counter file_server_log –cf c:\Windows\logs\counters.txt –s FileServer
–u jones mypassword
```

To add monitoring for another remote computer to the same logging session, use the *update* parameter:

```
logman update counter file_server_log -cf c:\Windows\logs\counters.txt -s WebServer
-u jones mypassword
```

> **Note** If you specify an asterisk as the password (*-u jones **), you are prompted for the password when you start the collection query. This way, the password is not saved in the collection query, nor is it echoed to the screen.

Configuring the Log Manager Output File

Log Manager does not display performance data on-screen. Instead, all performance measurements are recorded in an output log file. Whenever you create a collection query, you must include the *-o* parameter followed by the path for the log file. You do not need to include a file name extension; Log Manager will append the appropriate file name extension based on the format of the output file.

By default, Log Manager will save output files to the directory where you issued the command. For example, this command saves the output file in C:\My Documents\Web_server_log.blg, assuming you issued the command from the folder C:\My Documents:

```
logman -create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
```

To save your files somewhere other than in the default folder, include the full path. For example, to have performance data logged to the file C:\Scripts\Web_server_log.blg, use this command:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o
c:\scripts\web_server_log
```

Or, use a UNC path to save the file to a remote computer:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o
\\RemoteComputer\scripts\web_server_log
```

If the file Web_server_log does not exist, Log Manager creates it. If the file does exist, Log Manager returns an error saying that the file already exists. To overwrite the file, use the *-y* option. If you want Log Manager to append information to an existing log file (that is, to add new data without erasing existing data), you must use the *-a* parameter:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log -a
```

As an alternative to logging data to a text file, you can record data in a SQL database, provided you have used the ODBC Data Source Administrator to create a system Data Source Name (DSN) for that database. If a DSN exists, you can log performance data to a SQL database using the format *DSN!Table_name*, where *DSN* represents the Data Source Name, and *Table_name* represents the name of a table within that database. (If the table does not exist, Log Manager will create it.) For example, the following command logs performance data into a table named DailyLog in the PerformanceMonitoring database:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o
PerformanceMonitoring!DailyLog -f SQL
```

Note The *-f* (format) parameter is required when saving data to a SQL database. The DSN created *must* be a system DSN and not a user DSN. Logman will fail if a user DSN is used. Additionally, the DSN must point to a database that already exists. Your SQL database administrator can create a SQL database for you to use.

Adding Versioning Information to Log File Names with Log Manager

Versioning allows you to automatically generate unique counter log file names. Log Manager supports two different versioning methods:

- **Numeric** With numeric versioning, file names are appended with an incremental 6-digit numeric suffix. For example, the first log file created by a collection query might be called Web_server_log_000001.blg. The second file would then be called Web_server._log_000002.blg.
- **Date/Time** With date/time versioning, file names are appended with the current month, day, and time (based on a 24-hour clock), using the *mmddhhmm* format (month, date, hour, minute). For example, a file saved at 8:30 A.M. on January 23 might be named Web_server.log_01230830.

To add versioning information to your file names, use the *-v* parameter, followed by the appropriate argument. Logman uses *nnnnnn* as the argument for adding numeric versioning, and *mmddhhmm* as the argument for date/time versioning. These are the only two valid arguments.

Note Performance Logs and Alerts supports some additional date formats, including *yyyymmdd*, which appends the year, month, and day to the end of the file name. These additional formats cannot be set using Log Manager; however, you can create a collection query using Log Manager, and then use the Performance Logs and Alerts snap-in to modify the versioning format.

For example, this command configures versioning using the numeric format:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -v nnnnnn
```

This command configures versioning using the date/time format:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -v mmddhhmm
```

Formatting the Log Manager Output File

Log Manager allows you to specify the data format for your output file. To designate the file format for a Log Manager output file, use the *-f* parameter followed by the format type. For example, this command sets the file format to circular binary:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-f BINCIRC
```

Valid file formats are described in Table 2-19. Logman uses the same file formats as the Counter log facility in Performance Logs and Alerts. (See Table 2-14 for more details.) Binary file format is the most efficient and concise way to store counter log data. Because the Relog tool can be used to convert binary files into any other format, there should be little reason to utilize anything but binary format files.

Table 2-19 Log Manager File Formats

File Format	Description
BIN	Binary format.
BINCIRC	Circular binary format.
CSV	Comma-separated values.
TSV	Tab-separated values.
SQL	SQL database format.

Specifying a Maximum Size for the Log Manager Output File

Although Log Manager output files are relatively compact, they can still potentially grow quite large. For example, a collection query that monitors two performance counters every 15 seconds creates, after 24 hours, a file that is about one megabyte in size. This might be a reasonable size for a single computer. However, if you have multiple computers logging performance data to the same file, that file might grow so large that it would become very difficult to analyze the data.

To keep log files to a manageable number of bytes, you can set a maximum file size. To do this, add the *-max* parameter followed by the maximum file size in megabytes. (If you are logging data to a SQL database, the number following the *-max* parameter represents the maximum number of records that can be added to the table.) For example, to limit the file Web_server_log.blg to 5 megabytes, use this command:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -f CSV -max 5
```

File size limit checking If you specify a file size limit using the *-max* value, the file system where you plan to create the file is evaluated before the counter log session starts to see whether there is an adequate amount of space to run to completion. This file size limit is performed only once when the log file you are creating is first stored on the system drive. If the system drive has insufficient space, the logging session will fail, with a smlogsvc error message reported in the event log similar to the following:

```
An error occurred while trying to update the log file with the current data
for the <session name> log session.  This log session will be stopped.
The Pdh error returned is: Log file space is insufficient to support this operation.
```

An error return code of 0xC0000188 in this error message indicates an out-of-space condition that prevented the logging session from being started.

Note that additional configuration information is written to the front of every counter log file so that slightly more disk space than you have specified in the *-max* parameter is actually required to start a trace session.

Creating new log files automatically When a log file reaches its maximum size, the default behavior for Log Manager is to stop collecting data for that collection query. Alternatively, you can have Log Manager automatically start recording data to a new output file should a log file reach its maximum size. You do this with the *-cnf* (create new file) parameter. When you specify the create new file parameter, you must use the *-v* versioning option to generate unique file names. For example, the following command instructs Log Manager to create a new log file each time the maximum size is reached, and to use the numeric versioning method for naming files:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -f CSV -v nnnnnn -cnf
```

You can also force Log Manager to start a new log file after a specified amount of time with *-cnf hh:mm:ss*, where *hh* is hours, *mm* is minutes, and *ss* is seconds. For example, the following command causes Log Manager to start a new log file every 4 hours:

```
logman create web_server_log -c "\Memory\Available Bytes" -o web_server_log -a -f
BINCIRC -v nnnnnn -cnf 04:00:00
```

Configuring the Log Manager Sampling Interval

The default Log Manager sampling interval is 15 seconds, similar to the Performance Logs and Alerts facility. The Log Manager sampling interval can be changed using the *-si* parameter followed by the interval duration. To configure the sampling interval, use *-si hh:mm:ss*, where *hh* is hours, *mm* is minutes, and *ss* is seconds. Partial parameters for the sample duration interval can be specified. For example, this command sets the sampling interval to 45 seconds:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -f CSV -si 45
```

To sample every 1 minute 45 seconds, the following command can be used:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -f CSV -si 1:45
```

To sample every 1 hour 30 minutes 45 seconds, the following command can be used:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -f CSV -si 1:30:45
```

Scheduling Log Manager Data Collection

By default, Log Manager begins collecting performance data as soon as you start a collection query, and it continues to collect that performance data until you stop it. There might be times, however, when you want to schedule data collection for a specific period of time. For example, suppose you have a new backup program that automatically runs from 3:00 A.M. to 4:00 A.M. each morning. To know what sort of stress this backup program is placing on the system, you could come in at 3:00 A.M., start Log Manager, and then, an hour later, stop Log Manager. Alternatively, you can schedule Log Manager to automatically begin data collection at 3:00 A.M. and automatically to stop monitoring an hour later.

To schedule data collection with Log Manager, you must specify both a beginning date and time by using the *-b* parameter, and an ending time by using the *-e* parameter. Both of these parameters require time data to be formatted as *hh:mm:ssAMPM*, where *hh* is hours, *mm* is minutes, *ss* is seconds, and *AMPM* designates either morning or afternoon/evening. For example, this command monitors available bytes of memory between 3:00 A.M. and 4:00 A.M. on August 1, 2003:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -f CSV -b 08/01/2003 03:00:00AM -e 08/01/2003 04:00:00AM
```

The preceding command causes data collection to begin at 3:00 A.M. and to end—permanently—at 4:00 A.M. If you want data collection to take place every day between 3:00 A.M. and 4:00 A.M., add the repeat (*-r*) parameter:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -f CSV -b 08/01/2001 03:00:00AM -e 08/01/2001 04:00:00AM -r
```

You can also schedule data collection to take place during only a specified set of dates. For example, to collect data only for September 1, 2003 through September 5, 2003, add the dates to the *-b* and *-e* parameters using the *mm-dd-yyyy* (month-day-year) format:

```
logman create counter web_server_log -c "\Memory\Available Bytes" -o web_server_log
-a -f CSV -b 09-01-2003 03:00:00AM -e 09-05-2003 04:00:00AM
```

Starting, Stopping, Updating, and Deleting Data Collections Using Log Manager

Simply creating a collection query without specifying a scheduled begin time or end time does not cause Log Manager to start monitoring performance data. Instead, you must explicitly start data collection by using the start parameter. For example, to begin collecting data using the *web_server_log* collection query, type this command:

```
logman start web_server_log
```

Once data collection has begun, the collection query will run either until it reaches the end time (if you have included the *-e* parameter) or until you stop it by using the *stop* parameter. For example, this command stops the collection query *web_server_log*:

```
logman stop web_server_log
```

To help you keep track of all your collections, the *query* parameter shows you a list of all the collection queries stored on a computer, as well as their current status (running or stopped). To view the list of collection queries on the local computer, use this:

```
logman query
```

To view the list of collection queries on a remote computer, add the -*s* parameter and the name of the computer (in this case, DatabaseServer):

```
logman query -s DatabaseServer
```

> **Note** The *-query* parameter will tell you what collection queries are running on a computer; however, it will not tell you what collection queries are being run against a computer. For example, you might have 10 collection queries running on your computer, but each could be monitoring performance on a different remote computer. You will not know this, however, unless you know the specifics of each collection query.

To delete a collection query, use the *delete* parameter followed by the query name:

```
logman delete web_server_log
```

An existing collection query can be updated with new information using the *update* parameter followed by the collection name and the parameters you want to update. For example to update a collection called *web_server_log* with a new sample interval of 60 seconds, the following command can be used:

```
Logman update web_server_log -si 60
```

Any parameter can be updated using *update*. Note that for an update to take effect, the collection must be stopped and restarted.

Using Windows Script Host to Manage Log Manager Data Collection

Log Manager's built-in scheduling options allow you to schedule data collection at specific times during the day (for example, between 3:00 A.M. and 4:00 A.M.). By specifying a different sampling interval, you can also have Log Manager collect data at regular intervals. (For example, setting the sampling interval to 1:00:00 will cause Log Manager to take a single sample every hour.)

What you cannot do with Log Manager is schedule data collection for more irregular time periods. For example, suppose you want to collect performance data for 5 minutes at the beginning of every hour. There is no way to do this using Log Manager alone.

Note You could, however, do this using Typeperf. To do this, create a batch file that configures Typeperf to take only 5 minutes' worth of samples. Then schedule the batch file to run once every hour using the Task Scheduler.

However, because Logman is scriptable, you can combine Windows Script Host (WSH) and Log Manager to schedule data collection using more irregular intervals. The script shown in Listing 2-2 starts Log Manager (using the *-start* parameter) and then pauses for 5 minutes, using the WSH *Sleep* method. While the WSH script is paused, Log Manager collects data. After 5 minutes, the script resumes and issues the *stop* command to stop data collection. The script then pauses for 55 minutes (3300000 milliseconds) before looping around and starting again.

Listing 2-2 Running Log Manager Within a WSH Script

```
set WshShell = WScript.CreateObject(“WScript.Shell”)
Do
    WshShell.Run "%COMPSEC% /c logman -start web_server_log"
    WScript.Sleep 300000
    WshShell.Run "%COMPSEC% /c logman -stop web_server_log"
    WScript.Sleep 3300000
Loop
```

Listing 2-2 is designed to run indefinitely. To run it a finite number of times, use a *for-next* loop. For example, the script shown in Listing 2-3 causes the script to run 24 times (once an hour for an entire day) and stop.

Listing 2-3 Using a WSH Script to Run Log Manager Once an Hour for 24 Hours

```
Set WshShell = WScript.CreateObject("WScript.Shell")
For i = 1 to 24
    WshShell.Run "%COMPSEC% /c logman -start web_server_log"
    WScript.Sleep 300000
    WshShell.Run "%COMPSEC% /c logman -stop web_server_log"
    WScript.Sleep 3300000
Next i
Wscript.Quit
```

Managing Performance Logs

Both the Performance Logs and Alerts facility in the Performance Monitor console and the Log Manager command-line interface allow you considerable flexibility in gathering performance statistics. The Relog tool (Relog.exe) is a command-line tool that allows you manage the counter logs that you create on a regular basis. Using Relog, you can perform the following tasks:

- Combine multiple counter logs into a single log file. You can list the file names of all the counter logs that you want Relog to process separately, or you can use wildcards (for example, *.blg) to identify them. The logs you combine can contain counters from a single computer or from multiple computers.
- Create summarized output files from an input file or files.
- Edit the contents of a counter log by allowing you to drop counters by name or drop all counters not collected during a designated time interval.
- Convert counter data from one file format to another.

Note Log Manager can record performance data on multiple computers and save that data to the same log file. However, this can result in a considerable amount of unwanted network traffic. Relog allows you to monitor performance locally, and then retrieve the data as needed. By putting the Relog commands in a batch file, data retrieval can be scheduled to take place at times when network traffic is relatively low.

More Info For more information about Relog, in Help and Support Center for Microsoft Windows Server 2003, click Tools, and then click Command-Line Reference A–Z.

Using the Relog Tool

Relog requires two parameters: the path for the existing (input) log file, and the path for the new (output) log file (indicated by the *-o* parameter). For example, this command will extract the performance records from the file C:\Perflogs\Oldlog.blg and copy them into the file C:\Perflogs\Newlog.blg:

```
relog c:\Perflogs\oldlog.blg –o c:\Perflogs\newlog.blg
```

Relog gathers data from one or more performance logs and combines that data into a single output file. You can specify a single input file or a string of input files, as in the following example:

```
relog c:\Perflogs\oldlog1.blg c:\Perflogs\oldlog2.blg
c:\Perflogs\oldlog3.blg
–o c:\Perflogs\newlog.blg
```

If the file Newlog.blg does not exist, Relog will create it. If the file Newlog.blg does exist, Relog will ask you if you want to overwrite it with the new set of records, and any previously saved data is lost.

Command Syntax

The Relog tool supports a set of run-time parameters to define editing, summarization, and conversion options, with syntax and function that is similar to Logman. These parameters are summarized in Table 2-20.

Table 2-20 Relog Tool Parameters for Editing, Summarizing, and Converting Counter Log Files

Parameter	Syntax	Function	Notes
Settings file	*-config FileName*	Use the logging parameters defined in this setting file.	Use *-i* in the configuration file as a placeholder for a list of input files that can be placed on the command line.
Counters	*-c {Path [Path ...]*	Specify the counters from the input file that you want to write to the output file. If no counters are specified, all counters from the input files are written.	Use *-cf FileName* to use counter settings from an existing log file.
Summarization interval	*-t n*	Write output every *n* intervals of the input counter logs.	Defaults to creating output every input interval.

Table 2-20 Relog Tool Parameters for Editing, Summarizing, and Converting Counter Log Files

Parameter	Syntax	Function	Notes
Output file name	*-o {Path | DSN!CounterLog}*	Specify the output file name. If the file does not exist, Relog will create it.	Required.
Log file format	*-f bin | bincirc | csv | tsv | SQL}*	Choose the format of the output counter log file.	Defaults to binary format.
Append	*-a*	Append the output from this logging session to an existing file.	For binary input and output files only.
Begin relogging	*-b M/D/YYYY H:MM:SS [{AM | PM}]*	Specify the start date and time of the output file.	Defaults to the earliest start time of the input files.
End relogging	*-e M/D/YYYY H:MM:SS [{AM | PM}]*	Specify the end date and time of the output file.	Defaults to the latest end time of the input files.

To append data to an existing file, add the *-a* parameter.

```
relog c:\scripts\oldlog.blg –o c:\scripts\newlog.blg -a
```

Adding *-a* to the preceding example causes Relog to add the records extracted from Oldlog.blg to any existing records in Newlog.blg. Note that only binary files can be appended.

Note To append to an existing text file, use Relog to convert the text file to binary, use Relog again to append data from another file, and, finally, use Relog one more time to create the resulting text file.

Merging Counter Logs Using Relog

Use the Relog tool to merge data from multiple performance logs into a single file. You do this by specifying the path and file names for multiple performance logs as part of the initial parameter. For example, this command gathers records from three separate log files, and appends all the data to the file Newlog.blg:

```
relog c:\scripts\log1.blg c:\scripts\log2.blg c:\scripts\log3.blg –o
c:\Windows\logs\newlog.blg -a
```

Note Use the *-a* parameter to specify that Relog appended output to any existing data in the output file when you merge data from multiple logs.

Counter logs stored on remote computers can also be merged; use the UNC path instead of the local path. For example, this command gathers records from three different computers (DatabaseServer, PrintServer, and FileServer), and appends that data to the file Historyfile.blg:

```
relog \\DatabaseServer\logs\log1.blg \\PrintServer\logs\log2.blg
\\FileServer\logs\log3.blg -o c:\Windows\logs\Historyfile.blg -a -f blg
```

The individual paths cannot exceed a total of 1,024 characters. If you are retrieving performance data from a large number of computers, it is conceivable that you could exceed the 1,024-character limitation. In that case, you will need to break a single Relog command into multiple instances of Relog.

Formatting the Relog Output File

Relog supports the same input and output file formats as Logman except that you cannot use the binary circular file format or create output files with size limits. If a new output file is being created, the new file will be created in binary format by default. Relog can append data only to binary format files. Relog can append data only to existing files when both the input and output files use binary format.

When creating a new output file, you can use the *-f* parameter to specify one of the data formats shown in Table 2-21.

Table 2-21 Relog.exe File Formats

File Format	Description
bin	Binary format. This is the default file format.
csv	Comma-separated values.
tsv	Tab-separated values.
SQL	SQL database format.

Filtering Log Files Using Relog

Relog has a filtering capability that can be used to extract performance data from the input counter logs based on the following criteria:

- A list of counters, as specified by their paths
- A specified date and time range

Only the counters that meet the filtering criteria you specified are written to the output file that Relog creates.

Filtering by counter Filtering by counter is based on a counter list, specified either on the command line or in a Settings file. Relog writes to the designated output file only those values for the counters specified on the Relog command line or in the settings file. For more information about how to specify the counter path correctly, see "Performance Counter Path" earlier in this chapter.

In the following example, Relog writes values for only the \Memory\Available Bytes counter to the output file:

```
relog c:\scripts\oldlog.txt -o c:\scripts\newlog.txt -f csv -c "\Memory\Available
Bytes"
```

To extract the data for more than one counter, include each counter path as part of the *-c* parameter:

```
relog c:\scripts\oldlog.txt -f csv -o c:\scripts\newlog.txt -c "\Memory\Available
Bytes" "\Memory\Pages/sec" "\Memory\Cache Bytes"
```

Note Relog does not do any performance monitoring itself; all it does is collect data from existing performance logs. If you specify a performance counter that does not appear in any of your input files, the counter will not appear in your output file either.

If your input log contains data from multiple computers, include the computer name as part of the counter path. For example, to extract available memory data for the computer DatabaseServer, use this command:

```
relog c:\scripts\oldlog.txt -o -f csv c:\scripts\newlog.txt -c
"\\DatabaseServer\Memory\Available Bytes"
```

Instead of typing a large number of performance counters as part of your command string, you can use a settings file to extract data from a log file. A settings file is a text file containing the counter paths of interest. For instance, the settings file shown in Listing 2-4 includes 10 different counter paths:

Listing 2-4 Relog.exe Settings File

```
"\Memory\Pages/sec"
"\Memory\Page Faults/sec"
"\Memory\Pages Input/sec"
"\Memory\Page Reads/sec"
"\Memory\Transition Faults/sec"
"\Memory\Pool Paged Bytes"
"\Memory\Pool Nonpaged Bytes"
"\Cache\Data Map Hits %"
"\Server\Pool Paged Bytes"
"\Server\Pool Nonpaged Bytes"
```

To filter the input files so that only these counter values are output, use the *-cf* parameter, followed by the path of the settings file:

```
relog c:\scripts\oldlog.txt -o c:\scripts\newlog.txt -a -cf
c:\Windows\logs\memory.txt
```

Filtering by date Relog provides the ability to extract a subset of performance records based on date and time. To do this, specify the beginning time (*-b* parameter) and ending time (*-e* parameter) as part of your command string. Both of these parameters express dates and times using the *mm-dd-yyyy hh:mm:ss* format, where *mm-dd-yyyy* represents month-day-year; *hh:mm:ss* represents hours:minutes:seconds; and time is expressed in 24-hour format.

For example, to extract performance records from 9:00 P.M. on September 1, 2003 through 3:00 A.M. on September 2, 2003, use this command:

```
relog c:\scripts\oldlog.txt" -f csv -o c:\scripts\newlog.txt" -b 09-01-2003 21:00:00
-e 09-02-2003 03:00:00
```

If you code the time only on the *-b* and *-e* parameters, the current date is assumed. For example, this command extracts performance records logged between 9:00 A.M. and 5:00 P.M. on the current date:

```
relog c:\scripts\oldlog.txt -o c:\scripts\newlog.txt -b 09:00:00 -e 17:00:00
```

If you choose, you can filter the input files by both counter value and date and time in a single Relog execution.

Summarizing Log Files Using Relog

Relog allows you to reduce the size of the output files you create by writing only one out of every *n* records, where *n* is a parameter you can specify using the *-t* option. This has the effect of summarizing interval and averaging counters. Interval counters are ones like \Processor\Interrupts/sec and \Process\% Processor Time that report an activity rate over the measurement interval. Average counters like \Physical Disk\Avg. Disk sec/transfer are ones that report an average value over the measurement.

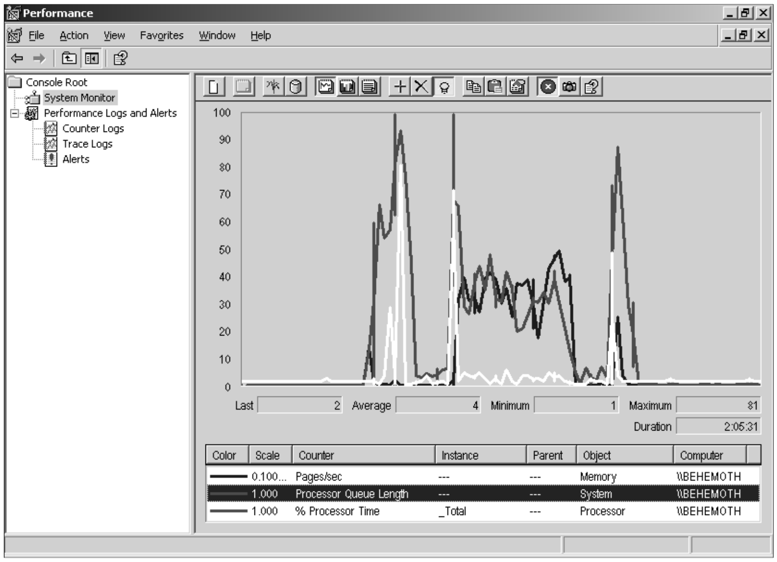
The *-t* parameter allows you to extract every *n*th record from the input counter logs and write them only to the output log file. For instance, *-t 40* selects every fortieth record; *-t 4* selects every fourth record. If the original input counter log was recorded with a sampling interval of once every 15 seconds, using Relog with the *-t 4* parameter results in an output file with data recorded at minute intervals. Interval counters like \Processor\Interrupts/sec and \Process\% Processor Time in the output file represent activity rates over 1-minute intervals. The same input file relogged using *-t 240* results in an output file with data recorded at 1-hour intervals. Interval counters like \Processor\Interrupts/sec and \Process\% Processor Time in the output file represent activity rates over 1-hour intervals.

Summarizing using Relog works by resampling the input counter log. The output file that results is not only more concise, it contains counter values that are summarized over longer intervals. You can expect some data smoothing to result from this sort of summarization, but nothing that would normally distort the underlying distribution and be difficult to interpret.

Not all the counters that you can collect can be summarized in this fashion, however. For an instantaneous counter like \Memory\Available Bytes or \System\Processor Queue Length, relogging the counter log merely drops sample observations. If you relog instantaneous counters to a large enough extent, you could wind up with an output file with so few observations that you do not have a large enough sample to interpret the measurements meaningfully. At that point, it is probably better to use a counter list with Relog to drop instantaneous counters from the output file entirely. Chapter 4, "Performance Monitoring Procedures," offers sample summarization scripts that illustrate this recommendation.

An example that uses Relog to summarize an input counter log from 1 minute to 4 minute intervals illustrates these key points.

Figure 2-18 shows a System Monitor Chart View that graphs three counters. The highlighted counter, the \System\Processor Queue Length, is an instantaneous counter, which Relog cannot summarize. The remaining counters shown on the chart are interval counters, which Relog can summarize. This counter log was created using a sample interval of 1 minute.



Last 2 Average 4 Minimum 1 Maximum 81
Duration 2:05:31

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	0.100...	Pages/sec	---	---	Memory	\\BEHEMOTH
▬	1.000	Processor Queue Length	---	---	System	\\BEHEMOTH
▬	1.000	% Processor Time	_Total	---	Processor	\\BEHEMOTH

Figure 2-18 Original counter log data before Relog summarization

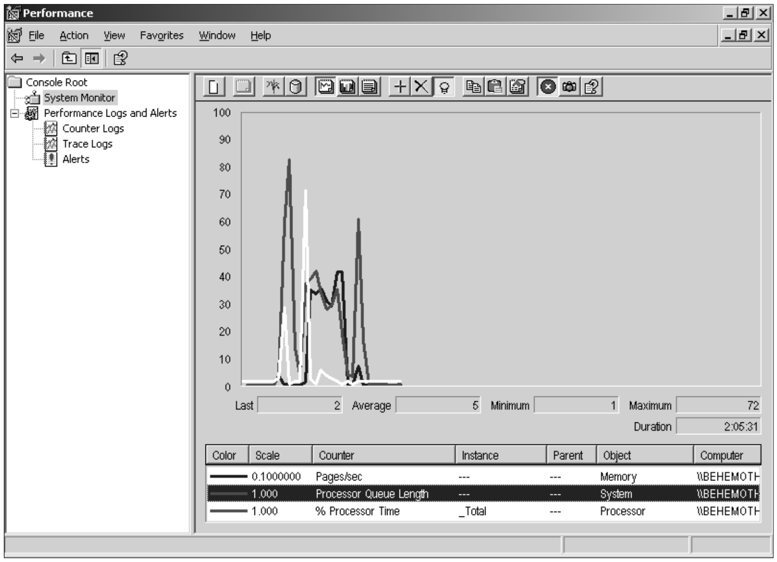
System Monitor reports that the counter log being charted covers a period with a duration of roughly 2 hours. Using the Log File Time Span feature, the Chart is zoomed into this period of unusual activity. Because the Chart View can display only 100 points, the line graphs are drawn based on skipping over a few of the data values that will not fit on the graph. (It is as if the Chart View has a built-in Relog function for drawing line graphs.) The statistics shown in the Value bar are based on all the observations in the period of interest. For the measurement period, the Processor Queue Length, sampled once every minute, shows an average value of 4, a minimum value of 1, and a maximum value of 81.

The Relog command to summarize this file to 4-minute intervals is shown in Listing 2-5, along with the output the tool produces.

Listing 2-5 Relogging Using the *-t* Parameter to Create Summarized Counter Logs

```
C:\PerfLogs>relog BasicDailyLog_12161833_001.blg -o
relogged_BasicDailyLog_12161833_001.blg -t 4

Input
----------------
File(s):
     BasicDailyLog_12161833_001.blg (Binary)

Begin:    12/16/2004 18:33:52
End:      12/17/2004 17:15:26
Samples:  1363

Output
----------------
File:     relogged_BasicDailyLog_12161833_001.blg

Begin:    12/16/2004 18:33:52
End:      12/17/2004 17:15:26
Samples:  341

The command completed successfully.
```

Relog reports that it found 1363 total sample collection intervals in the original input file, somewhat less than an entire day's worth of data. Relogging using the *-t 4* parameter creates an output file with 341 intervals over the same duration. Figure 2-19 is an identical System Monitor Chart View using the relogged file instead of the original data zoomed into the same 2-hour time span of interest.



Last 2 Average 5 Minimum 1 Maximum 72
Duration 2:05:31

Color	Scale	Counter	Instance	Parent	Object	Computer
—	0.1000000	Pages/sec	---	---	Memory	\\BEHEMOTH
—	1.000	Processor Queue Length	---	---	System	\\BEHEMOTH
—	1.000	% Processor Time	_Total	---	Processor	\\BEHEMOTH

Figure 2-19 Relogged data showing summarized data

The \System\Processor Queue Length counter is again highlighted. With only about 30 data points to report, the line graphs fall short of spanning the entire x-axis. The Chart View shows every observed data point, which was not the case in the original view. The graphs of the interval counters reveal some smoothing, but not much compared to the original. The average value of the Processor Queue Length counter is 5 in the relogged data, with a maximum value of 72 being reported.

The average values reported in the Report View (not illustrated) for four interval counters are compared in Table 2-22.

Table 2-22 Average Values in Report View

Interval Counter	Original (125 Points)	Relogged (30 Data Points)
Pages/sec	86.194	88.886
% Processor Time	16.820	17.516
% Privileged Time	9.270	9.863
Avg. Disk secs/transfer	0.008	0.009

As expected, the average values for the interval counters in the relogged file are consistent with the original sample.

The Processor Queue Length statistics that System Monitor calculated for this instantaneous value reflect the observations that were dropped from the output file. The maximum observed value for the Processor Queue Length in the relogged counter log file is 72, instead of 81 in the original. Due to chance, the observed maximum value in the original set of observations was lost. The average value, reflecting the underlying uniformity of the distribution of Processor Queue Length values that were observed, remains roughly the same across both views. When the underlying distribution is more erratic, you can expect to see much larger differences in the summary statistics that can be calculated for any instantaneous counters.

Using Typeperf Queries

The Typeperf tool provides a command-line alternative to the Windows System Monitor. Typeperf can provide a running list of performance counter values, giving you detailed performance monitoring in real-time. Typeperf also imposes less overhead than System Monitor. This can be important if the computer you are monitoring is already sluggish or performing poorly.

More Info For more information about Typeperf, in Help and Support Center for Microsoft® Windows Server™ 2003, click Tools, and then click Command-Line Reference A–Z.

To assist in building automated performance monitoring procedures, Typeperf provides an easy way to retrieve a list of all the performance counters installed on a given computer. (Although it is possible to view the set of installed performance counters using System Monitor, there is no way to review and save the entire list.) With Typeperf, you can list the installed performance counters, and save that list to a text file that can be edited later to create a Logman or Relog tool settings file. This capability of Typeperf queries is designed to complement the Log Manager (Logman.exe) and Relog (Relog.exe) command-line tools that have been discussed here.

Command Syntax

Typeperf supports a set of runtime parameters to define counter log queries, along with options to gather counters in real time. Typeperf parameters use syntax and perform functions that are very similar to Logman and Relog. These parameters are summarized in Table 2-23.

Table 2-23 Typeperf Tool Parameters for Gathering Performance Data and Listing Available Queries

Parameter	Syntax	Function	Notes
Query	*-q {Path [Path ...]*	Returns a list of counters, one path per line.	Use *-o* to direct output to a text file.
Extended query	*-qx{Path [Path ...]*	Returns a list of counters with instances.	Extended query output is verbose.
Settings file	*-config FileName*	Use the logging parameters defined in this settings file.	Code counter Path statements one per line.
Counters	*-c {Path [Path ...]*	Specify the counters that you want to gather.	Use *-cf FileName* to use counter settings from an existing log file.
Sample interval	*-si [[HH:]MM:]SS*	Specify the interval between data collection samples.	Defaults to 1 second.
# of samples	*-sc Samples*	Specify the number of data samples to collect.	
Output file name	*-o {FileName}*	Specify the output file name. If the file does not exist, Typeperf will create it.	Redirects output to a file. Defaults to *stdout*.
Log file format	*-f {bin | csv | tsv | SQL}*	Choose the format of the output counter log file.	Defaults to *.csv* format.
Computer	*-s ComputerName*	Specify the computer you want to gather the performance counters from.	If no computer name is provided, the local computer is assumed.

Obtaining a List of Performance Counters Using Typeperf Queries

Before you can monitor performance on a computer, you need to know which performance counters are available on that computer. Although a default set of performance counters is installed along with the operating system, the actual counters present on a given computer will vary depending on such things as:

- The operating system installed. Windows 2000, Windows XP, and Windows Server 2003, for example, all have different sets of default performance counters.
- Additional services or applications installed. Many applications—including Microsoft Exchange and Microsoft SQL Server—provide their own set of performance counters as part of the installation process.
- Whether performance counters have been disabled or become corrupted.

To retrieve a list of all the performance counters (without instances) available on a computer, start Typeperf using the *-q* parameter:

```
Typeperf -q
```

In turn, Typeperf displays the paths of the performance counters installed on the computer. The display looks something like the excerpted set of performance counters shown in Listing 2-6.

Listing 2-6 Abbreviated Typeperf Performance Counter Listing

```
\Processor(*)\% Processor Time
\Processor(*)\% User Time
\Processor(*)\% Privileged Time
\Processor(*)\Interrupts/sec
\Processor(*)\% DPC Time
\Processor(*)\% Interrupt Time
\Processor(*)\DPCs Queued/sec
\Processor(*)\DPC Rate
\Processor(*)\% Idle Time
\Processor(*)\% C1 Time
\Processor(*)\% C2 Time
\Processor(*)\% C3 Time
\Processor(*)\C1 Transitions/sec
\Processor(*)\C2 Transitions/sec
\Processor(*)\C3 Transitions/sec
\Memory\Page Faults/sec
\Memory\Available Bytes
\Memory\Committed Bytes
\Memory\Commit Limit
```

To return a list of all the performance counters available, including instances, use the *-qx* option. Be aware that the *-qx* parameter will return a far greater number of performance counters than the *-q* parameter.

The output from a counter query can be directed to a text file using the *-o* parameter. You can then edit this text file to create a settings file that can be referenced in subsequent Logman and Relog queries.

Retrieving Performance Counters from Remote Computers

You can append a UNC computer name to the command string to obtain a list of performance counters from a remote computer. For example, to list the performance counters on the remote computer DatabaseServer, type the following:

```
Typeperf –q \\DatabaseServer
```

To retrieve only the counters for the Memory object, type this:

```
Typeperf –q \\DatabaseServer\Memory
```

Tip Although Typeperf does not provide a way to specify an alternate user name and password directly, try connecting to the remote system first using a net use *\\remotesystems\ipc$ /user:<name>* Password command to make the connection first. Then issue the Typeperf command on the remote machine.

Monitoring Performance from the Command Line Using Typeperf

After you know which performance counters are installed on a computer, you can begin monitoring performance. To do this, start Typeperf followed by a list of the counters you want to monitor. For example, to monitor the available bytes of memory, type the following, enclosing the counter name in quotation marks:

```
Typeperf “\Memory\Available Bytes"
```

Typeperf will begin monitoring the available bytes of memory, and will display the performance data in real time in the command window. An example of this output is shown in Listing 2-7.

Listing 2-7 Sample Typeperf Output

```
C:\ Typeperf "\Memory\Available bytes"
"(PDH-CSV 4.0)","\\COMPUTER1\Memory\Available bytes"
"10/24/2001 13:41:31.193","35700736.000000"
"10/24/2001 13:41:32.195","35717120.000000"
"10/24/2001 13:41:33.196","35700736.000000"
"10/24/2001 13:41:34.197","35680256.000000"
```

The Typeperf output consists of the date and time the sample was taken, along with the value measured. A new sample is taken (and the command window updated) once every second. This will continue until you press CTRL+C and end the Typeperf session.

Monitoring Multiple Performance Counters Using Typeperf

To monitor two or more performance counters at the same time, include each counter name as part of the *-c* parameter. For example, to simultaneously monitor three separate memory counters, type the following, enclosing each counter name in quotation marks, and separating the individual counters by using commas:

```
Typeperf "\Memory\Available Bytes" "\Memory\Pages/sec" "\Memory\Cache Bytes"
```

Typeperf will display output similar to that shown in Listing 2-8, using commas to separate the fields.

Listing 2-8 Displaying Multiple Counters with Typeperf

```
"(PDH-CSV 4.0)","\\DCPRNTEST\Memory\Available Bytes","\\DCPRNTEST\Memory\Pages/
sec","\\DCPRNTEST\Memory\Cache bytes"
"02/06/2001 09:05:57.464","24489984.000000","0.000000","47214592.000000"
"02/06/2001 09:05:58.516","24489984.000000","0.000000","47214592.000000"
"02/06/2001 09:05:59.567","24530944.000000","0.000000","47214592.000000"
"02/06/2001 09:06:00.619","24514560.000000","0.000000","47214592.000000"
```

As an alternative, you can store counter paths in a settings file, and then reference that file as part of the command string the next time you start Typeperf by using the *-cf* parameter, similar to the way you would with both Logman and Relog.

Monitoring the Performance of Remote Computers Using Typeperf

Typeperf can also monitor performance on remote computers; one way to do this is to include the UNC computer name as part of the counter path. For example, to monitor memory use on the remote computer WebServer, type the following:

```
Typeperf "\\Webserver\Memory\Available Bytes"
```

Alternately, use the local counter name and specify the name of the remote computer by using the *-s* option:

```
Typeperf "\Memory\Available Bytes" -s Webserver
```

To monitor counters from multiple remote computers, use the settings file and specify the computer name UNC as part of the counter path.

Automating Typeperf Usage

By default, Typeperf measures (samples) performance data once a second until you press CTRL+C to manually end the session. This can be a problem if you are recording Typeperf output to a performance log, and no one is available to end the Typeperf session: at one sample per second, it does not take long for a log to grow to an enormous size. For example, if you use the default settings to monitor a single counter every second, after one day, the size of your log file will be more than 4 megabytes.

Note Windows Server 2003 has no limitation on the size of the log file created. However, even though log files of this size are supported, they can be difficult to analyze because of the huge amount of information contained within them.

Modifying the sampling interval Although you cannot place a specific time limit on a Typeperf session—for example, you cannot specify that Typeperf run for two hours and then stop—you can specify the number of samples that Typeperf collects during a given session. Once that number is reached, the session will automatically end.

To limit the number of samples collected in a Typeperf session, use the *-sc* parameter followed by the number of samples to collect. For example, the following command measures memory use 60 times and then stops:

```
Typeperf "\Memory\Available Bytes" -sc 60
```

Because a new sample is taken every second, 60 samples will take approximately one minute. Thus, this session of Typeperf will run for 1 minute, and then shut down.

Modifying the Typeperf sampling rate In addition to specifying the number of samples Typeperf will collect, you can specify how often Typeperf will collect these samples. By default, Typeperf collects a new sample once every second (3,600 times per hour, or 86,400 times in a 24-hour period). For routine monitoring activities, this might be too much information to analyze and use effectively.

To change the sampling rate, use the *-si* parameter, followed by the new sampling time in seconds. For example, this command measures memory use every 60 seconds:

```
Typeperf "\Memory\Available Bytes" -si 60
```

This command measures memory use every 10 minutes (60 seconds × 10):

```
Typeperf "\Memory\Available Bytes" -si 600
```

You must use the same sampling rate for all the performance counters being monitored in any one Typeperf instance. For example, suppose you use the *-si* parameter to change the sampling rate for a set of memory counters:

```
Typeperf "\Memory\Available Bytes" "\Memory\Pages/sec" "\Memory\Cache Bytes" -si 60
```

All three counters must use the sampling rate of 60 seconds. (You cannot assign different sampling rates to individual counters.) If you need to measure performance at different rates, you must run separate instances of Typeperf.

Writing Typeperf output to a counter log Typeperf was initially designed primarily for displaying real-time performance data on the screen. However, the application can also be used to record data in a log file. This allows you to keep a record of your Typeperf sessions, as well as carry out unattended performance monitoring from a script or batch file. For example, you could create a script that starts Typeperf, collects and records performance data for a specified amount of time, and then terminates the Typeperf session.

Note If you plan on collecting performance data on a regular basis (for example, every morning at 4:00 A.M.), you should consider using the Log Manager tool (Logman.exe), which has a built-in scheduling component.

To create a performance log using Typeperf, use the *-o* parameter followed by the path for the log. For example, to save performance data to the file C:\Windows\Logs\Memory.blg, type this command:

```
Typeperf "\Memory\Available Bytes" -o c:\Windows\logs\memory.blg -f bin
```

Note that the default log file type that Typeperf produces is in .csv format. You must specify *-f bin* if you want to create a binary format log file. If the file Memory.blg does not exist, Typeperf will create it. If the file Memory.blg does exist, Typeperf will ask if you want to overwrite the file. You cannot run Typeperf on multiple occasions and have all the information saved to the same log file. Instead, you must create separate log files for each Typeperf session, then use the Relog tool to merge those separate logs into a single file.

When Typeperf output is redirected to a performance log, the output does not appear onscreen. You can view performance data onscreen or you can redirect performance data to a log, but you cannot do both at the same time.

Changing the data format of a Typeperf performance log Unless otherwise specified, Typeperf saves its output in comma-separated values format (CSV). In addition to the CSV format, you can use the *-f* parameter and save the log file in either the TSV (tab-separated values) or BLG (binary log) formats; or to a SQL database.

To change the format of a Typeperf performance log, use the *-o* parameter to specify the file name, and the *-f* parameter to specify the data format. For example, this command creates an output file C:\Windows\Logs\Memory.tsv, saving the data in tab-separated-values format:

```
Typeperf "\Memory\Available Bytes" -o c:\Windows\logs\memory.tab -f TSV
```

Note The *-f* parameter is valid only when output is being directed to a file; it has no effect on output being displayed onscreen. Output is always displayed onscreen as comma-separated values.

Windows Performance Monitoring Architecture

A common set of architectural features of the Windows Server 2003 operating system support the operation of System Monitor; Performance Logs and Alerts; and the Logman, Relog, and Typeperf command-line tools. These performance tools all obtain data by means of using the Performance Data Helper (PDH) dynamic-link library (DLL) as an intermediary.

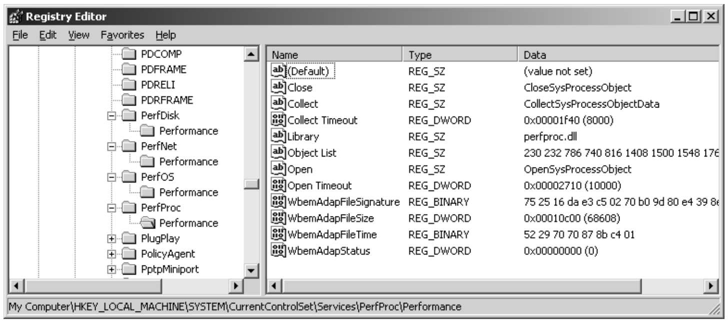
More Info For more information about PDH, see the Windows Server 2003 Software Development Kit (SDK) documentation.

Each of these performance tools gathers counters using the PDH interface. Each tool is then responsible for making the calculations to convert raw data into interval counters and for formatting the data for generating output and reporting.

Performance Library DLLs

Performance Library (Perflib) DLLs provide the raw data for all the measurements you see in System Monitor counters. The operating system supplies a base set of performance library DLLs for monitoring the behavior of resources such as memory, processors, disks, and network adapters and protocols. In addition, many other applications and services in the Windows Server 2003 family provide their own DLLs that install counters that you can use to monitor their operations. All the Performance Library DLLs that are installed on the machine are registered in \Performance subkeys under the HKLM\SYSTEM\CurrentControlSet\Services\<service-name>\ key.

Figure 2-20 shows the four Performance Library DLLs that are supplied with the operating system. They are responsible for gathering disk, network, system-level, and process-level performance counters.



Name	Type	Data
(Default)	REG_SZ	(value not set)
Close	REG_SZ	CloseSysProcessObject
Collect	REG_SZ	CollectSysProcessObjectData
Collect Timeout	REG_DWORD	0x00001f40 (8000)
Library	REG_SZ	perfproc.dll
Object List	REG_SZ	230 232 786 740 816 1408 1500 1548 176
Open	REG_SZ	OpenSysProcessObject
Open Timeout	REG_DWORD	0x00002710 (10000)
WbemAdapFileSignature	REG_BINARY	75 25 16 da e3 c5 02 70 b0 9d 80 e4 39 8
WbemAdapFileSize	REG_DWORD	0x00010c00 (68608)
WbemAdapFileTime	REG_BINARY	52 29 70 70 87 8b c4 01
WbemAdapStatus	REG_DWORD	0x00000000 (0)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\PerfProc\Performance

Figure 2-20 Performance Library DLLs

The entry for the Perfos.dll illustrates the registry fields that are associated with a *Perflib*. Some of these are documented in Table 2-24.

Table 2-24 Registry Fields

Registry Field	Description
Library	Identifies the file name (and path) of the Perflib DLL module. If it is an unqualified file name, %systemroot%\system32 is the assumed location.
Open	The entry point for the Perflib's *Open* routine to be called when initializing a performance counter collection session.
Open Timeout	How long to wait in milliseconds before timing out the call to the *Open* routine. If the *Open* routine fails to return in the specified amount of time, the Disable Performance Counters flag is set.
Collect	The entry point to call every sample interval to retrieve counter data.
Collect Timeout	How long to wait in milliseconds before timing out the call to the *Collect* routine. If the *Collect* routine fails to return in this amount of time, the Disable Performance Counters flag is set.
Close	The entry point for the Perflib's *Close* routine to be called to clean up prior to termination.

These registry fields are used by the PDH routines to load Perflib DLLs, initialize them for use in performance data collection, call them to gather the performance data, and close them when the performance monitoring session is over.

Performance Counter Text String Files

To save space in the registry, the large REG_MULTI_SZ string variables that make up the names and explanatory text of the performance counters are saved in performance counter text string files outside the registry. These files are mapped into the registry so that they appear as normal registry keys to users and applications. The performance counter text string file names are:

- %windir%\system32\perfc009.dat
- %windir%\system32\perfh009.dat

Storing this text data in a separate file also assists in internationalization. Perfc009.dat is the English version of this text data; the 009 represents the language ID of the country, in this case English.

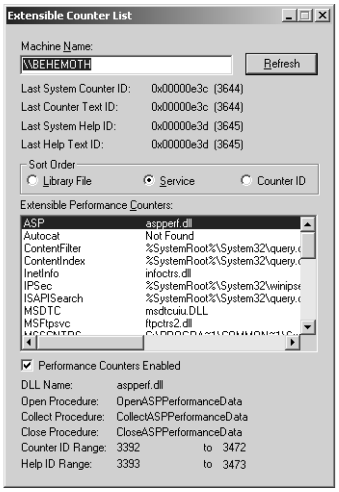
Performance Data Helper Processing

When called by a performance monitoring application, PDH initialization processing involves the following steps:

PDH processing steps

1. PDH accesses the perfc009.dat and perfh009.dat files that contain text that defines the performance objects and counter names that are installed on the machine, along with their associated Explain text.
2. PDH inventories the HKLM\SYSTEM\CurrentControlSet\Services\<service-name>\ key to determine which Perflib DLLs are available on this machine.
3. For each Performance key found, PDH loads the Perflib DLL.
4. After the Perflib DLL is loaded, PDH calls its *Open* routine. The *Open* routine returns information that describes the objects and counters that the Perflib DLL supports. The performance monitoring application can use this information to build an object and counter selection menu like the Add Counters form in System Monitor.
5. At the first sample interval, PDH calls the Perflib *Collect* routine to gather raw performance counters. The performance monitoring application can then make additional PDH calls to format these raw counters for display.
6. At termination, PDH calls the *Close* routine of all active Perflibs so that they end processing gracefully.

PDH processing hides most of the details of these processing steps from performance monitoring applications like the System Monitor and Log Manager. The Extensible Counter List (exctrlst.exe) tool (included as part of the Windows Support Tools for Windows Server 2003) illustrates step 2 of this processing. If you type **exctrlst** on a command line, you will see a display like the one in Figure 2-21.



Machine Name: \\BEHEMOTH

Last System Counter ID: 0x00000e3c (3644)
Last Counter Text ID: 0x00000e3c (3644)
Last System Help ID: 0x00000e3d (3645)
Last Help Text ID: 0x00000e3d (3645)

Sort Order: ( ) Library File [✓ Service] ( ) Counter ID

Extensible Performance Counters:

```
ASP            aspperf.dll
Autocat        Not Found
ContentFilter  %SystemRoot%\System32\query.
ContentIndex   %SystemRoot%\System32\query.
InetInfo       infoctrs.dll
IPSec          %SystemRoot%\System32\winipse
ISAPISearch    %SystemRoot%\System32\query.
MSDTC          msdtcuiu.DLL
MSFtpsvc       ftpctrs2.dll
```

[✓] Performance Counters Enabled

DLL Name: aspperf.dll
Open Procedure: OpenASPPerformanceData
Collect Procedure: CollectASPPerformanceData
Close Procedure: CloseASPPerformanceData
Counter ID Range: 3392 to 3472
Help ID Range: 3393 to 3473

Figure 2-21 Extensible Counter List dialog

The Extensible Counter List tool displays a complete inventory of the Performance Library DLLs that are registered on your machine.

Disable Performance Counters

If a call to a Perflib function fails or returns error status, PDH routines add an optional field to the Performance subkey called Disable Performance Counters. An Application Event log message is also generated when PDH encounters a Perflib error. (More information documenting the Perflib error messages is available in Chapter 6, “Advanced Performance Topics,” in this book.) If a Perflib’s Disable Performance Counters flag is set, PDH routines will not attempt to load and collect counter data from the library until the problem is resolved and you clear the Disable Performance Counters flag by using Exctrlst.exe.

Tip If you are expecting to collect a performance counter but cannot, use exctrlst to check whether the Disable Performance Counters flag has been set for the Perflib responsible for that counter. Once you have resolved the problem that caused the Disable Performance Counters flag to be set, use exctrlst to clear the flag and permit PDH to call the Perflib once again.

Always use the Extensible Counter List tool to reset the Disable Performance Counters flag rather than editing the Registry key directly.

Additional Perflib registry entries are at HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib and are designed to help you when you are encountering data collection problems or other failures inside Performance Library DLLs.

Remote Monitoring

Monitoring performance counters remotely requires that you have network access to the remote computer and an agent on the remote computer that collects performance data and returns it to the local computer that requested the data. The remote collection agent supplied with the Windows Server 2003 family is the Remote Registry service (Regsvc.dll). Regsvc.dll collects performance data about the computer it is running on and provides the remote procedure call (RPC) interface that allows other computers to connect to the remote computer and collect that data. This service must be started and running on the remote computer before other computers can connect to it and collect performance data. Figure 2-22 illustrates the different interefaces and functional elements used when monitoring performance data remotely.

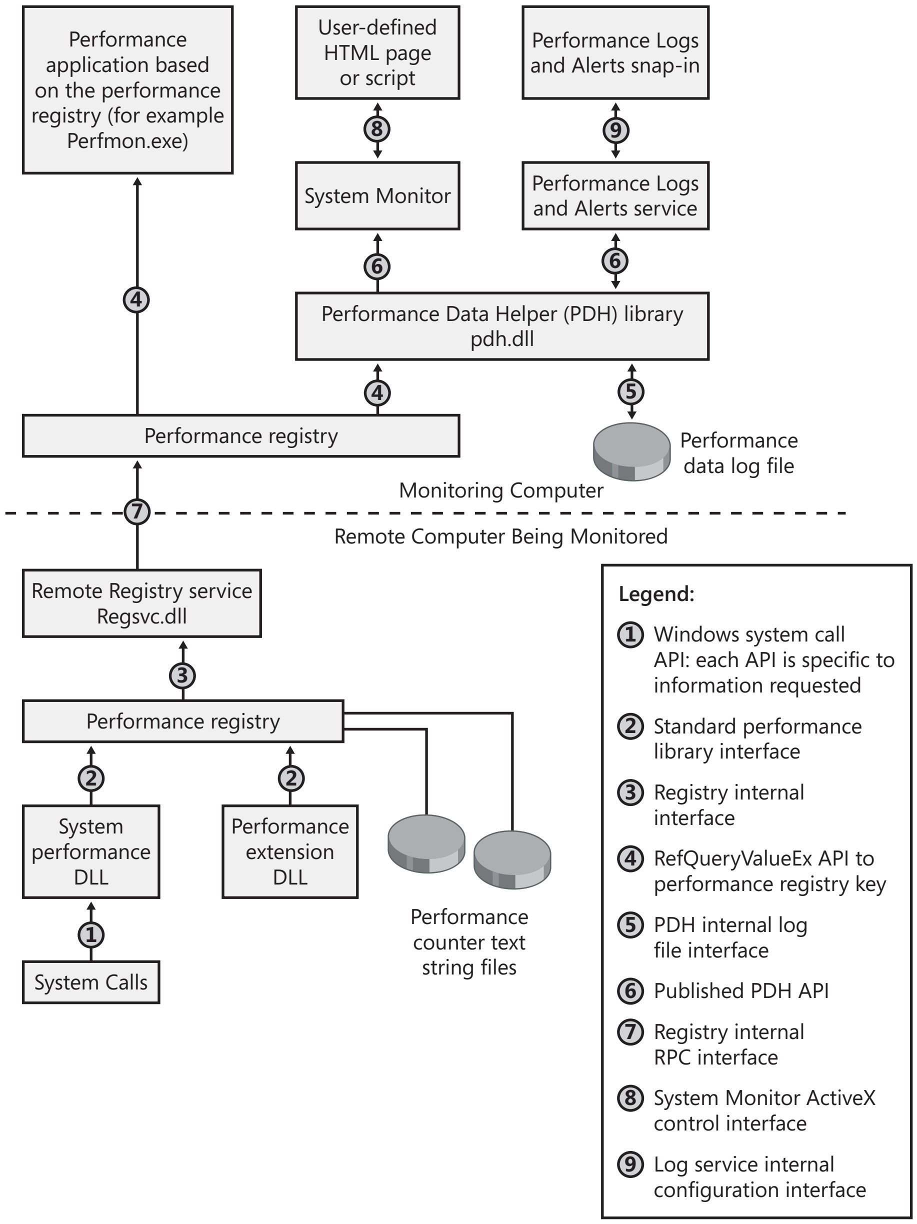

Performance application based on the performance registry (for example Perfmon.exe)
User-defined HTML page or script
Performance Logs and Alerts snap-in
8
9
System Monitor
Performance Logs and Alerts service
6
6
4
Performance Data Helper (PDH) library pdh.dll
4
5
Performance registry
Performance data log file
Monitoring Computer
7
Remote Computer Being Monitored
Remote Registry service Regsvc.dll
3
Performance registry
2
2
System performance DLL
Performance extension DLL
1
System Calls
Performance counter text string files
Legend:
1 Windows system call API: each API is specific to information requested
2 Standard performance library interface
3 Registry internal interface
4 RefQueryValueEx API to performance registry key
5 PDH internal log file interface
6 Published PDH API
7 Registry internal RPC interface
8 System Monitor ActiveX control interface
9 Log service internal configuration interface

Figure 2-22 Remote Performance Monitoring Architecture

Note The Messenger service in the Windows Server 2003 family sends users alert notifications. This service must be running for alert notifications to be received.

Event Tracing for Windows

Event Tracing for Windows (ETW) is an event-oriented instrumentation available from operating system and application providers. These events report precisely when certain performance-oriented events occur, including:

- Context switches
- Page faults
- File I/O requests
- Process creation and termination
- Thread creation and termination
- TCP Send, Receive, and connection requests

In addition, server applications like IIS 6.0 and Active Directory are extensively instrumented to provide diagnostic event traces. In IIS 6.0, for example, the HTTP driver, the Inetinfo process address space, ISAPI filtering, CGI Requests, and even ASP Requests provide specific request start and end events that allow you to trace the progress of an individual HTTP Get Request through various stages of its processing by these components.

Event traces not only record when these events occur, they also capture specific information that can be used to identify the event and the application that caused it. These events can be logged to a file where you can view them or report on them. Event tracing is a technique that you can rely on to diagnose performance problems that are not easy to solve using statistical tools like System Monitor.

The great benefit of event traces is that they are extremely precise. You know exactly what happened and when. But there are potential pitfalls to using event tracing that you also need to be aware of. A drawback of event tracing is the potential to generate large quantities of data that can complicate the analysis of the gathered data. In addition, manual analysis of raw event traces is complex, although Windows Server 2003 includes a built-in trace data-reporting tool called Tracerpt.exe that simplifies matters. If you just wanted to know a simple count of how many events occurred, you could ordinarily gather that information using statistical tools like System Monitor. If you need to understand in detail the sequence of events associated with a specific performance problem, Event Tracing for Windows can provide that information. It can tell you about a wide variety of system and application-oriented events.

Event Tracing Overview

Trace data in Windows Server 2003 is gathered in logging sessions which record data to a trace log file. You can create and manage event tracing sessions using the Log

Manager command-line interface. In addition, the Trace Logs facility in Performance Logs and Alerts in the Performance Monitor console provides an interactive facility for defining event tracing sessions, starting them, and stopping them. However, the interactive Trace Logs facility provides access to only a subset of the Trace definition options that are available using Logman. The Logman interface also has the advantage that it can be used in conjunction with scripts to automate all aspects of event trace logging. The Trace Reporting program, Tracerpt.exe, formats trace data and provides a number of built-in reports.

In a tracing session, you communicate with selected trace data providers that are responsible for reporting whenever designated events occur. When an instrumented event occurs, such as an application sending or receiving a TCP/IP segment or a thread context switch, the provider returns information about the event to the trace session manager, ordinarily the Performance Logs and Alerts service. The Performance Logs and Alerts service then writes a trace event entry in the log file.

Trace logs are saved only in binary format. Trace log files are automatically saved with an .etl extension. You can use either a circular trace file or a sequentially organized trace file. Like counter logs, you can set trace log file-size limits. When a circular trace file reaches its designated size limit, event logging continues by wrapping around to the beginning of the log file and overwriting the oldest trace events with current events. When a sequential trace file reaches its designated size limit, the logging session terminates. Use a circular log file when you want to run an event log tracing session long enough to capture information about some event that occurs unpredictably.

Viewing trace logs requires a parsing tool, such as Tracerpt.exe, to process the trace log output file and convert from binary format to CSV format so that you can read it. Event tracing reports are also available using the *-report* option of tracerpt. Typically, you will be interested in the Trace reports that are available using the *-report* option, rather than in viewing raw CSV files containing the event trace records.

Performance Logs and Alerts

When you open the Performance Monitor, you will notice the Performance Logs and Alerts function in the left tree view. There are three components to Performance Logs and Alerts: counter logs, trace logs, and alerts. This section documents the use of trace logs.

You can create trace logs using the Performance Logs and Alerts tool whenever you require detailed trace data to resolve a performance problem. Reports created from trace logs using the tracerpt tool can provide detailed insight into many problems that are difficult to unravel using performance statistics alone. Performance Logs and Alerts provides trace log capabilities that are similar to those available for counter logs. For example, you can:

- Manage multiple trace logging sessions from a single console window.
- Start and stop trace logging sessions manually, on demand, or automatically, at scheduled times for each log.
- Stop each log based on the elapsed time or the current file size.
- Specify automatic naming schemes and stipulate that a program be run when a trace log is stopped.

There is an upper limit on the number of trace sessions that can run concurrently. This limit is 32. You can define many more trace logs settings than that, but you can activate only 32 trace sessions at a time.

The Performance Logs and Alerts service process, Smlogsvc.exe, is responsible for executing the trace log functions you have defined. Comparable trace performance data logging capabilities are also available using the Logman command-line tool, which also interfaces with the Performance Logs and Alerts service process. This command-line interface to gather event traces is discussed in “Using Log Manager to Create Trace Logs.”

After you load the Performance Logs and Alerts console, you will need to configure the trace logs.

To configure trace logs

1. Click the Trace Logs entry to select it.

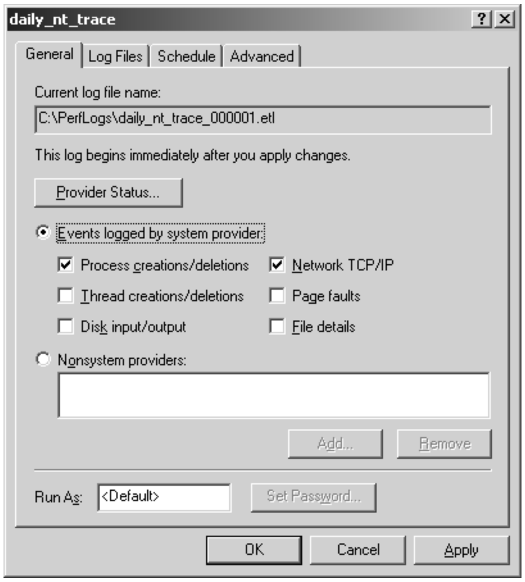
   Previously defined trace log sessions appear in the appropriate node of the details pane.

2. Right-click the details pane to create a new log. You can also use settings from an existing HTML file as a template.

> **Note** To run the Performance Logs and Alerts service, you must be a member of the Performance Log Users or Administrators security groups. These groups have special security access to a subkey in the registry to create or modify a log configuration. (The subkey is HKEY_CURRENT_MACHINE\SYSTEM\CurrentControlSet\Services\ SysmonLog\Log_Queries.)

3. In the New Log Settings box, type the name of your trace log session and click OK.

   Figure 2-23 shows the General tab for the Properties of a new counter log after you enter the counter log name.


daily_nt_trace

General | Log Files | Schedule | Advanced

Current log file name:
C:\PerfLogs\daily_nt_trace_000001.etl

This log begins immediately after you apply changes.

Provider Status...

(●) Events logged by system provider:

- [✓] Process creations/deletions
- [ ] Thread creations/deletions
- [ ] Disk input/output
- [✓] Network TCP/IP
- [ ] Page faults
- [ ] File details

( ) Nonsystem providers:

Add... | Remove

Run As: <Default> | Set Password...

OK | Cancel | Apply

Figure 2-23 General tab for a trace log

4. To configure a trace log, chose either events from the system Provider or one of the available application Providers. Click the Provider Status button to see what specific trace Providers are installed on your machine.
5. Click the Log Files, Schedule, and Advanced options tabs to set the file type, the file naming convention, and other file management options, and to configure the collection period. These options are discussed later.

Trace event providers Providers are responsible for sending information about an event to the Performance Logs and Alerts service when it occurs. By default, on the General tab, the Nonsystem Providers option is selected to keep trace logging overhead to a minimum. Click the Add button to include data from that Provider in the trace log. Application Providers include Active Directory, Microsoft Message Queue, IIS 6.0, and the print spooler. The Processor Trace Information Provider traces Dispatcher events, including thread context switches.

If you click Events Logged By System Provider, a built-in provider for Windows kernel events is used to monitor processes, threads, and other activity. To define kernel events for logging, select the check boxes as appropriate.

The built-in system Provider can trace the following kernel events:

- Process creation and deletion
- Thread creation and deletion by process

- Disk input/output operations specifying logical, physical, and network disk operations by process
- Network TCP/IP Send and Receive commands by process
- Page faults, both hard and soft, by process
- File details for disk input/output operations

Note that the ability to use the system provider to trace registry and image events is restricted to the Log Manager program (Logman.exe).

Before you turn on tracing for a class of event, it helps to understand what the performance impact of tracing might be. To understand the volume of event trace records that can be produced, use System Monitor to track the following counters over several minutes:

- \System\File Data Operations/sec
- \System\File Control Operations/sec
- \Memory\Page Faults/sec
- \TCPv4\Segments/sec

These counters count the events that the kernel trace Provider can trace, assuming you choose to select them. The size of the trace files that are produced and the overhead consumed by a trace is proportional to the number of these events that occur.

Configuring Trace Log Properties The Trace Log Properties sheets that allow you to set up automated event trace procedures. The file and scheduling options for trace logs are very similar to those that are available for counter logs. The log files tab is used to select the file type and automatic file naming options. For example, you can generate unique log file names that are numbered consecutively, or you can add a date and timestamp to the file name automatically. Or you can choose to write all performance data to the same log file; in this case, you specify that current performance data is used to overwrite any older data in the file. Once you set the appropriate option, the Log Files tab displays an example of the automatic file names that will be generated for you.

On the Schedule tab, you can choose manual or automatic startup options. You can then set the time you want the logging session to end using an explicit end time or a duration value in seconds, minutes, hours, or days; or by specifying that it end when the log file reaches its designated size limit.

Trace logging options are summarized in Table 2-25.

Table 2-25 **Summary of Trace Log Properties**

Tab	Settings to Configure	Notes
General	Select Providers	You can collect trace data from the local computer only. Configure system Provider events.
	Account and Password	You can use Run As to provide the logon account and password for data collection on remote computers.
Log Files	File Type	Trace logs are binary files stored with an .etl extension. You can select either circular trace files that wrap around to the beginning when they fill up, or sequential trace files. Use Configure to enter location, file name, and log file size.
	Automatic File Naming	You can choose to add unique file sequence numbers to the file name or append a time and date stamp to identify the file.
Schedule	Manual or Automated Start and Stop Methods and Schedule	You can specify that the log stop collecting data when the log file is full.
	Automated Start and Stop Times	Start and stop by time of day, or specify the log start time and duration.
	Automated Stop When the File Reaches its Maximum Size	
	Processing When the Log File Closes	For continuous data collection, start a new log file when the log file closes. You can also initiate automatic log file processing by running a designated command when the log file closes.
Advanced	Buffer Size	
	Minimum and Maximum buffers	Increase the number of buffers if too many trace events are being skipped.
	Buffer Flush timer	The longest amount of time, in seconds, that a trace entry can remain in memory without being flushed to the disk logging file.

Using Log Manager to Create Trace Logs

Log Manager (Logman.exe) can also be used from the command line to generate event trace logs. This section documents the use of Logman to create and manage trace log files.

More Info For more information about Logman, in Help and Support Center for Microsoft® Windows Server™ 2003, click Tools, and then click Command-Line Reference A–Z.

Command Syntax

Logman operates in one of two modes. In Interactive mode, you can run Logman from a command-line prompt and interact with the logging session. In Interactive mode, for example, you can control the start and stop of a logging session. In Background mode, Logman creates trace log configurations that are scheduled and processed by the same Performance Logs and Alerts service that is used with the Performance Monitor console. For more information about Performance Logs and Alerts, see "Performance Logs and Alerts" in this book.

Table 2-26 summarizes the seven basic Logman subcommands.

Table 2-26 Logman Subcommands

Subcommand	Function
create trace CollectionName	Creates collection queries for either counter data or trace collection sessions.
update CollectionName	Updates an existing collection query to modify the collection parameters.
delete CollectionName	Deletes an existing collection query.
query { CollectionName }	Lists the collection queries that are defined and their status. Use *query CollectionName* to display the properties of a specific collection. To display the properties on remote computers, use the *-s RemoteComputer* option in the command line.
query providers [ProviderName …]	Use *query providers ProviderName* to display a list of parameters that can be set for the specified provider, including their values and descriptions of what they enable. Note that this information is provider-dependant.
start CollectionName	Start a logging session manually.
stop CollectionName	Stop a logging session manually.

Collection queries created using Log Manager contain properties settings identical to the trace logs created using the Performance Logs and Alerts snap-in. If you open the Performance Logs and Alerts snap-in, you will see any collection queries you created using Log Manager. Likewise, if you use the *-query* command-line parameter in Log Manager to view a list of collection queries on a computer, you will also see any trace logs created using the Performance Logs and Alerts snap-in.

Interactive Sessions

The *-ets* command-line switch is used to establish an interactive event trace session. Without it, Logman assumes a scheduled trace collection. When *-ets* is used, Logman will not look at previously saved session configurations. The parameters will be passed to the event trace session directly without being saved or scheduled. Using the *-ets* switch, you can create multiple event trace sessions per console window.

In the following command sequence,

```
logman create trace "mytrace" -pf iistrace.txt –bs 64 -o mytrace.etl
logman start "mytrace" -ets
logman stop "mytrace" -ets
```

The session started by the second command is *not* the one created by the first command. The second Logman command will start a session *mytrace* with default settings because the user specified *-ets* without any other arguments. However, the second command does not erase the saved settings from the first command.

In contrast, without the *-ets* switch,

```
logman create trace "mytrace" -pf iistrace.txt –bs 64 -o mytrace.etl
logman start "mytrace"
logman stop "mytrace"
```

The second and third command will retrieve the settings from the saved session, start and stop the session. Please note that this command sequence looks like an interactive session, but without *-ets*, the commands will still go through the scheduling service, even for immediate start/stop.

Trace Providers

The Logman *-query providers* subcommand allows you to determine which trace providers you can gather trace data from. For example,

```
C:\>logman query providers

Provider                                 GUID
---------------------------------------------------------------------------
ACPI Driver Trace Provider               {dab01d4d-2d48-477d-b1c3-daad0ce6f06b}
Active Directory: Kerberos               {bba3add2-c229-4cdb-ae2b-57eb6966b0c4}
IIS: SSL Filter                          {1fbecc45-c060-4e7c-8a0e-0dbd6116181b}
IIS: WWW Server                          {3a2a4e84-4c21-4981-ae10-3fda0d9b0f83}
IIS: Active Server Pages (ASP)           {06b94d9a-b15e-456e-a4ef-37c984a2cb4b}
Local Security Authority (LSA)           {cc85922f-db41-11d2-9244-006008269001}
Processor Trace Information              {08213901-B301-4a4c-B1DD-177238110F9F}
Windows Kernel Trace                     {9e814aad-3204-11d2-9a82-006008a86939}
ASP.NET Events                           {AFF081FE-0247-4275-9C4E-021F3DC1DA35}
NTLM Security Protocol                   {C92CF544-91B3-4dc0-8E11-C580339A0BF8}
IIS: WWW Isapi Extension                 {a1c2040e-8840-4c31-ba11-9871031a19ea}
HTTP Service Trace                       {dd5ef90a-6398-47a4-ad34-4dcecdef795f}
Active Directory: NetLogon               {f33959b4-dbec-11d2-895b-00c04f79ab69}
Spooler Trace Control                    {94a984ef-f525-4bf1-be3c-ef374056a592}

The command completed successfully.
```

Warning If the application or service associated with the provider is not active on the machine, the provider it is not enabled to gather the corresponding trace events.

Some providers support additional options that allow you to select among the events that they can trace. For example, by default, by using the Windows Kernel Trace Provider, only Process, Thread, and Disk trace events are gathered. You must specifically set the provider flags that correspond to the other events the provider can trace to collect the other kernel trace events.

You can query to see what flags can be set using the *query providers ProviderName* command. You can use either the flag name returned from the *Query Providers* command or set the flag value. A flag value of 0xFFFFFFFF sets all the flags, allowing you to gather all the trace events the provider can supply.

```
C:\>logman query providers "Windows Kernel Trace"

Provider                                 GUID
---------------------------------------------------------------------------
Windows Kernel Trace                     {9e814aad-3204-11d2-9a82-006008a86939}

Flags               Value               Description
---------------------------------------------------------------------------
process             0x00000001          Process creations/deletions
thread              0x00000002          Thread creations/deletions
img                 0x00000004          image description
disk                0x00000100          Disk input/output
file                0x00000200          File details
pf                  0x00001000          Page faults
hf                  0x00002000          Hard page faults
net                 0x00010000          Network TCP/IP
registry            0x00020000          Registry details
dbgprint            0x00040000          Debug print

The command completed successfully.
```

The output of this query lists the flags that the Windows Kernel Trace Provider supports. To trace process creations/deletions, thread creations/deletions, and hard page faults, issue the following Logman command:

```
logman create trace "NT Kernel Logger" -P "Windows kernel trace" (process, thread,hf)
/u mydomain\username *
```

To trace all kernel events, set all the available flags, as follows:

```
logman create trace "NT Kernel Logger" -P "Windows kernel trace" 0xFFFFFFFF /u
mydomain\username *
```

Note The Windows trace Provider can write only to a special trace session called the NT Kernel Logger. It cannot write events to any other trace session. Plus, there can be only one NT Kernel Logger session running at any one time. To gather a Kernel Logger trace, the Performance Logs and Alerts service must run under an account with Administrator credentials.

IIS 6.0 trace providers Comprehensive IIS 6.0 event tracing uses several providers, namely HTTP.SYS, WWW server, WWW Isapi Extension, ASP, ASP.NET, and StreamFilter.

Create a config file (say, named Iisprovs.txt) by using the following lines:

```
"HTTP Service Trace"            0 5
"IIS: WWW Server"               0 5
"IIS: Active Server Pages (ASP)"      0 5
"IIS: WWW Isapi Extension"            0 5
"ASP.NET Events"                      0 5
```

Once you create this file, you can issue the following command to start the IIS 6.0 event trace.

```
logman start "IIS Trace" -pf iisprovs.txt -ct perf -o iistrace.etl -bs 64 –nb 200 400
–ets
```

On high volume Web sites, using more trace buffers and larger buffer sizes might be appropriate, as illustrated. Gathering a kernel trace at the same time is highly recommended.

More Info For more information about and documented procedures for using Logman to gather IIS trace data, see the “Capacity Planning Tracing” topic in the IIS 6.0 Help documentation, which is available with the IIS Administration Microsoft Management Console snap-in.

Active Directory 6.0 trace providers Comprehensive Active Directory trace also uses several providers. Create a config file Adprov.txt that contains the following Active Directory provider names:

```
"Active Directory: Core"
"Active Directory: SAM"
"Active Directory: NetLogon"
"Active Directory: Kerberos"
"Local Security Authority (LSA)"
"NTLM Security Protocol"
```

Then, issue the following command to start an Active Directory trace:

```
logman start ADTrace -pf adprov.txt -o adtrace.etl -ets
```

Gathering a kernel trace at the same time is strongly recommended.

Additional Logman parameters for event tracing include those listed in Table 2-27.

Table 2-27 Logman Parameters for Event Tracing

Parameter	Syntax	Function	Notes
Enable Trace Provider(s)	*-p {GUID | Provider [(Flags[,Flags ...])] Level |*	Specifies the trace data providers use for this session.	Use *-pf [FileName]}* to use provider names and flags from a settings file.
Buffer size	*-bf Value*	Specifies the buffer size in KB used for this trace data collection session.	If trace events occur faster than they can be logged to disk, some trace data can be lost. A larger buffer might be necessary.
Number of buffers	*nb Min Max*	Specifies the minimum and maximum number of buffers for trace data collection.	Minimum default is the number of processors on the system plus two. Default maximum is 25.
Set trace mode options	*-mode [TraceMode [TraceMode ...]]*	*TraceMode* can be globalsequence, localsequence, or pagedmemory.	The pagedmemory option uses pageable memory buffers.
Clock resolution	*-ct {system | perf | cycle}*	*perf* and *cycle* use a 100 ns timer vs. a 1 ms *system* clock.	*-cycle* uses the least overhead. *-perf* provides the most accurate timer resolution
Create and start the trace session	*-ets*	Starts a trace session by using the logging parameters defined on the command line for this session.	
Computer	*-s ComputerName*	Specifies the computer you want to gather the performance counters from.	If no computer name is provided, the local computer is assumed.
Realtime session	*-rt*	Displays trace data in real time. Do not log trace data to a file.	
User mode tracing	*-ul*	Specifies that the event trace session is run in User mode.	In User mode, one provider can be enabled for the event trace session.

Table 2-27 **Logman Parameters for Event Tracing**

Parameter	Syntax	Function	Notes
Output file name	*-o {Path |*	Specifies the output file name. If the file does not exist, Logman will create it.	Required. Event trace log files are in binary format, identified by an .etl extension.
File versioning	*-v {NNNNNN | MMDDHHMM}*	Generates unique file names, either by numbering them consecutively or adding a time and date stamp to the file name.	
Flush timer	*-ft [[HH:]MM:]SS*	Flushes events from buffers after the specified time.	
File size limit	*-max Value*	Specifies the maximum log file or database size in MB.	Logging ends when the file size limit is reached
Logger Name	*ln LoggerName*	Specifies a user-defined name for the event trace logging session.	By default, the logger name is the collection name.
Append	*-a*	Appends the output from this logging session to an existing file.	
Begin logging	*-b M/D/YYYY H:MM:SS* *[{AM | PM}]*	Begins a logging session automatically at the designated date and time.	
End logging	*-e M/D/YYYY H:MM:SS* *[{AM | PM}]*	Ends a logging session automatically at the designated date and time.	Or use *-r* to specify the duration of a logging session.
Log duration	*-rf [[HH:]MM:]SS*	Ends a logging session after this amount of elapsed time.	Or use *-e* to specify a log end date/time.
Repeat	*-r*	Repeats the collection every day at the same time. The time period is based on either the *-b* and *-rf* options, or the *-b* and *-e* options.	Use in conjunction with *-cnf*, *-v*, and *-rc* options.
Start and stop data collection	*-m [start] [stop]*	Starts and stops an interactive logging session manually.	
User name and password	*-u UserName Password*	Specifies user name and password for remote computer access.	The User account must be a member of the Performance Log Users Group. Specify *to be prompted for the password at the command line.

Event Timestamps

If event trace entries appear to be out of order, you might need to use a higher resolution clock. Sometimes, if you use system time as the clock, the resolution (10 ms) might not be fine enough for a certain sequence of events. When a set of events all show the same timestamp value, the order that the events appear in the log is not guaranteed to be the same order in which the events actually occurred. If you see this occurring, use the *perf* clock, which has a finer resolution (100ns). When using Logman, *-ct perf* will force the usage of the *perf* clock.

File Size Limit Checking

If you specify a file size limit using the *-max* value, the file system in which you plan to create the trace log file is evaluated before the trace session starts to see whether an adequate amount of space exists to run the entire trace to completion. This file size limit is performed only when the log file being stored is on the system drive. If the system drive has insufficient space, the logging session will fail, with a generic smlogsvc error message reported in the event log that is similar to the following:

```
Unable to start the trace session for the <session name> trace log configuration. The
Kernel trace provider and some application trace providers require  Administrator
privileges in order to collect data.  Use the Run As option  in the configuration
application to log under an Administrator account for these providers.  System error
code returned is in the data.

The data field is "0000: 70 00 00 00              p...    "
```

The error return code of 70 00 00 00 in this error message indicates an out-of-space condition that prevented the logging session from being started.

Note that additional configuration information is written to the front of every trace log file so that slightly more disk space than you specified in the *-max* parameter is actually required to start a trace session.

Event Trace Reports

You can use the Trace Report (Tracerpt.exe) tool to convert one or more .etl files to .csv format so that you can view the contents of a trace log. To do this, issue the following command:

```
tracerpt iistrace.etl -o iistrace.csv
```

Opening the output csv file in, for example, Microsoft Excel, allows you to view, in sequence, the trace events recorded. The fields in Table 2-28 accompany every trace event.

Table 2-28 **Fields Accompanying Trace Events**

Field	Description
TID	Thread identifies
Clock time	The time the event occurred, using the clock timer resolution in effect for the logging session
Kernel (ms)	Processor time in Kernel mode
User (ms)	Processor time in User mode
User data	Variable, depending on the event type
IID	Instance ID
PID	Parent instance ID

More Info For more information about tracerpt, in Help and Support Center for Microsoft Windows Server 2003, click Tools, and then click Command-Line Reference A–Z.

When the *-report* option is used with a log file that contains trace data from the Windows Kernel Trace, IIS, Spooler, or Active Directory providers, tracerpt generates additional tables in the report that contain preformatted data related to each. For example, the following command generates a report showing tables that incorporate information from the Windows Kernel Trace and IIS providers.

```
tracerpt iistrace.etl  kerneltrace.etl -report iis.html -f html
```

Alerts

The capability to generate real-time alerts automatically based on measurements that exceed designated thresholds is an important aspect of any program of proactive performance monitoring. Performance Logs and Alerts provides an alerting service that can be set to take action when one or more specific counter values have tripped a predetermined limit. In this way, you can be notified of potential performance problems without having to constantly monitor your systems.

This section documents how to set up alerting using the Performance Logs and Alerts facility of the Performance Monitor console. Recommendations for Alert thresholds are provided in Chapter 3, “Measuring Server Performance.” Several practical examples of proactive performance monitoring procedures that utilize alerts are described in Chapter 4, “Performance Monitoring Procedures.”

Configuring Alerts

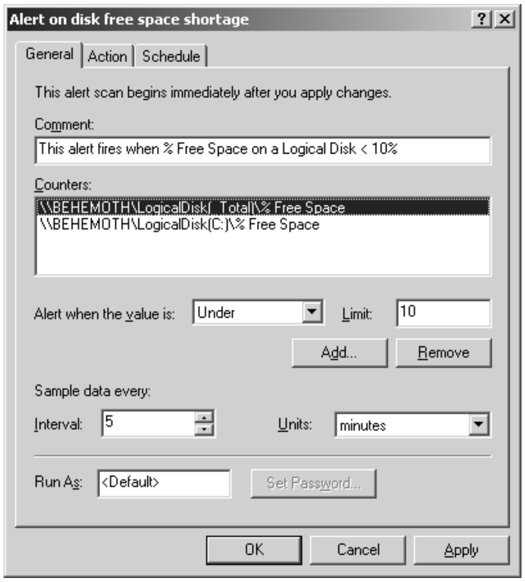
An alert is one or more threshold tests, defined for values of specific performance counters, that trigger an action when those defined measurements exceed their designated threshold values. You can configure numerous actions to occur automatically when the alert occurs, including sending a message, running a designated program to take further action, or starting a counter or event trace logging session. You can also configure multiple alerts to perform different actions.

For example, suppose you have a file server, and you want to log a serious event in the Event log when the free disk space on the primary drive drops below 20 percent. In addition, you might want to be notified via a network message when disk free space drops below a critical 10 percent level, because in that case you might need to take immediate action. To accomplish this, you would configure two different alerts. In both cases, the alert definition will check the same performance counter (\LogicalDisk(D:)\% Free Space). Each separate alert definition will have a different threshold test and perform different actions when the threshold is exceeded. Note that in this example, when the monitored disk space goes below 10 percent, both alerts are triggered and will perform their designated action. Therefore, it would be redundant to configure the 10 percent alert to log an event to the Event log, because the 20 percent alert will already do that. When you use different alerts to monitor the same counter, keep the logical relationship of their limits in mind so that unnecessary actions do not occur as a result of overlapping conditions.

Alerts are identified by a name as well as a more descriptive comment. Each alert definition must have a unique name, and you can define as many alerts as you think appropriate.

Scheduling Alerts

You must consider two aspects of alert scheduling, remembering that each alert you define is scheduled to run independently of every other defined alert. The first scheduling aspect is the frequency with which you want to evaluate the counters that are involved in the threshold test or tests. This is equivalent to a sampling interval. The sampling interval you set determines the maximum number of alert messages or event log entries that can be created per alert test. The sampling interval is set on the General tab of the alert definition Properties pages, as illustrated in Figure 2-24. For example, if you want to test a counter value every 5 seconds and generate an alert message if the designated threshold is exceeded, you would set a sample interval of 5 seconds on the General tab.


Alert on disk free space shortage

This alert scan begins immediately after you apply changes.

Comment: This alert fires when % Free Space on a Logical Disk < 10%

Counters:
\\BEHEMOTH\LogicalDisk(_Total)\% Free Space
\\BEHEMOTH\LogicalDisk(C:)\% Free Space

Alert when the value is: Under Limit: 10

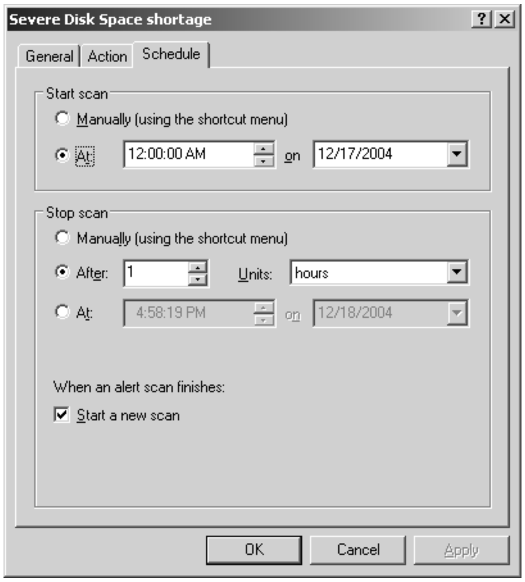
Sample data every:

Interval: 5 Units: minutes

Run As: <Default>

Figure 2-24 The General tab for a Severe Disk Free Space Shortage alert

The other alert actions that can occur, which are running a program automatically or starting an event tracing or counter log session automatically, are governed by a separate scheduling parameter. These actions are performed no more than once per alert session, no matter how many times the alert threshold is exceeded over the course of any single session. Note that these actions, when defined, are performed at the first sampling interval that the alert threshold is exceeded during the session. The duration of the Alert session is set on the Scheduling tab of the alert definition Properties pages, as illustrated in Figure 2-25. For example, if you wanted to gather an IIS 6.0 event trace no more than once per hour if ASP Request execution time exceeds some threshold service level, you would schedule the Alert scan to run for one hour and to start a new scan when the current Alert scan finishes.



Start scan
- ○ Manually (using the shortcut menu)
- ● At: 12:00:00 AM on 12/17/2004

Stop scan
- ○ Manually (using the shortcut menu)
- ● After: 1 Units: hours
- ○ At: 4:58:19 PM on 12/18/2004

When an alert scan finishes:
- ✓ Start a new scan

Figure 2-25 Scheduling properties for an alert

Configuring Alert Thresholds

Adding performance counters to an alert definition is similar to adding performance counters to a performance log query; however, adding them to an alert is a two-step process. The first step is to select the performance counters you want to monitor, and specify the interval between the data samples. The next step, unique to an alert configuration, is to set the limit threshold for each counter. You define the counter values that you want to test and their threshold values on the General property page of the alert definition, as illustrated in Figure 2-24.

When planning your alerting strategy, keep the logical relationship of all related counter threshold values in mind. For example, two entries in the same alert in which the same performance counter is listed twice but has different and overlapping thresholds might be redundant. In the following example, the second condition is unnecessary, because the first one will always be exceeded before the second one is exceeded.

```
\Processor(_Total)\% Processor Time > 80%
\Processor(_Total)\% Processor Time > 90%
```

It can also be worthwhile to have the same counter listed twice in an alert, but *without* overlapping values. For example, you might want to know when the server is not busy so that you can perform routine maintenance; you might also want to be alerted when the server reaches a threshold of excessive activity. You can configure one alert scan to track server usage, as in the following example:

```
\Server\Files Open < 20
\Server\Files Open > 1000
```

In this example, the same counter is monitored; however, the conditions do not overlap.

If you need to monitor the same counter and be alerted to two different but overlapping thresholds, you might want to consider using separate alerts. For example, you can configure a "warning" alert scan that tracks one or more values by using the first—or warning—threshold values and that initiates a warning action (such as logging an event to the event log). You might also want to configure a second alert scan that tracks the "danger" thresholds that require immediate attention. This makes it possible for different actions to be taken at the different thresholds and prevents overlapping thresholds from being masked.

Configuring Alert Notification

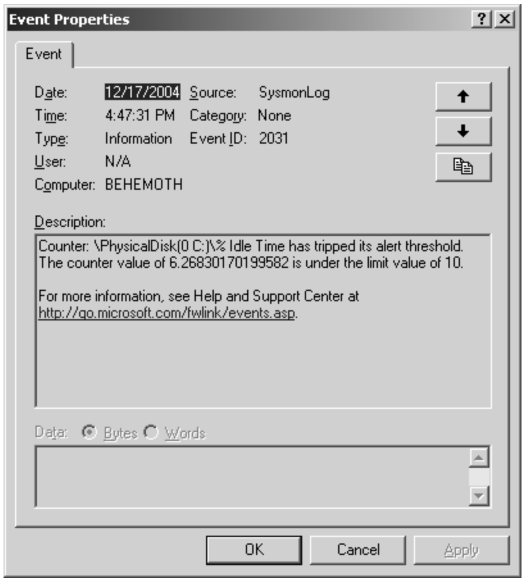
Alerts can be configured to perform several different actions when one of the conditions is met. The action to be taken is configured on the Action tab in the alert property sheet.

Log an Event to an Application Log

This is the default action taken. It can be useful in several ways:

- The event that was generated by an alert condition can be compared to other events in the event log to determine whether there is some correlation between the alert and other system or application events.
- The events in the event log can be used as a record to track issues and alert conditions that occur over time. Event-log analysis tools can be used to further refine this information.

A sample event log entry that the Alert facility creates is illustrated in Figure 2-26. These log entries are found in the Application Event log. The source of these event log messages is identified as SysmonLog with an Event ID of 2031. The body of the event log message identifies the counter threshold that was tripped and the current measured value of the counter that triggered the Alert message.



Date: 12/17/2004 Source: SysmonLog
Time: 4:47:31 PM Category: None
Type: Information Event ID: 2031
User: N/A
Computer: BEHEMOTH

Description:

Counter: \PhysicalDisk(0 C:)\% Idle Time has tripped its alert threshold. The counter value of 6.26830170199582 is under the limit value of 10.

For more information, see Help and Support Center at http://go.microsoft.com/fwlink/events.asp.

Figure 2-26 Event log entry

Send a Network Message

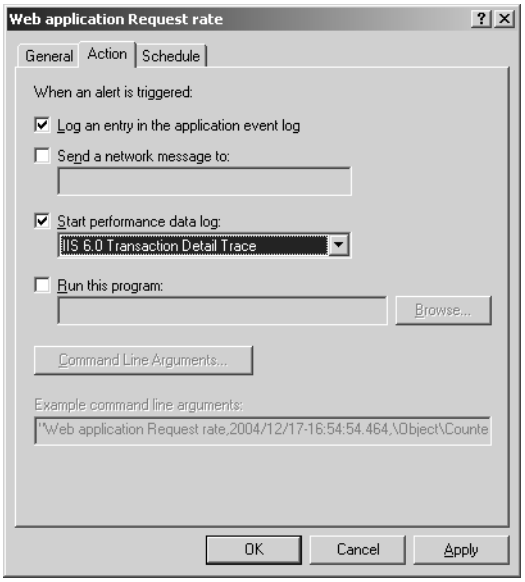
For alert conditions that require immediate attention, a network message can be sent to a specific computer. You can specify either a computer name to send the message to or an IP address.

Start a Performance Data Log

An alert can also be configured to start a performance data log in which additional performance data will be collected. For example, you can configure an alert that monitors processor usage; when that counter exceeds a certain level, Performance Logs and Alerts can start a performance data log that collects data on which processes were running at the time and how much processor time each was using. You can use this feature to collect performance data at critical times without having to collect data when there is nothing noteworthy to observe, thus saving disk space and analysis time.

To log data in response to an alert threshold being reached, you need to create the log query first. Define the log query by using Counter Logs in Performance Logs and Alerts. When configuring the log file that will run in response to an alert trigger, be sure to define a long enough logging session so that you will get enough data to analyze.

After the log query is defined, you can configure the alert by using Alerts in Performance Logs and Alerts to define the alert conditions. On the Action tab in the alert property sheet, select the log query from the list under the Start Performance Data Log check box, as illustrated in Figure 2-27.


Web application Request rate

General | Action | Schedule

When an alert is triggered:

[✓] Log an entry in the application event log

[ ] Send a network message to:

[✓] Start performance data log: IIS 6.0 Transaction Detail Trace

[ ] Run this program:

Example command line arguments:

"Web application Request rate,2004/12/17-16:54:54.464,\Object\Counte

Figure 2-27 Action tab for an alert

In this example, when the alert fires, the alert initiates an event trace logging session. This session gathers both IIS and kernel trace information that will allow you to report on Web site activity. Note that the performance data log session is initiated only once per alert session, corresponding to the first time in the alert session that the alert fires.

Run a Program

The most powerful action an alert can take is to run a command when the alert threshold condition is met. The specified command is passed a command line detailing the alert condition and time. The format of this command line is configured by using the Command Line Arguments dialog box; a sample of the command line is displayed on the Action tab. It is important to make sure the command line sent to the command to be run is formatted correctly for that command. In some cases, it might be necessary to create a command file that reformats the command line so that the command runs properly.

Command-line argument format The information passed to the program can be formatted in several different ways. The information can be passed as a single argument by using the individual information fields delimited by commas, or as separate arguments, each enclosed within double quotation marks and separated by spaces. Choose the format that is most suitable to the program they are being passed to. Note that the program you schedule to run is run only once per alert session, corresponding to the first time in the alert session that the alert fires.

Command-line arguments passed by the alert service might not conform to the arguments expected by another program unless the program was specifically written to be used with the alert service. In most cases, you will need to write a command file that formats the arguments for use by your program, or develop a specific program to accept the arguments passed by the alert service. Here are some examples:

Example 1

```
REM Command file to log alert messages to a text file
REM    This file expects the alert commands to be passed
REM    as a separate strings and for the user text to be
REM          the destination file name.
REM          All alert info should be sent to the command file
REM            %1 = the alert name
REM            %2 = the date/time of the alert
REM            %3 = the counter path
REM            %4 = the measured value
REM            %5 = the alert condition
REM            %6 = the user text (the file name to log this info to)
REM
Echo %1 %2 %3 %4 %5 >>%6
End
```

Example 2

```
REM Command file to send alert data to (an imaginary) program
REM This file expects the alert string to be passed
REM    as a single string. This file adds the command
REM         line switches necessary for this program
REM             %1 = the command string formatted by the alert service
REM
MyLogApp /data=%1 /logtype=alert
End
```

Command-line argument fields The argument passed to the program can contain information fields that describe the alert and the condition that was met. The fields that are passed can be individually enabled or disabled when the alert is configured; however, the order in which they appear cannot. These fields in the following list are described in the order in which they appear in the command-line argument from left to right, or from first argument to last argument, as they are processed by the command-line processor:

- **Alert Name** Name of the alert as it appears in Performance Logs and Alerts in the Performance Console. It is the unique name of this alert scan.
- **Date/Time** Date and time the alert condition occurred. The format is:

  YYYY/MM/DD-HH-MM-SS-mmm

  Where:

  - *YYYY* is the four-digit year.
  - *MM* is the two-digit month.

    - *DD* is the two-digit day.
    - *HH* is the two-digit hour from the 24-hour clock (00=midnight).
    - *MM* is the two-digit minutes past the hour.
    - *SS* is the two-digit seconds past the minute.
    - *mmm* is the number of milliseconds past the second.
- **Counter Name** This is the name of the performance object, the instance (if required), and the counter of the performance counter value that was sampled and tested to meet the specified alert condition.
- **Measured Value** This is the decimal value of the performance counter that met the alert condition.
- **Limit Value** This is the limit condition that was met.
- **Text Message** This is a user-specified text field.

Windows System Resource Manager

Windows System Resource Manager (WSRM) is a new MMC snap-in that comes on a separate CD. It is shipped only with Microsoft Windows Server 2003, Enterprise Edition and Microsoft Windows Server 2003, Datacenter Edition. The benefit of using WSRM is that you can manipulate individual processes or groups of processes to enhance system performance. Processes that are aggregated together into manageable groups are called process matching criteria. WSRM allows you to set limits on CPU usage and memory allocation per process based on process matching criteria. For more information about WSRM, see Chapter 6, "Advanced Performance Topics."

Network Monitor

A poorly performing system is sometimes the result of a bottleneck in your network. Network Monitor is a tool that allows you to monitor your network and detect traffic problems. It also allows you to isolate different types of network traffic, such as all traffic created by accessing the DNS database or all network traffic caused by domain controller replication. By using Network Monitor you can quickly tell what percentage of your network is being utilized and which applications are using too much bandwidth.

A version of Network Monitor with reduced functionality is included with Microsoft Windows Server 2003, Standard Edition; Microsoft Windows Server 2003, Enterprise Edition; and Microsoft Windows Server 2003, Datacenter Edition. It is limited to monitoring local network traffic only. If you want to monitor network traffic on other computers, you must install the version of Network Monitor that comes with Microsoft Systems Management Server.

Chapter 3

Measuring Server Performance

In this chapter:

Using Performance Measurements Effectively 237
Key Performance Indicators 241

Microsoft® Windows Server™ 2003 provides extensive statistics on its operation and performance. You can gather statistics on processor scheduling, virtual memory management, disk operation, and network communications. In addition, server applications such as Active Directory, Internet Information Services (IIS), network file and print sharing services, and Terminal Services provide measurements that enable you to understand what is going on inside these applications.

This chapter identifies the most important performance counters that are used to diagnose and solve performance problems, support capacity planning, and improve operational efficiency. It identifies those counters that are *primary indicators* of specific performance problems related to critical resource shortages. These primary indicators provide direct evidence that specific capacity constraints are potentially limiting current performance levels. This chapter also identifies important *secondary indicators* of performance and capacity problems. Secondary indicators provide more indirect evidence of capacity constraints that are impacting performance levels. Alone, a single secondary indicator is often inconclusive, but a combination of secondary indicators can reliably build a case for specific capacity constraints that are causing problems. In conjunction with primary indicators, these secondary indicators are useful for confirming and supporting your diagnosis and conclusions.

Gathering these performance statistics is useful when you can use the information to diagnose and resolve performance problems. Performance problems arise whenever there is an overloaded resource for which requests waiting to be processed are delayed. Overloaded resources become *bottlenecks* that slow down the processing of requests that your Windows Server 2003 machines must service. For bottleneck detection, it is important to capture measurements showing how busy various computer resources are and the status of the queues where requests are delayed. Fortunately, many Windows Server 2003 performance statistics can help you pinpoint saturated computer resources and identify queues with backed-up requests.

You can also use the performance statistics you gather daily proactively to anticipate performance problems that are brewing and to forecast workload growth. You can then act to relieve a potential bottleneck before it begins to hamper application performance. For forecasting purposes, it is important to capture measures of load, such as requests per second or the number of connected users. Fortunately, there are many metrics available that are good indicators of load and workload growth.

Finally, many performance measurements are important for reasons other than performance tracking, such as for reporting operational problems. Measurements that show system and application availability help pinpoint where operational problems exist. Gathering and reporting performance statistics that show application up time help to measure the stability of your information technology (IT) infrastructure. Tracking and reporting error conditions involving connections lost or error messages sent also focuses attention on these operational issues. This chapter identifies the performance statistics that you should collect regularly to:

- Resolve the performance problems you encounter
- Support the capacity planning process so that you can intervene in a timely fashion to avoid future performance problems
- Provide feedback to IT support staff and customers on operational trends

This chapter also provides tips that will help you set up informative alerts based on performance counter measurements that exceed threshold values. In addition, you'll find measurement notes for many important performance counters that should answer many of your questions about how to interpret the values you observe. Finally, extensive usage notes are provided that describe how to get the most value from the performance monitoring statistics you gather.

Note You will find that the same set of performance counters described in this chapter is available in many other tools. Other applications that access the same performance statistics include Microsoft Operations Manager (MOM) and those developed by third parties. All applications that gather Windows Server 2003 performance measurements share a common measurement interface—a performance monitoring application programming interface (API) discussed in this book in Chapter 2, "Performance Monitoring Tools." The performance monitoring API is the common source of all the performance statistics these tools gather.

Using Performance Measurements Effectively

This chapter focuses on the key performance counters that you should become familiar with to better understand the performance of your machines running Windows Server 2003. Guidance is provided here to explain how these key measurements are derived and how they should be interpreted. It is assumed that you are familiar with the performance monitoring concepts discussed in this book in Chapter 1, "Performance Monitoring Overview." That chapter discusses the relationship between these counters and the hardware and software systems they measure, and understanding these relationships is a prerequisite for performing effective analysis of common computer performance problems. Chapter 4, "Performance Monitoring Procedures," provides a set of recommended performance monitoring procedures that you can use to gather these and related performance counters on a regular basis, which will support problem diagnosis, management reporting, and capacity planning.

Interpreting the performance data you gather can be challenging. It requires considerable expertise in understanding the way computer hardware and operating software work. Interpreting the performance data you gather correctly also requires good analytical and problem-solving skills. The analysis of many computer performance and capacity problems involves identification of resource bottlenecks and the systematic elimination of them. In large-scale environments in which you are responsible for the performance and capacity of many machines, it is also important to take steps to deal with the large amount of performance data that you must potentially gather. This critical topic is discussed thoroughly in Chapter 4, "Performance Monitoring Procedures."

Identifying Bottlenecks

As discussed in Chapter 1, "Performance Monitoring Overview," the recommended way to locate a bottlenecked resource that is the major contributor to a computer performance problem requires the following actions:

- Gathering measurement data on resource utilization at the component level
- Gathering measurement data on queuing delays that are occurring at resources that might be overloaded
- Determining the *relationship* between resource utilization and queuing

Theoretically, a nonlinear relationship exists between utilization and queuing, which becomes evident when a resource approaches saturation. When you detect a nonlinear relationship between utilization and queuing at a resource, there is a good chance that this overloaded resource is causing a performance constraint. You might be able to add additional capacity at this point, or you might be able to tune the system so that demands for the resource are reduced. *Performance tuning* is the process of systematically finding and eliminating resource constraints that constrain performance levels.

For a variety of reasons—some of which were discussed in Chapter 1, "Performance Monitoring Overview"—this nonlinear relationship might not be readily apparent, making bottleneck detection complicated to perform in practice. For example, consider disks and disk arrays that use some form of cache memory. Using small computer system interface (SCSI) command-tag queuing algorithms that sort a queue of requests to favor the requests that can be serviced the fastest improves the efficiency of disk I/O request processing as the disks get busier and busier. This is known as a *load-dependent server*. Another common example is network adaptors that support the Ethernet protocol. The Ethernet collision detection and avoidance algorithm can lead to saturation of the link long before the effective utilization of the interface reaches 100 percent busy.

Specific measurement statistics that you gather should never be analyzed in a vacuum. You will frequently need to augment the general approach to bottleneck detection discussed here with information about how specific hardware and software components are engineered to work. In this chapter, the "Usage Notes" section for each key counter identified discusses how this specific counter is related to other similar counters. In addition, several case studies that show how to identify these and other specific resource bottlenecks are provided in Chapter 5, "Performance Troubleshooting."

Management by Exception

System administrators who are responsible for many Windows Server 2003 machines have an additional challenge: namely, how to keep track of so much measurement data across so many machines. Often this is best accomplished by paying close attention to only that subset of your machines that is currently experiencing critical resource shortages. This approach to dealing with a large volume of information is sometimes known as *management by exception*. Using a management by exception approach, you carefully identify those machines that are experiencing the most severe performance problems and subject them to further scrutiny. Management by exception is fundamentally reactive, so it also needs to be augmented by a proactive approach that attempts to anticipate and avoid future problems.

Several kinds of performance level exceptions need to be considered. *Absolute exceptions* are easy to translate into threshold-based rules. For example, a Web server machine that is part of a load-balanced cluster and is currently not processing any client requests is likely to be experiencing an availability problem that needs further investigation. Unfortunately, in practice, absolute exceptions that can be easily turned into alerting thresholds are rare in computer systems. As a result, this chapter makes very few specific recommendations for setting absolute alerting thresholds for key performance counters.

Exception-reporting thresholds that are related to configuration-specific capacity constraints are much more common. Most of the alerting thresholds that are discussed in this chapter are *relative exceptions*. A threshold rule to define a relative exception requires that you know some additional information about the specific counter, such as:

- Whether there is excessive utilization of a resource relative to the effective capacity of the resource, which might be configuration-dependent
- Whether there is a backlog of requests being delayed by excessive utilization of a resource
- Which application is consuming the resource bandwidth and when this is occurring
- Whether the current measurement observation deviates sharply from historical norms

A good example of an exception that is relative to specific capacity constraints is an alert on the Memory\Pages/sec counter, which can indicate excessive paging to disk. What is considered excessive paging to disk depends to a large degree on the capacity of the disk or disks used to perform I/Os, and how much of that capacity can be devoted to paging operations without negatively impacting the I/O performance of other applications that rely on the same disk or disks. This is a function of both the configuration and of the specific workloads involved. Consequently, it is impossible to recommend a single threshold value for Memory\Pages/sec that should be used to generate a performance alert that you can apply across all your machines running Windows Server 2003.

Although there is no simple rule that you can use to establish unacceptable and acceptable values of many measurements, those measurements can still be effective in helping you identify many common performance problems. The Memory\Pages/sec counter is a key performance indicator. When it reports high rates of paging to disk relative to the capacity of the physical disk configuration, you have a telltale sign that the system is being operated with a physical memory constraint.

Another example of an exception that is relative to specific capacity constraints is an alert on the Processor(_Total)\% Processor Time counter, indicating excessive processor utilization. The specific alerting threshold you choose for a machine to let you know that excessive processor resources are being consumed should depend on the number of processors in the machine, and also on whether those processors are configured symmetrically so that any thread can be serviced on any processor. (Configuring asymmetric processors to boost the performance of large- scale multiprocessors is discussed in Chapter 6, "Advanced Performance Topics.") You might also like to know which processes are associated with the excessive processor utilization that was measured. During disk-to-tape backup, running one or more processors at nearly 100 percent utilization might be expected and even desirable. On the other hand, an application component called from a .NET application that utilizes excessive processor resources over an extended period of time is often a symptom associated with a programming bug that can be very disruptive of the performance of other applications running on the same machine.

Tip To establish meaningful alert thresholds for many relative exceptions, it is important to be able to view specific counter measurements in a broader, environment-specific context.

In many cases, your understanding of what constitutes an exception should be based on deviation from historical norms. Many performance counter measurements, such as System\Context Switches/sec or Processor(_Total)\% Interrupt Time, are meaningful error indicators when they deviate sharply from the measurements you have gathered in the past. A sharp change in the number of the context switches that are occurring, relative to the amount of processor time consumed by a workload, might reflect a programming bug that is degrading performance. Similarly, a sharp increase in % Interrupt Time or % DPC Time might be indirect evidence of hardware errors. This is the whole foundation of statistical quality control methods, for example, which have proved very effective in detecting defects in manufacturing and other mass production processes. Applying *statistical quality control* methods, for example, you might classify a measurement that is two, three, or four standard deviations from an historical baseline as an exception requiring more scrutiny.

Finally, the technique of management by exception can be used to help you focus on machines needing the most attention in large-scale environments in which you must monitor many machines. The management by exception approach to crisis intervention is termed *triage*—that is, classifying problems according to their severity so that the limited time and attention available for devising solutions can be allocated appropriately. This approach suggests, for example, creating Top Ten lists that show the servers that are overloaded the most in their use of critical resources, and, in general, devoting your attention to dealing with the most severe problems first.

Key Performance Indicators

This section reviews the most important performance counters available on machines running Windows Server 2003. These counters are used to report on system and application availability and performance. Key performance indicators are discussed. Measurement notes describe how these indicators are derived, and usage notes provide additional advice on how to interpret these measurements in the context of problem solving and capacity planning. Some basic measures of system and application availability are discussed first, followed by the key counters that report on the utilization of the processor, memory, disk, and network resources. The last section of this chapter discusses some important server applications that are integrated with the base operating system. These include discussions of the performance counters that are available to monitor file and print servers, Web servers, and thin-client Terminal servers.

System and Application Availability

Before you can worry about performance issues, servers and server applications have to be up and running and available for use. This section describes the performance counters that are available to monitor system and application up time and availability. Table 3-1 describes the System\System Up Time counter.

Table 3-1 System\System Up Time Counter

Counter Type	Elapsed time.
Description	Shows the time, in seconds, that the computer has been operational since it was last rebooted.
Measurement Notes	The values of this counter are cumulative until the counter is reset the next time the system is rebooted.
Usage Notes	The primary indicator of system availability.
Performance	Not applicable.
Capacity Planning	Not applicable.
Operations	Reporting on system availability.
Alert Threshold	Not applicable.
Related Measures	Process(*n*)\Elapsed Time.

Availability can also be tracked at the application level by looking at the Process object. Any measurement interval in which a Process object instance is not available means that the process was not running at the end of the data collection interval.

When the process is active, the Process(*n*)\Elapsed Time counter contains a running total that shows how long the process has been active. Note that some processes are short-lived by design. The Process(*n*)\Elapsed Time counter can be used effectively only for long-lived processes. Table 3-2 describes the Process(*n*)\Elapsed Time counter.

Table 3-2 Process(*n*)\Elapsed Time Counter

Counter Type	Elapsed time.
Description	Shows the time, in seconds, that the process has been active since it was last restarted.
Measurement Notes	The process instance exists only during an interval in which the process was found running at the end of the interval. The values of this counter are cumulative across measurement intervals while the process is running.
Usage Notes	The primary indicator of application availability. For system services, compare the value of Process(*n*)\Elapsed Time with the System\System Up Time counter to determine whether the application has been available continuously since the machine was rebooted.
Performance	Not applicable.
Capacity Planning	Not applicable.
Operations	Reporting on application availability.
Alert Threshold	Not applicable.
Related Measures	System\System Up Time.

Other potential measures of system availability are the TCP\Connections Active counter and the Server\Server Sessions counter, which indicate the status of network connectivity. A system that is up and running but cannot communicate with other machines is probably not available for use. For Web application hosting, separate FTP Service\FTP Service Uptime and Web Service\Service Uptime counters are available for those Web server applications.

Processor Utilization

Program execution threads consume processor (CPU) resources. These threads can be part of User-mode processes or the operating system kernel. High-priority device interrupt processing functions are performed by Interrupt Service Routines (ISRs) and deferred procedure calls (DPCs). Performance counters are available that measure how much CPU processing time threads and other executable units of work consume. These processor utilization measurements allow you to determine which applications are responsible for CPU consumption. The performance counters available for monitoring processor usage include the Processor object, which contains an instance for each hardware engine and a *_Total* instance that summarizes usage levels over all available processors. In addition, processor usage is tracked at the process and thread level.

Process-level processor utilization measures are sometimes available for specific server applications like Microsoft SQL Server, too. These applications are not able to report any more detailed information than the process and thread instances provide, but you might find it more convenient to gather them at the application level, along with other related application-specific counters.

Measuring Processor Utilization

Processor utilization statistics are gathered by a Windows Server 2003 operating system measurement function that gains control during each periodic clock interval. This measurement function runs inside the Interrupt Service Routine (ISR) that gains control during clock interrupt processing. The ISR code determines what work, if any, was being performed at the time the interrupt occurred. Each periodic clock interval is viewed as a random sample of the processor execution state. The ISR processor measurement routine determines which process thread was executing, and whether the processor was running in Interrupt mode, Kernel mode, or User mode. It also records the number of threads in the processor Ready Queue.

The ISR processor measurement routine develops an accurate picture of how the processor is being utilized by determining what thread (and what kind of thread) was running just before the interrupt occurred. If the routine that was interrupted when the ISR processor measurement routine gained control is the Idle Thread, the processor is assumed to be Idle.

The operating system accumulates measurement samples 50–200 times per second, depending on the speed and architecture of the machine. This sampling of processor state will be quite accurate for any measurement interval for which at least several thousand samples can be accumulated. At the process level, measurement intervals of 30–60 seconds should provide enough samples to identify accurately even those processes that consume trace amounts of CPU time. For very small measurement intervals in the range of 1–5 seconds, the number of samples that are gathered is too small to avoid sampling errors that might cast doubt on the accuracy of the measurements.

Overall Processor Utilization

The primary indicator of processor utilization is contained in counters from the *_Total* instance of the Processor object. The Processor(_Total)\% Processor Time counter actually reports the *average* processor utilization over all available processors during the measurement interval. Table 3-3 describes the Processor(_Total)\% Processor Time counter.

Table 3-3 Processor(_Total)\% Processor Time Counter

Counter Type	Interval (% Busy).
Description	Overall average processor utilization over the interval. Every interval in which the processor is *not* running the Idle Thread, the processor is presumed to be busy on behalf of some real workload.
Measurement Notes	The processor state is sampled once every periodic interval by a system measurement function. The % Processor Time counter is computed from the ratio of samples in which the processor is detected running the Idle thread compared to the total number of samples, as follows: *100% – ((TotalSamples – IdleThreadSamples) ÷ TotalSamples × 100)*
Usage Notes	The primary indicator of overall processor usage. ■ Values fall within the range of 0–100 percent busy. The *_Total* instance of the processor object calculates average values of the processor utilization instances, not the total. ■ Normalize based on clock speed for comparison across machines. Clock speed is available in the ~MHz field at HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\n. ■ Drill down to process level statistics.
Performance	Primary indicator to determine whether the processor is a potential bottleneck.
Capacity Planning	Trending and forecasting processor usage by workload over time.
Operations	Sustained periods of 100 percent utilization might mean a runaway process. Investigate further by looking at the Process(*n*)\% Processor Time counter to see whether a runaway process thread is in an infinite loop.
Alert Threshold	For response-oriented workloads, beware of sustained periods of utilization above 80–90 percent. For throughput-oriented workloads, extended periods of high utilization are seldom a concern, except as a capacity constraint.
Related Measures	Processor(_Total)\% Privileged Time Processor(_Total)\% User Time Processor(*n*)\% Processor Time Process(*n*)\% Processor Time Thread(*n/Index#*)\% Processor Time

Processors that are observed running for sustained periods at greater than 90 percent busy are running at their CPU capacity limits. Processors observed running regularly in the 75–90 percent range are near their capacity constraints and should be monitored more closely. Processors reported regularly only 10–20 percent busy might be good candidates for consolidation.

Unique hardware factors in multiprocessor configurations and the use of Hyperthreaded logical processors raise difficult interpretation issues. These are discussed in Chapter 6, "Advanced Performance Topics."

Normalizing processor utilization measures The % Processor Time counters are reported as percentage busy values over the measurement interval. For comparison across machines of different speeds, you can use the value of the ~MHz field at HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\n to normalize these measurements to values that are independent of the speed of the specific hardware. Processor clock speed is a good indicator of processor capacity, but a less than perfect one in many cases. Comparisons across machines of the same processor family or architecture are considerably more reliable than comparisons across machines with quite different architectures. For example, it is difficult to compare hyperthreaded multiprocessors with conventional multiprocessors based on clock speed alone, or 32-bit processor families with 64-bit versions. For more discussion about processor architectures and their impact on processor performance, see Chapter 6, "Advanced Performance Topics."

Some processor hardware, especially processor hardware designed for use in battery-powered portable machines, can run at multiple clock speeds. These processors drop to a lower clock speed to save power when they are running on batteries. As a result, the value of the ~MHz field at HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\n might not reflect the current clock speed. The ProcessPerformance\Processor Frequency and ProcessPerformance\% of Maximum Frequency counters enable you to weight processor utilization by the clock speed over a measurement interval for a processor that supports multiple clock speeds.

In the case of hyperthreaded processors, it might make sense to normalize processor utilization of the logical processors associated with a common physical processor core to report the utilization of the physical processor unit. The operating system measures and reports on the utilization of each logical processor. The weighted average of the utilization of the logical processors is a good estimate of the average utilization of the physical processor core over the same interval. Normalizing the measures of processor utilization in this fashion avoids the logical error of reporting greater than 100 percent utilization for a physical processor core. To determine whether the machine is a hyperthreaded multiprocessor or a conventional multiprocessor, User mode applications can make a *GetLogicalProcessorInformation* API call. This API call returns an array of SYSTEM_LOGICAL_PROCESSOR_INFORMATION structures that show the relationship of logical processors to physical processor cores. On a hyperthreaded machine with two physical processors, processor instances 0 and 2 are associated with the first physical processor, and processor instances 1 and 3 are associated with the second physical processor. For more discussion about hyperthreaded processor architectures, see Chapter 6, "Advanced Performance Topics."

Diagnosing processor bottlenecks Observing that the processors on a machine are heavily utilized does not always indicate a problem that you need to address. During disk-to-tape backup operations, for example, it is not unusual for the backup agent to drive processor utilization to near capacity. Your server might be performing many other CPU-intensive tasks including data compression and encryption, which you can expect will be CPU-intensive. Try to drill down to the process level and identify the processes that are the heaviest consumers of % Processor Time. You might also find that breaking down overall processor utilization by processor execution state is useful for determining whether User mode or Kernel mode functions are responsible for driving processor utilization up.

A heavily utilized processor is a concern when there is contention for this shared resource. You need to determine whether a processor capacity constraint is causing contention that would slow application response time, or a single application process is responsible for most of the processor workload. The important indicator of processor contention is the System\Processor Queue Length counter, described in Table 3-4, which measures the number of threads delayed in the processor Ready Queue.

Table 3-4 System\Processor Queue Length Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of threads that are observed as delayed in the processor Ready Queue and waiting to be scheduled for execution. Threads waiting in the processor Ready Queue are ordered by priority, with the highest priority thread scheduled to run next when the processor is idle.
Measurement Notes	The processor queue length is sampled once every periodic interval. The sample value reported as the Processor Queue Length is the *last* observed value of this measurement that was obtained from the processor measurement function that runs every periodic interval.
Usage Notes	Many program threads are asleep in voluntary wait states. The subset of active threads sets a practical upper limit on the length of the processor queue that can be observed.

Table 3-4 System\Processor Queue Length Counter

Performance	Important secondary indicator to determine whether the processor is a potential bottleneck.
Capacity Planning	Normally, not a useful indicator for capacity planning.
Operations	An indication that a capacity constraint might be causing excessive application delays.
Alert Threshold	On a machine with a single very busy processor, repeated observations where Processor Queue Length > 5 is a warning sign indicating that there is frequently more work available than the processor can handle readily. Ready Queue lengths > 10 are a strong indicator of a processor constraint, again when processor utilization also approaches saturation. On multiprocessors, divide the Processor Queue Length by the number of physical processors. On a multiprocessor configured using hard processor affinity to run asymmetrically, large values for Processor Queue Length can be a sign of an unbalanced configuration.
Related Measures	Thread(parent-process\Index#)\Thread State.

It is a good practice to observe the Processor(_Total)\% Processor Time in tandem with the System\Processor Queue Length. Queuing Theory predicts that the queue length should rise exponentially as processor utilization increases. Keep in mind that Processor(_Total)\% Processor Time is based on a continuous sampling technique, whereas the System\Processor Queue Length is an instantaneous value. It is not a simple matter to compare a continuously measured value with an instantaneous one.

Queuing Theory also predicts that the queue length approaches infinity as the processor utilization approaches 100 percent. However, many program threads, especially those inside background service processes, spend most of their time asleep in a voluntary wait state. These threads are normally not vying for the processor. Only active threads do that. Consequently, the number of active threads sets a practical limit on the size of the processor queue length that you are likely to observe.

Another factor limiting the size of the processor Ready Queue is server applications that utilize thread pooling techniques and regulate their thread scheduling internally. Be sure you check whether requests are queued internally inside these applications by checking, for example, counters like ASP\Requests Queued, ASP.NET\Requests Queued, and Server Work Queues(*n*)\Queue length. For more information about thread pooling applications, see Chapter 6, "Advanced Performance Topics."

The Thread(*)\Thread State counter is closely related to the System\Processor Queue Length. Active threads showing a Thread State of 1 are Ready to run. The Thread State of a running thread is 2. When processor contention is evident, being able to determine which process threads are being delayed can be quite helpful. Unfortunately, the volume of thread instances that you need to sift through is normally too large to attempt to correlate Thread(*)\Thread State with the Processor Queue Length over any reasonable period of time.

The size of the processor Ready Queue sometimes can appear disproportionately large, compared to overall processor utilization. This is a by-product of the clock interrupt mechanism that is used to gather the processor Ready Queue length statistics. Because there is only one hardware clock per machine, it is not unusual for threads waiting on a timer interval to get bunched together. If this "bunching" occurs shortly before the last periodic clock interval when the processor Ready Queue Length is measured, the Processor Queue Length can artificially appear quite large. This can sometimes be observed in machines supporting a large number of Terminal Services sessions. Keyboard and mouse movements at Terminal Services client machines are sampled by the server on a periodic basis. The Processor Queue Length value measured might show a large number of Ready threads for the period immediately following session sampling.

Process-Level CPU Consumption

If a Windows Server 2003 machine is dedicated to performing a single role, knowing that machine's overall processor utilization is probably enough information to figure out what to do. However, if the server is performing multiple roles, it is important to drill down and determine which processes are primarily responsible for the CPU usage profile that you are measuring. Statistics on processor utilization that are compiled at the process level allow you to determine which workloads are consuming processor resources. Table 3-5 describes the Process(*instancename*)\% Processor Time counter.

Table 3-5 Process(*instancename*)\% Processor Time Counter

Counter Type	Interval (% Busy).
Description	Total processor utilization by threads belonging to the process over the measurement interval.
Measurement Notes	The processor state is sampled once every periodic interval. % Processor Time is computed as *(Process Busy Samples ÷ Total Samples) × 100.*

Table 3-5 Process(*instancename*)\% Processor Time Counter

Usage Notes	The primary indicator of processor usage at the process level. ■ Values fall within the range of 0–100 percent busy by default. A multithreaded process can be measured consuming more than 100 percent processor busy on a multiprocessor. On multiprocessors, the default range can be overridden by disabling the CapPercents at100 setting in the HKLM\SOFTWARE\Microsoft\Perfmon key. ■ The Process object includes an Idle process instance. Unless capped at 100 percent busy on a multiprocessor, Process(_Total)% Processor Time will report 100 percent busy × the number of processor instances. ■ Volume considerations may force you to gather process level statistics only for specific processes. For example, collect Process(inetinfo)\% Processor Time for Web servers, Process(store)\% Processor Time for Exchange Servers, etc. ■ Process name is not unique. Operating system services are distributed across multiple svchost processes, for example. COM+ server applications run inside instances of Dllhost.exe. IIS 6.0 application pools run in separate instances of the W3wp.exe process. Within a set of processes with the same name, the ID Process counter is unique. ■ Be cautious about interpreting this counter for measurement intervals that are 5 seconds or less. They might be subject to sampling error.
Performance	Primary indicator to determine whether process performance is constrained by a CPU bottleneck.
Capacity Planning	Trending and forecasting processor usage by application over time.
Operations	Sustained periods of 100 percent utilization might mean a runaway process in an infinite loop. Adjust the base priority of the process downward or terminate it.
Alert Threshold	Sustained periods at or near 100 percent busy might mean a runaway process. You should also build alerts for important server application processes based on deviation from historical norms.
Related Measures	Process(*n*)\% Privileged Time Process(*n*)\% User Time Process(n)\Priority Base Thread(*n/Index#*)\% Processor Time Thread(*n/Index#*)\Thread State

In rare cases, you might find it useful to drill further down into a process by looking at its thread data. % Processor Time can also be measured at the thread level. Even more interesting are the Thread(*n/Index#*)\Thread State and Thread(*n/Index#*)\Wait State Reason counters. These are instantaneous counters containing coded values that indicate the execution state of each thread and the reason threads in the Wait state are waiting. Detailed event tracing of the operating system thread Scheduler can also be performed using Event Tracing for Windows.

Processor Utilization by Processor

On a multiprocessor are multiple instances of the Processor object, one for each installed processor, as well as the _*Total* instance that reports processor utilization values that are *averaged* across available processors.

Note On a multiprocessor, counters in the _*Total* instance of the Processor object that are event counters, such as Interrupts/sec, do report totals over all processor instances. Only the % Processor Time measurements are averaged.

By default, multiprocessors are configured for symmetric multiprocessing. *Symmetric multiprocessing* means that any thread is eligible to run on any available processor. This includes Interrupt Service Routines (ISRs) and deferred procedure calls (DPCs), which can also be dispatched on any physical processor. When machines are configured for symmetric multiprocessing, individual processors tend to be loaded evenly. Over any measurement interval, differences in % Processor Time or Interrupts/sec at individual processor level instances should be uniform, subject to some variability because of Scheduler decisions based on soft processor affinity. Otherwise, differences in the performance of an individual processor within a multiprocessor configuration are mainly the result of chance. As a consequence, on symmetric multiprocessing machines, individual processor level statistics are seldom interesting.

However, if the machine is configured for asymmetric processing using the Interrupt Affinity tool in the Windows Server 2003 Resource Kit, WSRM, or application-level processor affinity settings that are available in IIS 6.0 and SQL Server, monitoring individual instances of the processor object can be very important. See Chapter 6, “Advanced Performance Topics,” for more information about using the Interrupt Affinity tool.

Context Switches/Sec

A *context switch* occurs whenever the operating system stops one thread from running and starts executing another thread. This can happen because the thread that was originally running *voluntarily* relinquishes the processor, often because it needs to wait until an I/O finishes before it can resume processing. A running thread can also be *preempted* by a higher priority thread that is ready to run, again, often because an I/O interrupt has just occurred. User-mode threads also switch to a corresponding Kernel mode thread whenever the User-mode application needs to perform a Privileged mode operating system or subsystem service. All of these events are counted as context switches in Windows.

The rate of thread context switches that occur is tallied at the thread level and at the overall system level. This is an intrinsically interesting statistic, but a system administrator usually can do very little about the rate that context switches occur. Table 3-6 describes the System\Context Switches/sec counter.

Table 3-6 System\Context Switches/sec Counter

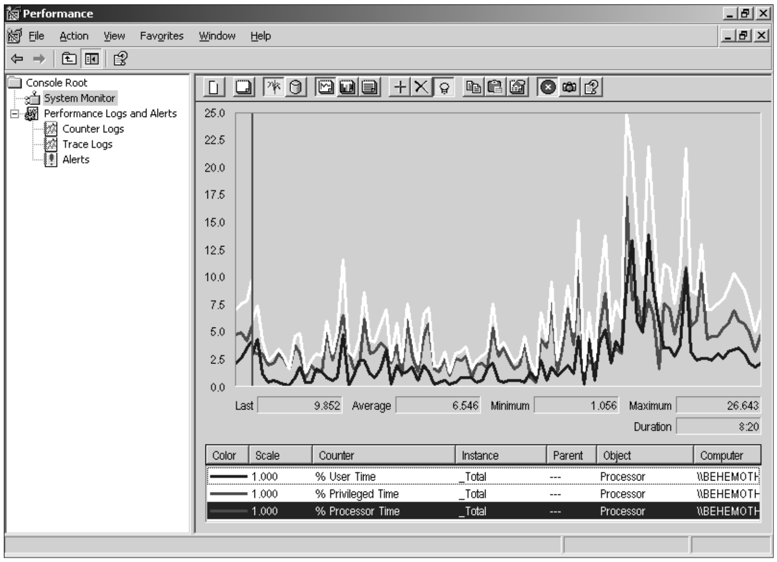
Counter Type	Interval difference counter (rate/second).
Description	A context switch occurs when one running thread is replaced by another. Because Windows Server 2003 supports multithreaded operations, context switches are normal behavior for the system. When a User-mode thread calls any privileged operating system function, a context switch occurs between the User-mode thread and a corresponding Kernel-mode thread that performs the called function in Privileged mode.
Measurement Notes	The operating system counts the number of context switches as they occur. The measurement reported is the difference between the current number of context switches and the number from the previous measurement interval: $(ContextSwitches+_1 - ContextSwitches+_0) \div Duration$
Usage Notes	Context switching is a normal system function, and the rate of context switches that occur is a by-product of the workload. A high rate of context switches is not normally a problem indicator. Nor does it mean the machine is out of CPU capacity. Moreover, a system administrator usually can do very little about the rate that context switches occur. A large increase in the rate of context switches/sec relative to historical norms might reflect a problem, such as a malfunctioning device. Compare Context Switches/sec to the Processor(_Total)\Interrupts/sec counter with which it is normally correlated.
Performance	High rates of context switches often indicate application design problems and might also foreshadow scalability difficulties.
Capacity Planning	Not applicable.
Operations	Not applicable.
Alert Threshold	Build alerts for important server machines based on extreme deviation from historical norms.
Related Measures	Thread\Context Switches/sec.

The number of context switches that occur is sometimes related to the number of system calls, which is tracked by the System\System Calls/sec counter. However, no hard and fast rule governs this relationship because some system calls make additional system calls.

Processor Utilization by Processor Execution State

The Windows Server 2003 operating system measurement function that gathers processor utilization statistics also determines whether the processor was running in Kernel (or Privileged) mode, or User mode.

Figure 3-1 illustrates that `Processor(n)\% Processor Time = Processor(n)\% Privileged Time + Processor(n)\% User Time.`



Last 9.852 Average 6.546 Minimum 1.056 Maximum 26.643
Duration 8:20

Color	Scale	Counter	Instance	Parent	Object	Computer
━	1.000	% User Time	_Total	---	Processor	\\BEHEMOTH
━	1.000	% Privileged Time	_Total	---	Processor	\\BEHEMOTH
━	1.000	% Processor Time	_Total	---	Processor	\\BEHEMOTH

Figure 3-1 Processor utilization by processor execution state

Privileged mode includes both the time the processor is serving interrupts inside Interrupt Service Routines (% Interrupt Time) and executing Deferred Procedure Calls (% DPC Time) on behalf of ISRs, as well as all other Kernel-mode functions of the operating system and device drivers.

There is no set rule to tell you what percent of User Time or Privilege Time to expect. It is a workload-dependent variable. System administrators should mainly take note of measurements that vary significantly from historical norms. In addition, the processor state utilization measurements can provide insight into the type of running thread causing a CPU usage spike. This is another piece of information that can help you narrow down the source of a problem. The Kernrate processor sampling tool can then be used to identify a specific User-mode application or kernel module that is behaving badly. Using Kernrate is discussed in Chapter 5, "Performance Troubleshooting."

Only Interrupt Service Routines (ISRs) run in Interrupt state, a subset of the amount of time spent in Privileged mode. An excessive amount of % Interrupt Time might indicate a hardware problem such as a malfunctioning device. Excluding ISRs, all operating system and subsystem functions run in Privileged state. This includes initiating I/O operations to devices, managing TCP/IP connections, and generating print or graphic display output. A good portion of device interrupt processing is also handled in Privileged mode by deferred procedure calls (DPCs) running with interrupts enabled. User-mode application program threads run in the User state. Many User applications make frequent calls to operating system services and wind up spending a high percentage of time running in Privileged mode.

% Interrupt Time The % Interrupt Time counter measures the amount of processor time devoted to interrupt processing by ISR functions running with interrupts disabled. % Interrupt Time is included in overall % Privileged Time.

Table 3-7 describes the Processor(_Total)\% Interrupt Time counter.

Table 3-7 Processor(_Total)\% Interrupt Time Counter

Counter Type	Interval (% Busy).
Description	Overall average processor utilization that occurred in Interrupt mode over the interval. Only Interrupt Service Routines (ISRs), which are device driver functions, run in Interrupt mode.
Measurement Notes	The processor state is sampled once every periodic interval: *InterruptModeSamples ÷ Total Samples × 100*
Usage Notes	■ The *_Total* instance of the Processor objects calculates average values of the processor utilization instances, not the total. ■ Interrupt processing by ISRs is the highest priority processing that takes place. Interrupts also have priority, relative to the IRQ. When an ISR is dispatched, interrupts at an equal or lower priority level are disabled. ■ An ISR might hand off the bulk of its device interrupt processing functions to a DPC that runs with interrupts enabled in Privileged mode. ■ Interrupt processing is a system function with no associated process. The ISR and its associated DPC service the device that is interrupting. Not until later, in Privileged mode, does the I/O Manager determine which thread from which process was waiting for the I/O operation to complete. ■ Excessive amounts of % Interrupt Time can identify that a device is malfunctioning but cannot pinpoint which device. Use Kernrate, the kernel debugger, to determine which ISRs are being dispatched most frequently.

Table 3-7 Processor(_Total)\% Interrupt Time Counter

Performance	Used to track the impact of using the Interrupt Filter tool to restrict Interrupt processing for specific devices to specific processors using hard processor affinity. See Chapter 6, "Advanced Performance Topics," for more information.
Capacity Planning	Not applicable.
Operations	Secondary indicator to determine whether a malfunctioning device is contributing to a potential processor bottleneck.
Alert Threshold	Build alerts for important server machines based on extreme deviation from historical norms.
Related Measures	Processor(_Total)\Interrupts/sec Processor(_Total)\% DPC Time Processor(_Total)\% Privileged Time

Processor(_Total)\Interrupts/sec records the number of device interrupts that were serviced per second. If you are using the Windows Server 2003 Resource Kit's Interrupt Filter tool to restrict interrupt processing for certain devices to specific processors using hard processor affinity, drill down to the individual processor level to evaluate how this partitioning scheme is working. Monitor both Processor(*n*)\Interrupts/sec and Processor(*n*)\% Interrupt Time to see how your interrupt partitioning scheme is working. See Chapter 6, "Advanced Performance Topics" for more information about using the Windows Server 2003 Resource Kit's Interrupt Filter tool.

% Privileged mode Operating system functions, including ISRs and DPCs, run in Privileged or Kernel mode. Virtual memory that is allocated by Kernel mode threads can be accessed only by threads running in Kernel mode. When a User-mode thread needs to perform an operating system function of any kind, a context switch takes place between the User-mode thread and a corresponding Kernel-mode thread, which changes the state of the machine. Table 3-8 describes the Processor(_Total)\% Privileged Time counter.

Table 3-8 Processor(_Total)\% Privileged Time Counter

Counter Type	Interval (% Busy).
Description	Overall average processor utilization that occurred in Privileged or Kernel mode over the interval. All operating system functions, including Interrupt Service Routines (ISRs) and deferred procedure calls (DPCs), run in Privileged mode. Privileged mode includes device driver code involved in initiating device Input/Output operations and deferred procedure calls that are used to complete interrupt processing.

Table 3-8 Processor(_Total)\% Privileged Time Counter

Measurement Notes	The processor state is sampled once every periodic interval: *PrivilegedModeSamples ÷ Samples × 100*
Usage Notes	■ The *_Total* instance of the Processor objects calculates average values of the processor utilization instances, not the total. ■ The ratio of % Privileged Time to overall % Processor Time is workload-dependent. ■ Drill down to the Process(*n*)\% Privileged Time to determine what application is issuing the system calls.
Performance	Secondary indicator to determine whether operating system functions, including device driver functions, are responsible for a potential processor bottleneck.
Capacity Planning	Not applicable.
Operations	The state of the processor, when a runaway process thread is in an infinite loop, can pinpoint whether a system module is implicated in the problem.
Alert Threshold	Build alerts for important server machines based on extreme deviation from historical norms.
Related Measures	Processor(_Total)\% Interrupt Time Processor(_Total)\% DPC Time Process(*n*)\% Privileged Time

Calculate the ratio of % Privileged Time to overall % Processor Time usage:

```
Privileged mode ratio =
Processor(_Total)\% Privileged Time ÷ Processor(_Total)\% Processor Time
```

No fixed ratio value is good or bad. The relative percentage of Privileged mode CPU usage is workload-dependent. However, a sudden change in this ratio for the same workload should arouse your curiosity and trigger your interest in finding out what caused the change.

% Privileged Time is measured at the overall system level, by processor, and by process. If % Privileged Time at the system level appears excessive, you should be able to drill down the process level and determine which process is responsible for the system calls. You might also need to use Kernrate or the kernel debugger to track down the source of excessive % Privileged Time in an Interrupt Service Routine (ISR) or deferred procedure call (DPC) associated with a device driver.

When multiprocessors are configured to run symmetrically, drilling down to the machine state on individual processors is seldom interesting or useful. However, if you are making extensive use of hard processor affinity, processor-level measurements can reveal how well your partitioning scheme is working.

Monitoring Memory and Paging Rates

Counters in the Memory object report on both physical and virtual memory usage. They also report paging rates associated with virtual memory management. Because of virtual memory management, a shortage of RAM is often evident indirectly as a disk performance problem, when excessive paging to disk consumes too much of the available disk bandwidth. Consequently, paging rates to disk are an important memory performance indicator.

On 32-bit systems, virtual memory is limited to 4 GB, normally partitioned into a 2-GB private area that is unique per process, and a common 2-GB shared range of memory addresses that is common to all processes. On machines configured with large amounts of RAM (for example, 1–2 GB of RAM or more), virtual memory might become exhausted before a shortage of physical memory occurs. When virtual memory becomes exhausted because of a bug in which a program allocates virtual memory but never releases it, the situation is known as a *memory leak*. If virtual memory is exhausted because of the orderly expansion of either a process address space or the system range, the problem is an architectural constraint. In either instance, the result can be catastrophic, leading to widespread application failure and/or a system crash. On 32-bit systems, it is important to monitor virtual memory usage within the system memory pools and at the process address space level.

Virtual Memory and Paging

Physical memory in Windows Server 2003 is allocated to processes on demand. On 32-bit systems, each process address space is able to allocate up to 4 billion bytes of virtual memory. The operating system builds and maintains a set of per-process page tables that are used to map process address space virtual memory locations into physical memory (RAM) pages. At the process level, allocated memory is either reserved or *committed*. When virtual memory is committed, the operating system reserves room for the page, either in RAM or on a paging file on disk, to allow the process to reference that range of virtual addresses.

The current resident set of a process's virtual memory pages is called its *working set*. So long as process working sets can fit readily into RAM, virtual memory addressing has little performance impact. However, when process working sets require more RAM than is available, performance problems can arise. When processes acquire new ranges of virtual memory addresses, the operating system fulfills these requests by allocating pages from a pool of available pages. (Because the page size is hardware-dependent, Available Bytes is reported in bytes, kilobytes, and megabytes.) The Memory Manager attempts to maintain a minimum-sized pool of available pages so that it is capable of granting new pages to processes promptly. The target minimum size of the Available Pages pool is about 8 MB for every 1 GB of RAM. When the pool of available pages becomes depleted, something has to give.

When RAM is full, the operating system is forced to trim older pages from process working sets and add them to the pool of Available pages. Trimmed working set pages that are "dirty"—that contain changed data—must be written to the paging file before they can be granted to another process and used. A page writer thread schedules page writes as soon as possible so that these older pages can be made available for new allocations. When the virtual memory manager (VMM) trims older working set pages, these pages are initially added to the pool of available pages provisionally and stored in the Standby list. Pages on the Standby list are flagged "in transition," and in that state they can be restored to the process working set where they originated with very little performance impact.

Over time, unreferenced pages on the Standby list age and are eventually moved to the Zero list, where they are no longer eligible to transition fault back into their original process working set. A low-priority system thread zeros out the contents of older pages, at which point these pages are moved to the Zero list. The operating system assigns new process working set pages to available pages from either the Zero list or the Free list. The Free list contains pages that have been explicitly freed by the processes that originally allocated them.

The dynamics of virtual memory management are extensively instrumented. Available Bytes represents the total number of pages currently on the Standby list, the Free list, and the Zero list. There are also counters for three different types of page faults: so-called hard page faults (Page Reads/sec), transition faults, and demand zero faults. Pages Output/sec counts the number of trimmed pages that are written to disk. The number of resident system pages and process working set pages is also counted.

A few performance aspects of memory management are not directly visible, but these can usually be inferred from the statistics that are provided. There is, for example, no direct measure of the rate at which page trimming occurs. However, page trimming is normally accompanied by an increase in the rate of transition faults. In addition, the number of trimmed, dirty pages waiting in memory to be written to the paging file is not reported. During sustained periods in which the number of Page Writes/sec reaches 80 percent of total disk transfers to the paging file disk, you might assume that a backlog of trimmed dirty pages is building up.

In a system in which physical memory is undersized, the virtual-memory manager is hard pressed to keep up with the demand for new working set pages. This memory management activity will be reflected in both the high rates of paging to disk (Memory\Pages/sec) and the soft faults (Memory\Transition Faults/sec).

Available Bytes is also an extremely important indicator of physical memory usage; it is a reliable indicator that shows there is an ample supply of RAM. It can also help you identify configurations in which you do not have enough physical memory. Once Available Bytes falls to its minimum size, the effect of page trimming will tend to keep Available Bytes at or near that value until the demand for RAM slackens. Several server applications—notably IIS, SQL Server, and Microsoft Exchange—interact with the Memory Manager to grow their working sets when free RAM is abundant. These server applications will also jettison older pages from their process working sets when the pool of available pages is depleted. This interaction also tends to keep Available Bytes at a relatively constant level at or near its minimum values.

Over longer-term periods, virtual memory Committed Bytes can serve as an indicator of physical memory demand. As the number of Committed Bytes grows larger than the size of RAM, older pages are trimmed from process working sets and relegated to the paging file on disk. The potential for paging operations to disk increases as the number of Committed Bytes grows. This *dynamic* aspect of virtual memory management must be understood when you are planning for RAM capacity on new machines, and when you are forecasting the point at which paging problems are apt to emerge on existing machines with growing workloads.

Pages/sec Because of virtual memory, a shortage of RAM is transformed into a disk I/O bottleneck. Not having enough RAM for your workload is often evident indirectly as a disk performance problem. Excessive paging rates to disk might consume too much of the available disk bandwidth and slow down applications attempting to access their files on the same disk or disks. The Memory\Pages/sec counter, which tracks total paging rates to disk and is described in Table 3-9, is the single most important physical memory performance indicator.

Table 3-9 **Memory\Pages/sec Counter**

Counter Type	Interval difference counter (rate/second).
Description	The number of paging operations to disk during the interval. Pages/sec is the sum of Page Reads/sec and Page Writes/sec.
Measurement Notes	Each paging operation is counted.
Usage Notes	■ Page Reads/sec counters are hard page faults. A running thread has referenced a page in virtual memory that is not in the process working set. Nor is it a trimmed page marked in transition, but rather is still resident in memory. The thread is delayed for the duration of the I/O operation to fetch the page from disk. The operating system copies the page from disk to an available page in RAM and then redispatches the thread. ■ Page writes (the Page Writes/sec counter) occur when dirty pages are trimmed from process working sets. Page trimming is triggered when the pool of available pages drops below its minimum allotment. ■ Excessive paging can usually be reduced by adding RAM. ■ When paging files coexist with application data on the same disk or disks, calculate the percentage of disk paging operations to total disk I/O operations: *Memory\Pages/sec ÷ Physical Disk(_Total)\ Disk Transfers/sec* ■ Disk bandwidth is finite. Capacity used for paging operations is unavailable for other application-oriented file operations. ■ Be aware that the operation of the system file cache redirects normal application I/O through the paging subsystem. Note that the Memory\Cache Faults/sec counter reflects both hard and soft page faults as a result of read requests for files that are cached.
Performance	Primary indicator to determine whether real memory is a potential bottleneck.
Capacity Planning	Watch for upward trends. Add memory when paging operations absorb more than 20–50 percent of your total disk I/O bandwidth.
Operations	Excessive paging can lead to slow and erratic response times.
Alert Threshold	Alert when Pages/sec exceeds 50 per paging disk.
Related Measures	Memory\Available Bytes Memory\Committed Bytes Process(*n*)\Working Set

Disk throughput capacity creates an upper bound on Pages/sec. That is the basis for the configuration rule discussed earlier. If paging rates to disk are high, they will delay application-oriented file I/Os to the disks where the paging files are located.

Physical memory usage When physical memory is a scarce resource, you will observe high Pages/sec rates. The Available Bytes counter is a reliable indicator of when RAM is plentiful. But if RAM is scarce, it is important to understand which processes are using it. There are also system functions that consume RAM. These physical memory usage measurements are discussed in this section.

Available bytes The Memory Manager maintains a pool of available pages in RAM that it uses to satisfy requests for new pages. The current size of this pool is reported in the Memory\Available Bytes counters, described in Table 3-10. This value, like all other memory allocation counters, is reported in bytes, not pages. (The page size is hardware-dependent. On 32-bit Intel-compatible machines, the standard page size is 4096 bytes.) You can calculate the size of the pool in pages by dividing Memory\Available Bytes by the page size. For convenience, there are Available Kbytes and Available Mbytes counters. These report available bytes divided by 1024 and 1,048,576, respectively.

Whenever Available Bytes drops below its minimum threshold, a round of working-set page trimming is initiated to replenish the system's supply of available pages. The minimum threshold that triggers page trimming is approximately 8 MB per 1 GB of RAM, or when RAM is 99 percent allocated. Once RAM is allocated to that extent, working set page trimming tends to keep the size of the Available Page pool at or near this minimum value. This means that Available Bytes can reliably indicate that the memory supply is ample—it is safe to regard any machine with more than 10 percent of RAM available as having an adequate supply of memory for the workload. Once RAM fills up, however, the Available Bytes counter alone cannot distinguish between machines that have an adequate supply of RAM and those that that have an extreme shortage. Direct measures of paging activity, like Pages/sec and Transition Faults/sec, will help you distinguish between these situations.

The Memory Manager organizes Available pages into three list structures. These are the Standby list, the Free list, and the Zero list. The sizes of these lists are not measured separately. Trimmed working-set pages are deposited in the Standby list first. (Trimmed pages that are dirty—that contain modified or changed data—must be written to disk before they can be regarded as immediately available.) Pages on the Standby list are marked "in transition." If a process references a page on the Standby list, it transition faults back into the process working set with very little performance impact. Transition faults are also known as "soft faults," as opposed to hard page faults that require I/O to the paging disk to resolve.

Eventually, unreferenced pages on the Standby list migrate to the Zero list, where they are no longer marked as being in transition. References to new pages can also be satisfied with pages from the Free list. The Free list contains pages that have been explicitly freed by the processes that originally allocated them.

Table 3-10 **Memory\Available Bytes Counter**

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of free pages in RAM available for immediate allocation. Available Bytes counts available pages on the Standby, Free, and Zero lists.
Measurement Notes	Can also be obtained by calling the *GlobalMemoryStatusEx* Microsoft Win32 API function.
Usage Notes	Available Bytes is the best single indicator that physical memory is plentiful. When memory is scarce, Pages/sec is a better indicator. ■ Divide by the size of page to calculate the number of free pages. Available KBytes is the same value divided by 1024. Available MBytes is the same value divided by 1,048,576. ■ Calculate % Available Bytes as a percentage of total RAM (available in Task Manager on the Performance tab): *% Available Bytes = Memory\Available Bytes ÷ sizeof(RAM)* ■ A machine with Available Bytes > 10 percent of RAM has ample memory. ■ The Memory Manager's page replacement policy attempts to maintain a minimum number of free pages. When available memory falls below this threshold, it triggers a round of page trimming, which replenishes the pool of Available pages with older working set pages. So when memory is scarce, Available Bytes will consistently be measured at or near the Memory Manager minimum threshold. ■ The Available Bytes threshold that triggers working-set page trimming is approximately 8 MB per 1 GB of RAM, or less than 1 percent Available Bytes. When Available Bytes falls to this level, monitor Pages/sec and Transition Faults/sec to see how hard the Memory Manager has to work to maintain a minimum-sized pool of Available pages. ■ Some server applications, such as IIS, Exchange, and SQL Server, manage their own working sets. They interact with the Memory Manager to allocate more memory when there is an ample supply and jettison older pages when RAM grows scarce. When these servers' applications are running, Available Bytes measurements tend to stabilize at or near the minimum threshold.
Performance	Primary indicator to determine whether the supply of real memory is ample.
Capacity Planning	Watch for downward trends. Add memory when % Available Bytes consistently drops below 10 percent.
Operations	Excessive paging can lead to slow and erratic response times.
Alert Threshold	Alert when Available Bytes < 2 percent of the size of RAM.

Table 3-10 **Memory\Available Bytes Counter**

Related Measures	Memory\Pages/sec Memory\Transition Faults/sec Memory\Available KBytes Memory\Available MBytes Memory\Committed Bytes Process(n)\Working Set

Process working set bytes When there is a shortage of available RAM, it is often important to determine how the allocated physical memory is being used. Resident pages of a process are known as its *working set*. The Process(*)\Working Set counter, described in Table 3-11, is an instantaneous counter that reports the number of resident pages of each process.

Table 3-11 **Process(*)\Working Set Counter**

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The set of resident pages for a process. The number of allocated pages in RAM that this process can address without causing a page fault to occur.
Measurement Notes	Includes both resident private pages and resident pages mapped to an Image file, which are often shared among multiple processes.
Usage Notes	Process(*n*)\Working Set tracks current RAM usage by active processes. ■ Resident pages from a mapped Image file are counted in the working set of every process that has loaded that Image file. Because of the generous use of shared DLLs, resident pages of active DLLs are counted many times in different process working sets. This is the reason Process(*n*)\Working Set is often greater than Process(*n*)\Private Bytes and Process(*n*)\Virtual Bytes. ■ Divide by the size of a page to calculate the number of allocated pages. ■ Calculate % RAM Used as a percentage of total RAM (available in Task Manager on the Performance tab): *% RAM Used = Process(n)\Working Set ÷ sizeof(RAM)* ■ Some server applications, such as IIS, Exchange, and SQL Server, manage their own process working sets. These server applications build and maintain memory-resident caches in their private process address spaces. You need to monitor the effectiveness of these internal caches to determine whether these server processes have access to an adequate supply of RAM. ■ Monitor Process(_Total)\Working Set in the Process object to see how RAM is allocated overall across all process address spaces.

Table 3-11 Process(*)\Working Set Counter

Performance	If memory is scarce, Process(*n*)\Working Set tells you how much RAM each process is using.
Capacity Planning	Watch for upward trends for important applications.
Operations	Not applicable.
Alert Threshold	Build alerts for important processes based on extreme deviation from historical norms.
Related Measures	Memory\Available Bytes Memory\Committed Bytes Process(*n*)\Private Bytes Process(*n*)\Virtual Bytes Process(*n*)\Pool Paged Bytes

The Windows Server 2003 Memory Manager uses a global last-recently used (LRU) policy. This ensures that the active pages of any process remain resident in RAM. If the memory access pattern of one process leads to a shortage of RAM, all active processes can be affected. At the process level, there is a Page Faults/sec interval counter that can be helpful in determining what processes are being impacted by a real memory shortage. However, the Process(*n*)\Page Faults/sec counter includes all three type of page faults that occur, so it can be difficult to interpret.

Resident pages in the system range Memory resident pages in the system range are counted by two counters in the Memory object: Cache Bytes and Pool Nonpaged Bytes. Cache Bytes is the pageable system working set and is managed like any other process working set. Cache Bytes can be further broken down into Pool Paged Resident Bytes, System Cache Resident Bytes, System Code Resident Bytes, and System Driver Resident Bytes. Both System Code Resident Bytes and System Driver Resident Bytes are usually quite small relative to the other categories of resident system pages. On the other hand, both Pool Paged Resident Bytes and System Cache Resident Bytes can be quite large and might also vary greatly from period to period, depending on memory access patterns. Memory\Cache Bytes is described in Table 3-12.

Table 3-12 Memory\Cache Bytes Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The set of resident pages in the system working set. The number of allocated pages in RAM that kernel threads can address without causing a page fault to occur.
Measurement Notes	Includes Pool Paged Resident Bytes, System Cache Resident Bytes, System Code Resident Bytes, and System Driver Resident Bytes.

Table 3-12 **Memory\Cache Bytes Counter**

Usage Notes	The system working set is subject to page replacement like any other working set. ■ The Pageable pool is an area of virtual memory in the system range from which system functions allocate pageable memory. Pool Paged Resident Bytes is the number of Pool Paged Bytes that are currently resident in memory. ■ Calculate the ratio: *Pool Paged Bytes ÷ Pool Paged Resident Bytes* This ratio can serve as a memory contention index that might be useful in capacity planning. ■ The System Cache is an area of virtual memory in the system range in which application files are mapped. System Cache Resident Bytes is the number of pages from the System Cache currently resident in RAM. ■ The kernel debugger *!vm* command can be used to determine the maximum size of the Paged pool. ■ Add Process(_Total)\Working Set, Memory\Cache Bytes, and Memory\Pool Nonpaged Bytes to see how RAM overall is allocated. ■ Divide by the size of a page to calculate the number of allocated pages.
Performance	If memory is scarce, Cache Bytes tells you how much pageable RAM system functions are using.
Capacity Planning	The ratio of Pool Paged Bytes to Pool Paged Resident Bytes is a memory contention index that might be useful in capacity planning.
Operations	Not applicable.
Alert Threshold	Build alerts for important machines based on extreme deviation from historical norms.
Related Measures	Pool Nonpaged Bytes Pool Paged Resident Bytes System Cache Resident Bytes System Code Resident Bytes System Driver Resident Bytes Process(_Total)\Working Set

There are also system functions that allocate memory from the Nonpaged pool. Pages in the Nonpaged pool are always resident in RAM. They cannot be paged out. An example of a system function that will allocate memory in the Nonpaged pool is working storage that might be accessed inside an Interrupt Service Routine (ISR). An ISR runs in Interrupt mode with interrupts disabled. An ISR that encounters a page fault will crash the system, because a page fault generates an interrupt that cannot be serviced when the processor is already disabled for interrupt processing. The Memory\Pool Nonpaged Bytes counter is described in Table 3-13.

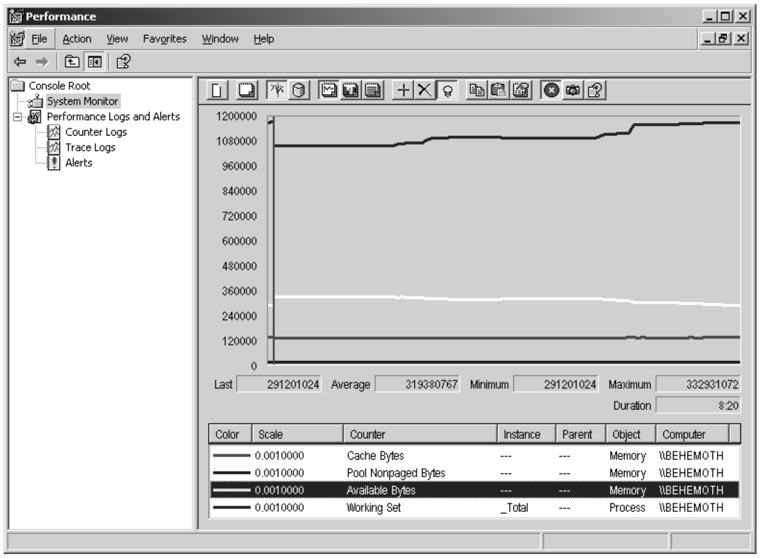
Table 3-13 Memory\Pool Nonpaged Bytes Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	Pages allocated from the Nonpaged pool are always resident in RAM.
Usage Notes	■ Status information about every TCP connection is stored in the Nonpaged pool. ■ The kernel debugger *!vm* command can be used to determine the maximum size of the Nonpaged pool. ■ Divide by the size of a page to calculate the number of allocated pages.
Performance	If memory is scarce, Pool Nonpaged Bytes tells you how much nonpageable RAM system functions are using.
Capacity Planning	Can be helpful when you need to plan for additional TCP connections.
Operations	Not applicable.
Alert Threshold	Build alerts for important machines based on extreme deviation from historical norms.
Related Measures	Pool Paged Bytes Pool Paged Resident Bytes System Cache Resident Bytes System Code Resident Bytes System Driver Resident Bytes Process(_Total)\Working Set

Process(_Total)\Working Set, Memory\Cache Bytes and Memory\Pool Nonpaged Bytes account for how RAM is allocated. If you also add Memory\Available Bytes, you should be able to account for all RAM. Usually, however, this occurs:

```
sizeof(RAM) ≠ Process(_Total)\Working Set +
    Memory\Cache Bytes + Memory\Pool Nonpaged Bytes
    + Memory\Available Bytes
```

This situation arises because the Process(_Total)\Working Set counter contains resident pages from shared DLLs that are counted against the working set of every process that has the DLL loaded. When you are trying to account for how all RAM is being used, you can expect to see something like Figure 3-2. (The example is from a server with 1 GB of RAM installed.)



Last 291201024 Average 319380767 Minimum 291201024 Maximum 332931072
Duration 8:20

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	0.0010000	Cache Bytes	---	---	Memory	\\BEHEMOTH
▬	0.0010000	Pool Nonpaged Bytes	---	---	Memory	\\BEHEMOTH
▬	0.0010000	Available Bytes	---	---	Memory	\\BEHEMOTH
▬	0.0010000	Working Set	_Total	---	Process	\\BEHEMOTH

Figure 3-2 Accounting for RAM usage

Figure 3-2 shows what happens when you add Process(_Total)\Working Set to Cache Bytes, Pool Nonpaged Bytes, and Available Bytes. Although the sum of these four counters accounts for overall RAM usage in the system pools and in process working sets, they clearly do not add up to the size of physical RAM.

In addition to this anomaly, some RAM allocations are not counted anywhere. Trimmed working set pages that are dirty are stored in RAM prior to being copied to disk. There is no Memory counter that tells you how much RAM these pages are currently occupying, although it is presumed to be a small quantity because page writes to disk are scheduled as soon as possible. There is also no direct accounting of RAM usage by the IIS Kernel-mode cache that stores recently requested HTTP Response messages.

Transition faults An increase in the rate of transition faults that are occurring is a clear indicator that the Memory Manager is working harder to maintain an adequate pool of available pages. When the rate of Transition Faults appears excessive, adding RAM should reduce the number of transition faults that occur. Table 3-14 describes the Memory\Transition Faults/sec counter.

Table 3-14 Memory\Transition Faults/sec Counter

Counter Type	Interval difference counter (rate/second).
Description	The total number of soft or transition faults during the interval. Transition faults occur when a recently trimmed page on the Standby list is re-referenced. The page is removed from the Standby list in memory and returned to the process working set. No page-in operation from disk is required to resolve a transition fault.

Table 3-14 Memory\Transition Faults/sec Counter

Measurement Notes	Each type of page fault is counted.
Usage Notes	High values for this counter can easily be misinterpreted. Some transition faults are unavoidable—they are a natural by-product of the LRU-based page-trimming algorithm the Windows operating system uses. High rates of transition faults should not be treated as performance concerns, if other indicators of paging performance problems are not present. ■ When Available Bytes is at or near its minimum threshold value, the rate of transition faults is an indicator of how hard the operating system has to work to maintain a pool of available pages.
Performance	Use Pages/sec and Page Reads/sec instead to detect excessive paging.
Capacity Planning	An upward trend is a leading indicator of a developing memory shortage.
Operations	Not applicable.
Alert Threshold	Do not Alert on this counter.
Related Measures	Memory\Pages/sec Memory\Demand Zero Faults/sec Memory\Page Reads/sec

Demand Zero Faults/sec reflects processes that are acquiring new pages, the contents of which are always zeroed by the operating system before being reassigned to a new process. This is normal behavior for modularly constructed applications that acquire a new heap in each nested function call. As long as processes are releasing older memory pages at approximately the same rate as they acquire new pages on demand–in other words, they are not leaking memory–the Memory Manager should have no trouble keeping up with the demand for new pages.

Page Faults/sec The Memory\Page Faults/sec counter, described in Table 3-15, is the sum of the three types of page faults that can occur: hard faults, which require an I/O to disk; and transition faults and demand zero faults, which do not. The Page Faults/sec counter is the sum of these three measurements: Page Reads/sec, Transition Faults/sec, and Demand Zero Faults/sec. It is recommended that you do not use this field to generate Performance Alerts or alarms of any kind.

Table 3-15 Memory\Page Faults/sec Counter

Counter Type	Interval difference counter (rate/second).
Description	The total number of paging faults during the interval, including both hard and soft faults. Only hard faults—Page Reads/sec—have a significant performance impact. Page faults/sec is the sum of Page Reads/sec, Transition Faults/sec, and Demand Zero Faults/sec.

Table 3-15 **Memory\Page Faults/sec Counter**

Measurement Notes	Each type of page fault is counted.
Usage Notes	High values for this counter can easily be misinterpreted. It is safer to report the following counters separately, rather than this composite number. ■ *Page Reads/sec* are hard page faults. A running thread has referenced a page in virtual memory that is not in the process working set. It is also not a trimmed page marked in transition, but rather is still resident in memory. The thread is delayed for the duration of the I/O operation to fetch the page from disk. The operating system copies the page from disk to an available page in RAM and then redispatches the waiting thread. ■ *Transition Faults/sec* occur when pages trimmed from process working sets are referenced before they are flushed from memory. Page trimming is triggered when the pool of available pages drops below its minimum allotment. The oldest pages in a process working set are trimmed first. Transition faults are also known as "soft" faults because they do not require a paging operation to disk. Because the page referenced is still resident in memory, that page can be restored to the process working set with minimal delay. Many transition faults are unavoidable—they are a natural by-product of the LRU-based page-trimming algorithm that the Windows operating system uses. High rates of transition faults should not be treated as performance concerns. However, a constant upward trend might be a leading indicator of a developing memory shortage. ■ *Demand Zero Faults/sec* measures requests for new virtual memory pages. Older trimmed pages eventually are zeroed by the operating system in anticipation of Demand Zero requests. Windows zeros out the contents of new pages as a security feature. Processes can acquire new pages at very highs rates with little performance impact, unless they never free the memory after it is allocated. ■ Older trimmed pages from the Standby list are *repurposed* when they are moved to the Zero list.
Performance	Use Pages/sec and Page Reads/sec instead to detect excessive paging.
Capacity Planning	Not applicable.
Operations	Not applicable.
Alert Threshold	Do not Alert on this counter.
Related Measures	Memory\Transition Faults/sec Memory\Demand Zero Faults/sec Memory\Page Reads/sec

Virtual Memory Usage

When the active working sets of running processes fit comfortably into RAM, there is little virtual memory management overhead. When the working sets of running processes overflow the size of RAM, the Memory Manager trims older pages from process working sets to try to make room for newer virtual pages. Because of the operating system's use of a global LRU page replacement policy, the memory access pattern of one process can impact other running processes, which could see their working sets trimmed as a result. If excessive paging then leads to a performance bottleneck, it is because the demand for virtual memory pages exceeds the amount of physical memory installed.

Measurements of virtual memory allocations allow you to observe the demand side of a dynamic page replacement policy. Each process address space allocates committed memory that has to be backed by physical pages in RAM or slots on the paging file. Monitoring the demand for virtual memory also allows you detect *memory leaks*—program bugs that cause a process to commit increasing amounts of virtual memory that it never frees up.

Committed Bytes represents the overall demand for virtual memory by running processes. Committed Bytes should be compared to the system's Commit Limit, an upper limit on the amount of virtual pages the system will allocate. The system's Commit Limit is the size of RAM, plus the sizing of the paging file, minus a small amount of overhead. At or near the Commit Limit, memory allocation requests will fail, which is usually catastrophic. Not many applications, system services, or system functions can recover when a request to allocate virtual memory fails. Prior to reaching the Commit Limit, the operating system will automatically attempt to grow the size of the paging file to try to forestall running out of virtual memory. You should add memory or increase the size of your paging file or files to avoid running up against the Commit Limit.

Virtual memory allocations can also be tracked at the process level. In addition, system functions allocate pageable virtual memory from the Paged pool, which is something that should also be monitored.

Committed bytes Virtual memory in a process address space is free (unallocated), reserved, or committed. Committed memory is allocated memory that the system must reserve space for either in physical RAM or out on the paging file so that this memory can be addressed properly by threads running in the process context. Table 3-16 describes the Memory\Committed Bytes counter.

Table 3-16 **Memory\Committed Bytes Counter**

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of committed virtual memory pages. A committed page must be backed by a physical page in RAM or by a slot on the paging file.
Measurement Notes	Can also be obtained by calling the *GlobalMemoryStatusEx* Win32 API function.
Usage Notes	Committed Bytes reports how much total virtual memory process address spaces have allocated. Each committed page will have a Page Table entry built for it when an instruction first references a virtual memory address that it contains. ■ Divide by the size of a page to calculate the number of committed pages. ■ A related measure, % Committed Bytes in Use, is calculated by dividing Committed Bytes by the Commit Limit: *Memory\% Committed Bytes in Use = Memory\Committed Bytes ÷ Memory\Commit Limit* A machine with % Committed Bytes in Use > 90 percent is running short of virtual memory. ■ The Commit Limit is the amount of virtual memory that can be committed without having to extend the paging file or files. The system's Commit Limit is the size of RAM, plus the sizing of the paging file, minus a small amount of overhead. The paging file can be extended dynamically when it is full (if it is not already at its maximum size and there is sufficient space in the file system where it is located.) When the Commit Limit is reached, the system is out of virtual memory. No more than the Commit Limit number of virtual pages can be allocated. ■ Program calls to allocate virtual memory will fail at or near the Commit Limit. The results are usually catastrophic. ■ When a Paging File(*n*)\% Usage approaches 100 percent, the Memory Manager will extend the paging file—if the configuration permits—which will result in an increase to the Commit Limit. ■ Calculate a memory contention index: *Committed Bytes ÷ sizeof(RAM)* If the Committed Bytes:RAM ratio is > 1, virtual memory exceeds the size of RAM, and some memory management will be necessary. As the Committed Bytes:RAM ratio grows above 1.5, paging to disk will usually increase up to a limit imposed by the bandwidth of the paging disks.

Table 3-16 Memory\Committed Bytes Counter

Performance	The Committed Bytes:RAM ratio is a secondary indicator of a real memory shortage.
Capacity Planning	Watch for upward trends in the Committed Bytes:RAM ratio. Add memory when the Committed Bytes:RAM ratio exceeds 1.5.
Operations	Excessive paging can lead to slow and erratic response times.
Alert Threshold	Alert when the Committed Bytes:RAM ratio exceeds 1.5.
Related Measures	Memory\Pages/sec Memory\Commit Limit Memory\% Committed Bytes in Use Memory\Pool Paged Bytes Process(*n*)\Private Bytes Process(*n*)\Virtual Bytes

When a memory leak occurs, or the system is otherwise running out of virtual memory, drilling down to the individual process is often useful. Three counters at the process level describe how each process is allocating virtual memory: Process(*n*)\Virtual Bytes, Process(*n*)\Private Bytes, and Process(*n*)\Pool Paged Bytes.

Process(*n*)\Virtual Bytes shows the full extent of each process's virtual address space, including shared memory segments that are used to map files and shareable image file DLLs. If you need more information about how the virtual memory is allocated inside a process virtual address space, run the Virtual Address Dump (vadump.exe) command-line tool. The use of the vadump command-line tool is illustrated in Chapter 5, "Performance Troubleshooting."

If a process is leaking memory, you should be able to tell by monitoring Process(*n*)\Private Bytes or Process(*n*)\Pool Paged Bytes, depending on the type of memory leak. A memory leak that is allocating but not freeing virtual memory in the process's private range will be reflected in monotonically increasing values of the Process(*n*)\Private Bytes counter, described in Table 3-17. A memory leak that is allocating but not freeing virtual memory in the system range will be reflected in monotonically increasing values of the Process(*n*)\Pool Paged Bytes counter.

Table 3-17 Process(*n*)\Private Bytes Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of a process's committed virtual memory pages that are private. Private page addresses can be addressed only by a thread running in this process context.

Table 3-17 Process(*n*)\Private Bytes Counter

Usage Notes	Process(*n*)\Private Bytes reports how much private virtual memory the process address space has allocated. Process(*n*)\Virtual Bytes includes shared segments associated with mapped files and shareable Image files. ■ Divide by the size of the page to calculate the number of free pages. ■ Identify the cause of a memory leak by finding a process with an increasing number of Process(*n*)\Private Bytes. A process leaking memory may also see growth in its Working Set bytes, but Private Bytes is the more direct symptom. ■ Some outlaw processes may leak memory in the system's Paged Pool. The Process(*n*)\Paged Pool Bytes counter helps you to identify those leaky applications.
Performance	Not applicable.
Capacity Planning	Not applicable.
Operations	Primarily used to identify processes that are leaking memory.
Alert Threshold	In general, do not Alert on this counter value. However, it is often useful to Alert on Process(*n*)\Private Bytes as soon as a process suspected of leaking memory exceeds a critical allocation threshold.
Related Measures	Memory\Commit Limit Memory\% Committed Bytes in Use Process(*n*)\Pool Paged Bytes Process(*n*)\Virtual Bytes

Virtual memory in the system range The upper half of the 32-bit 4-GB virtual address range is earmarked for system virtual memory. The system virtual memory range, 2-GB wide, is divided into three major pools: the Nonpaged pool, the Paged pool, and the system file cache. When the Paged pool or the Nonpaged pool is exhausted, system functions that need to allocate virtual memory will fail. These pools can be exhausted before the system Commit Limit is reached. If the system runs out of virtual memory for the file cache, file cache performance could suffer, but the situation is not as dire.

The size of the three main system area virtual memory pools is determined initially based on the amount of RAM. These initial allocation decisions can also be influenced by a series of settings in the HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management key. These settings are discussed in more detail in Chapter 6, "Advanced Performance Topics." The size of these pools is also adjusted dynamically, based on virtual memory allocation patterns, to try and avoid shortages in one area or another. Still, sometimes shortages can still occur, with machines configured with large amounts of RAM being the most vulnerable. Using the boot options that shrink the system virtual address range in favor of a larger process private address range sharply increases the risk of running out of system PTEs. These boot options are discussed in Chapter 6, "Advanced Performance Topics."

System services and other functions allocate pageable virtual memory from the Pageable pool. A system function called by a process could also allocate pageable virtual memory from the Pageable pool. If the Pageable pool runs out of space, system functions that attempt to allocate virtual memory from the Pageable pool will fail. It is possible for the Pageable pool to be exhausted long before the Commit Limit is reached. Registry configuration and tuning parameters that can affect the size of the Pageable pool are discussed in Chapter 6, "Advanced Performance Topics." The Memory\Paged Pool Bytes Counter is described in Table 3-18.

Table 3-18 Memory\Paged Pool Bytes Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description:	The number of committed virtual memory pages in the system's Paged pool. System functions allocate virtual memory pages that are eligible to be paged out from the Paged pool. System functions that are called by processes also allocate virtual memory pages from the Paged pool.
Usage Notes	Memory\Paged Pool Bytes reports how much virtual memory is allocated in the system Paged pool. Memory\Paged Pool Resident Bytes is the current number of Paged pool pages that are resident in RAM. The remainder is paged out. ■ Divide by the size of a page to calculate the number of allocated virtual pages. ■ A memory leak can deplete the Paged pool, causing system functions that allocate virtual memory from the Paged pool to fail. You can identify the culprit causing a memory leak by finding a process with an increasing number of Process(*n*)\Paged Pool Bytes. A process that is leaking memory might also see growth in its Working Set bytes, but Paged Pool Bytes is the more direct symptom. ■ Some outlaw processes might leak memory in the system's Paged pool. The Process(*n*)\Paged Pool Bytes counter helps you to identify those leaky applications.
Performance	Not applicable.
Capacity Planning	Not applicable.
Operations	Primarily used to identify processes that are leaking memory.
Alert Threshold	In general, do not Alert on this counter value. However, it is often useful to Alert on Process(*n*)\Paged Pool Bytes as soon as a process suspected of leaking memory exceeds a critical allocation threshold.
Related Measures	Memory\Commit Limit Memory\% Committed Bytes in Use Process(*n*)\Pool Paged Bytes Process(*n*)\Virtual Bytes

Depending on what it is doing, a process could also leak memory in the system's Paged pool. The Process(*n*)\Pool Paged Bytes counter allows you to identify processes that are leaking memory in the system Paged pool. The Memory\Nonpaged Pool Bytes counter is described in Table 3-19.

Table 3-19 Memory\Nonpaged Pool Bytes

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of allocated pages in the system's Nonpaged pool. System functions allocate pages from the Nonpaged pool when they require memory that cannot be paged out. For example, device driver functions that execute during interrupt processing must allocate memory from the Nonpaged pool.
Usage Notes	Memory\Nonpaged Pool Bytes reports how much memory is allocated in the system Nonpaged pool. Because pages in the Nonpaged pool cannot be paged out, this counter measures both virtual and real memory usage. ■ Divide by the size of the page to calculate the number of allocated virtual pages. ■ If the Nonpaged pool fills up, key system functions might fail. ■ Important functions that allocate memory from the Nonpaged pool include TCP/IP session connection data that is accessed during Network Interface interrupt processing.
Performance	Not applicable.
Capacity Planning	Sizing and planning for network connections.
Operations	Used to identify device drivers that are leaking memory.
Alert Threshold	In general, do not Alert on this counter value. However, it is often useful to Alert on Process(*n*)\Nonpaged Pool Bytes as soon as a process suspected of leaking memory exceeds a critical allocation threshold.
Related Measures	Memory\Pool Paged Bytes.

System PTEs are built and used by system functions to address system virtual memory areas. When the system virtual memory range is exhausted, the number of Free System PTEs drops to zero, and no more system virtual memory of any type can be allocated. On 32-bit systems with large amounts of RAM (1–2 GB or more), tracking the number of Free System PTEs is important. Table 3-20 describes the Memory\Free System Page Table Entries counter.

Table 3-20 **Memory\Free System Page Table Entries Counter**

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of free System PTEs. A free System PTE is used to address virtual memory in the system range. This includes both the Paged pool and the Nonpaged pool. When no free System PTEs are available, calls to allocate new virtual memory areas in the system range will fail.
Usage Notes	Memory\Paged Pool Bytes reports how much virtual memory is allocated in the system Paged pool. Memory\Paged Pool Resident Bytes is the current number of Paged pool pages that are resident in RAM. The remainder is paged out. ■ The system virtual memory range is exhausted when the number of free System PTEs drops to zero. At that point, no more system virtual memory of any type can be allocated. ■ On 32-bit systems with 2 GB or more of RAM, tracking the number of free System PTEs is important. Those systems are vulnerable to running out of free System PTEs.
Performance	Not applicable.
Capacity Planning	Not applicable.
Operations	Primarily used to identify processes that are leaking memory.
Alert Threshold	Alert when the number of free System PTEs < 100.
Related Measures	Memory\Commit Limit Memory\% Committed Bytes in Use Process(*n*)\Pool Paged Bytes Process(*n*)\Virtual Bytes

System functions allocate pageable virtual memory from a single, shared Paged pool. A process or device driver function that leaks memory from the Paged pool will deplete and eventually exhaust the pool. When the pool is depleted, subsequent requests to allocate memory from the Paged pool will fail. Any operating system function, device driver, or application process that requests virtual memory is subject to these memory allocation failures. It is not always easy to pinpoint exactly which application is responsible for this pool becoming exhausted. Fortunately, at least one server application—the file Server service—reports on Paged pool memory allocation failures when they occur. This counter can prove helpful even when a server is not primarily intended to serve as a network file server. Table 3-21 describes the Server\Paged Pool Failures counter.

Table 3-21 **Server\Paged Pool Failures Counter**

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The cumulative number of Paged pool allocation failures that the Server service experienced since being initialized.
Usage Notes	The file Server service has a number of functions that allocate virtual memory pages from the Paged pool. ■ If a memory leak exhausts the Paged pool, the file Server service might encounter difficulty in allocating virtual memory from the Paged pool. ■ If a call to allocate virtual memory fails, the file Server service recovers gracefully from these failures and reports on them. ■ Because many other applications and system functions do not recover gracefully from virtual memory allocation failures, this counter can be the only reliable indicator that a memory leak caused these allocation failures.
Performance	Not applicable.
Capacity Planning	Not applicable.
Operations	Primarily used to identify a virtual memory shortage in the Paged pool.
Alert Threshold	Alert on any nonzero value of this counter.
Related Measures	Memory\Pool Paged Bytes Memory\Commit Limit Memory\% Committed Bytes in Use Server\Pool Paged Bytes Process(*n*)\Pool Paged Bytes

Memory-Resident Disk Caches

Memory-resident disk caches are one of the major consumers of RAM on many Windows Server 2003 machines. Many applications can run faster if they cache frequently accessed data in memory rather than access it from disk repeatedly. The system file cache is an area of system virtual memory that is reserved for the purpose of storing frequently accessed file segments in memory for quicker access. The system file cache has three interfaces: the Copy interface, which is used by default; the Mapping interface, which is used by system applications that need to control when cached writes are written to the disk; and the MDL interface, which is designed for applications that need access to physical memory buffers. Each of these interfaces is associated with a separate set of the counters in the Cache object.

Besides the built-in system file cache, some applications build and maintain their own memory-resident caches specifically designed to cache objects other than files. IIS 6.0 operates a Kernel-mode cache that caches frequently requested HTTP Response messages. Because the IIS Kernel-mode driver operates this cache, the HTTP Response cache is built using physical memory.

SQL Server and Exchange both build caches to store frequently accessed information from disk databases. The SQL Server and Exchange caches are carved from the process private address space. On 32-bit machines, they both can benefit from an extended private area address space. SQL Server might also benefit from being able to access more than 4 GB of RAM using the Physical Address Extension (PAE) and the Address Windowing Extensions (AWE). PAE and AWE are discussed in Chapter 6, "Advanced Performance Topics"

Each of the application-oriented caches is instrumented and provides performance counters. These application cache counters are mentioned briefly here. When they are active, these caches tend to be larger consumers of RAM. The most important performance measurements associated with caches are the amount of memory they use, the rate of read and write activity to the cache, and the percentage of cache hits. A *cache hit* is an access request that is satisfied from current data that is stored in the cache, not on disk. Usually, the more memory devoted to the cache, the higher the rate of cache hits. However, even small amounts of cache memory are likely to be very effective, while allocating too much physical memory to the cache is likely to be a waste of resources.

A *cache miss* is a request that misses the cache and requires a disk access to satisfy. Most applications read through the cache, which means that following a cache miss, the data requested is available in the cache for subsequent requests. When writes occur, the disk copy of the data is no longer current and must be updated. Most caches defer writes to disk as long as possible. This is also known as *lazy write*. Then, after enough dirty blocks in cache have accumulated, writes are flushed to the disk in efficient, bulk operations. For data integrity and recoverability reasons, applications sometimes need to control when data on disk is updated. The System File Cache's Mapping Interface provides that capability, for example. Table 3-22 describes the Memory\System Cache Resident Bytes counter.

Table 3-22 **Memory\System Cache Resident Bytes Counter**

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of resident pages allocated to the System File Cache. The System File Cache occupies a reserved area of the system virtual address range. This counter tracks the number of virtual memory pages from the File Cache that are currently resident in RAM.
Usage Notes	On file print and servers, System Cache Resident Bytes is often the largest consumer of RAM. Compare memory usage to each of the following file cache hit ratios: ■ Copy Read Hits %: The Copy interface to the cache is invoked by default when a file is opened. The System Cache maps the file into the system virtual memory range and copies file data from the system range to the process private address space, where it can be addressed by the application. ■ Data Map Hits %: Returns virtual addresses that point to the file data in the system virtual memory range. Requires that application threads run in Privileged mode. The Mapping interface supports calls to Pin and Unpin file buffers to control the timing of physical disk writes. Used by the Redirector service for the client-side file cache and Ntfs.sys for caching file system metadata. ■ MDL Read Hits %: MDL stands for Memory Descriptor List, which consists of physical address parameters passed to DMA controllers. Requires that application threads run in Privileged mode and support physical addresses. Used by the Server service for the server-side file cache, and IIS for caching htm, .gif, .jpg, .wav, and other static files. ■ On a System File Cache miss, a physical disk I/O to an application file is performed. The paging file is unaffected. PTEs backing the File Cache do not point directly to RAM, but instead to Virtual Address Descriptors (VADs). ■ Divide by the size of a page to calculate the number of allocated virtual pages. ■ The System File Cache reserves approximately 1 GB of virtual memory in the system range by default. ■ System Cache Resident Bytes is part of the system's working set (Cache Bytes) and is subject to page trimming when Available Bytes becomes low.

Table 3-22 Memory\System Cache Resident Bytes Counter

Performance	When the System File Cache is not effective, performance of server applications that rely on the cache are impacted. These include Server, Redirector, NTFSs, and IIS.
Capacity Planning	Not applicable.
Operations	Primarily used to identify processes that are leaking memory.
Alert Threshold	Do not Alert on this counter value.
Related Measures	Memory\Cache Bytes Memory\Transition Pages rePurposed/sec Cache\MDL Read Hits % Cache\Data Map Hits % Cache\Copy Read Hits %

IIS version 6.0 relies on a Kernel-mode cache for HTTP Response messages in addition to the User-mode cache, in which recently accessed static objects are cached. The kernel cache is used to store complete HTTP Response messages that can be returned to satisfy HTTP GET Requests without leaving Kernel mode. Web service cache performance statistics for both the Kernel mode and User mode caches are available in the Web Service Cache object. SQL Server 2000 cache statistics per database are available in the SQL Server:Cache Manager object.

Monitoring Disk Operations

Performance statistics on both logical and physical disks are provided by measurement layers in the I/O Manager stack, as described in Chapter 1, "Performance Monitoring Overview." These measurement functions track and compute performance information about disk I/O requests at both the Logical and Physical Disk level. Logical Disk performance measurements are provided by the Logical Disk I/O driver: either the Ftdisk.sys driver for basic volumes or Dmio.sys for dynamic volumes. Physical disk measurements are maintained by the Partmgr.sys driver layer. Interval performance statistics include the activity rate to disk, the disk % Idle Time, the average response time (including queue time) of disk requests, the bytes transferred, and whether the operations were Reads or Writes. Additional disk performance statistics are then calculated by the PerfDisk.dll Performance Library based on these measurements, including the Avg. Disk Queue Length.

To diagnose and resolve disk I/O performance problems, calculating some additional disk statistics beyond those that are provided automatically is extremely useful. A few simple calculations allow you to generate some important additional disk performance metrics, namely, disk utilization, average disk service time, and average disk queue time. Being able to decompose disk response time, as reported by the Avg. Disk secs/Transfer counters, into device service time and queue time allows you to distinguish between a device that is running poorly and a device that is overloaded.

A *logical disk* represents a single file system with a unique drive letter, for example. A *physical disk* is the internal representation of a SCSI Logical Unit Number (LUN). When you are using array controllers and RAID disks, the underlying physical disk hardware characteristics are not directly visible to the operating system. These physical characteristics—the number of disks, the speed of the disks, the RAID-level organization of the disks—can have a major impact on performance. Among simple disk configurations, device performance characteristics vary based on seek time, rotational speed, and bit density. More expensive, performance-oriented disks also incorporate on-board memory buffers that boost performance substantially during sequential operations. In addition, disk support for SCSI tagged command queuing opens the way for managing a device's queued requests so that ones with the shortest expected service time are scheduled first. This optimization can boost the performance of a disk servicing random requests by 25–50 percent in the face of queued requests.

What appears to the operating system as a simple disk drive might in fact be an array of disks behind a caching controller. Disk arrays spread the I/O load evenly across multiple devices. Redundant array of independent disks (RAID) provides storage for duplicate data, in the case of disk mirroring schemes; or parity data, in the case of RAID 5, that can be used to reconstitute the contents of a failed device. Maintaining redundant data normally requires additional effort, leading to I/O activity within the disk array that was not directly initiated by the operating system. Any I/Os within the array not directly initiated by the operating system are not counted by the I/O Manager instrumentation.

Similarly, I/Os that are resolved by controller caches are counted as physical I/O operations whether or not there is physical I/O to the disk or disks configured behind the cache that occurs. For battery-backed, nonvolatile caches, write operations to disk are frequently deferred and occur only later asynchronously. When cached writes to disk are deferred, device response time measurements obtained by the I/O Manager instrumentation layers are often much lower than expected. This is because the cache returns a successful I/O completion status to the operating system almost immediately as soon as the data transfer from host memory to the controller cache memory completes.

When you run any of these types of devices, the performance data available from System Monitor needs to be augmented by configuration and performance information available from the array itself. See Chapter 5, "Performance Troubleshooting," for an in-depth discussion of disk performance measurements issues when disk array controllers and disk controller caches are present.

It is important to be proactive about disk performance because it is prone to degrade rapidly, particularly when memory-resident caches start to lose their effectiveness or disk-paging activity erupts. The interaction between disk I/O rates and memory cache effectiveness serves to complicate disk capacity planning. (In this context, paging to disk can be viewed as a special case of memory-resident disk caching where the most active virtual memory pages are cached in physical RAM.)

Cache effectiveness tends to degrade rapidly, leading to sharp, nonlinear spikes in disk activity beginning at the point where the cache starts to lose its effectiveness. Consequently, linear trending based on historical patterns of activity is often not a reliable way to predict future activity levels. Disk capacity planning usually focuses, instead, on provisioning to support growing disk space requirements, not poor performance. However, planning for adequate disk performance remains an essential element of capacity planning. Because each physical disk has a finite capacity to service disk requests, the number of physical disks installed usually establishes an upper bound on disk I/O bandwidth. The Physical Disk(*n*)\Avg. Disk secs/transfer counter is described in Table 3-23.

Note Logical disk and physical disk statistics are defined and derived identically by I/O Manager instrumentation layers, except for the addition of two file system disk space usage measurements in the Logical Disk object.

Table 3-23 **Physical Disk(*n*)\Avg. Disk secs/transfer Counter**

Counter Type	Average.
Description	Overall average response time of physical disk requests over the interval. Avg. Disk secs/transfer includes both device service time and queue time.
Measurement Notes	The start and end time of each I/O Request Packet (IRP) is recorded by the I/O Manager instrumentation layer. The result, averaged over the interval, is the round trip time (RTT) of a disk request.

Table 3-23 Physical Disk(*n*)\Avg. Disk secs/transfer Counter

Usage Notes	The primary indicator of physical disk I/O performance. Physical disks are the equivalent of SCSI LUNs. Performance is dependent on the underlying disk configuration, which is transparent to the operating system. ■ Individual disks range in performance characteristics based on seek time, rotational speed, recording density, and interface speed. More expensive, performance-oriented disks can provide 50 percent better performance. ■ Disk arrays range in performance based on the number of disks in the array and how redundant data is organized and stored. RAID 5 disk arrays, for example, suffer a significant performance penalty when writes occur. ■ Disk cache improves performance on read hits up to the interface speed. Deferred writes to cache require reliable, on-board battery backup of cache memory.
Performance	Primary indicator to determine whether the disk is a potential bottleneck.
Capacity Planning	Not applicable.
Operations	Poor disk response time slows application response time.
Alert Threshold	Depends on the underlying disk hardware.
Related Measures	Physical Disk(*n*)\Disk Transfers/sec Physical Disk(*n*)\% Idle Time Physical Disk(*n*)\Current Disk Queue Length

If disks are infrequently accessed, even very poor disk response time is not a major problem. However, when Physical Disk\Disk Transfers/sec exceeds 15–25 disk I/Os per second per disk, the reason for the poor disk response time should be investigated. When Avg. Disk secs/transfer indicates slow disk response time, you need to determine the underlying cause. As a first step, separate the disk response time value recorded in the Avg. Disk secs/transfer counter into average service time and average queue time. Table 3-24 describes the Physical Disk(*n*)\% Idle Time counter.

Table 3-24 Physical Disk(*n*)\% Idle Time Counter

Counter Type	Interval (% Busy).
Description	% of time that the disk was idle during the interval. Subtract % Idle Time from 100 percent to calculate disk utilization.
Measurement Notes	Idle Time accumulates whenever there are no requests outstanding for the device.

Table 3-24 **Physical Disk(*n*)\% Idle Time Counter**

Usage Notes	% Idle Time is the additive reciprocal (1–x) of disk utilization. ■ Derive disk utilization as follows: *Physical Disk(n)\Disk utilization = 100% – Physical Disk(n)\% Idle Time* ■ For disk arrays, divide disk utilization by the number disks in the array to estimate individual disk busy. Note, however, that additional I/Os might be occurring on the disks that are invisible to the operating system and that cause disks in redundant arrays to be busier than this estimated value. RAID subsystems require additional I/Os to maintain redundant data. If cached disks use Lazy Write to defer writes to disk, these writes to disk still take place, but only at some later time. ■ Queue time can be expected to increase exponentially as disk utilization approaches 100 percent, assuming independent arrivals to the disk. Derive disk queue time as follows: *Physical Disk(n)\Disk service time =* *Physical Disk(n)\Disk utilization ÷ Physical Disk(n)\Disk Transfers/sec* *Physical Disk(n)\Disk queue time = Physical Disk(n)\Avg. Disk sec/Transfer – Physical Disk(n)\Disk service time* ■ Apply an appropriate optimization strategy to improve disk performance, depending on whether the problem is excessive service time or queue time delays. See Chapter 5, "Performance Troubleshooting," for more details.
Performance	Primary indicator to determine whether a physical disk is overloaded and serving as a potential bottleneck.
Capacity Planning	Not applicable.
Operations	Increased queue time contributes to poor disk response time, which slows application response time.
Alert Threshold	Alert when % Idle Time is < 20 percent.
Related Measures	Physical Disk(*n*)\Avg. Disk secs/Transfer Physical Disk(*n*)\Disk Transfers/sec Physical Disk(*n*)\Current Disk Queue Length

Calculate disk utilization, disk service time, and disk queue time to determine whether you have a poor performing disk subsystem, an overloaded disk, or both. If disk I/O rates are high, you should also reconsider how effectively your workload is utilizing memory-resident cache to reduce the number of disk I/Os that reach the physical disk. These and other disk optimization strategies are discussed in more depth in Chapter 5, "Performance Troubleshooting." Table 3-25 describes the Physical Disk(*n*)\Disk Transfers/sec counter.

Table 3-25 Physical Disk(*n*)\Disk Transfers/sec Counter

Counter Type	Interval difference counter (rate/second).
Description	The rate physical disk requests were completed over the interval.
Measurement Notes	The start and end time of each I/O Request Packet (IRP) is recorded by the I/O Manager instrumentation layer. This counter reflects the number of requests that completed during the interval.
Usage Notes	The primary indicator of physical disk I/O activity. Also known as the disk arrival rate. ■ Also broken down by Reads and Writes: *Physical Disk(n)\Disk Transfers/sec = Physical Disk(n)\Disk Reads/sec + Physical Disk(n)\Disk Writes/sec* ■ For Disk arrays, divide Disk Transfers/sec by the number of disks in the array to estimate individual disk I/O rates. Note, however, that additional I/Os might be occurring on the disks that are invisible to the operating system that cause disks in redundant arrays to be busier than this estimated value. In a RAID 1 or RAID 5 organization, additional I/Os are required to maintain redundant data segments. ■ Used to calculate disk service time from % Idle Time by applying the Utilization Law.
Performance	Primary indicator to determine whether the disk is a potential bottleneck.
Capacity Planning	Not applicable.
Operations	Poor disk response time slows application response time.
Alert Threshold	Depends on the underlying disk hardware.
Related Measures	Physical Disk(*n*)\Disk Transfers/sec Physical Disk(*n*)\% Idle Time Physical Disk(*n*)\Current Disk Queue Length

Physical disk hardware can perform only one I/O operation at a time, so the number of physical disks attached to your computer serves as an upper bound on the sustainable disk I/O rate. Table 3-26 describes the Physical Disk(*n*)\Current Disk Queue Length counter.

Table 3-26 Physical Disk(*n*)\Current Disk Queue Length Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The current number of physical disk requests that are either in service or are waiting for service at the disk.
Measurement Notes	The start and end time of each I/O Request Packet (IRP) is recorded by the I/O Manager instrumentation layer. This counter reflects the number of requests that are outstanding at the end of the measurement interval.

Table 3-26 Physical Disk(*n*)\Current Disk Queue Length Counter

Usage Notes	A secondary indicator of physical disk I/O queuing. ■ Current Disk Queue Length is systematically under-sampled because Interrupt processing, which reduces the length of the disk request queue, runs at a higher dispatching priority than the software that gathers the disk performance measurements from the PerfDisk.dll Performance Library. ■ Useful to correlate this measured value with derived values like the Avg. Disk Queue Time, which you can calculate, and the Avg. Disk Queue Length counter to verify that disk queuing is a significant problem. ■ Values of the Current Disk Queue Length counter should be interpreted based on an understanding of the nature of the underlying physical disk entity. What appears to the host operating system as a single physical disk might, in fact, be a collection of physical disks that appear as a single LUN. Array controllers are often used to create Virtual LUNs that are backed by multiple physical disks. With array controllers, multiple disks in the array can be performing concurrent operations. Under these circumstances, the Physical Disk entity should then no longer be viewed as a single server. ■ If multiple disks are in the underlying physical disk entity, calculate the Current Disk Queue Length per physical disk.
Performance	Secondary indicator to determine whether the disk is a potential bottleneck.
Capacity Planning	Not applicable.
Operations	Poor disk response time slows application response time.
Alert Threshold	Alert when Current Disk Queue Length exceeds 5 requests per disk.
Related Measures	Physical Disk(*n*)\% Idle Time Physical Disk(*n*)\Avg. Disk secs/Transfer Physical Disk(*n*)\Avg. Disk Queue Length

Because disk I/O interrupt processing has priority over System Monitor measurement threads, the Current Disk Queue Length counter probably underestimates the extent that disk queuing is occurring. It is useful, nevertheless, to confirm the extent that disk queuing is occurring. Many I/O workloads are bursty, so you should not become alarmed when you see nonzero values of the Current Disk Queue Length from time to time. However, when the Current Disk Queue Length is greater than zero for sustained intervals, the disk is overloaded.

Derived Disk Measurements

The Logical and Physical Disk counters include several counters derived from the direct disk performance measurements that are prone to misinterpretation. These counters include % Disk Read Time, % Disk Write Time, % Disk Time, Avg. Disk Read Queue Length, Avg. Disk Write Queue Length, and Avg. Disk Queue Length. All of these derived counters need to be interpreted very carefully to avoid confusion.

Caution Unlike the Physical Disk\% Idle Time counter, the % Disk Read Time, % Disk Write Time, or % Disk Time counters do not attempt to report disk utilization. Whereas % Idle Time is measured directly by the I/O Manager instrumentation layer, % Disk Read Time, % Disk Write Time, and % Disk Time are derived from basic measurements using a formula based on Little's Law. The application of Little's Law might not be valid at that moment for your disk configuration.

The Avg. Disk Read Queue Length, Avg. Disk Write Queue Length, and Avg. Disk Queue Length measurements are based on a similar formula. These derived counters attempt to calculate the average number of outstanding requests to the Physical (or Logical) Disk over the measurement interval using Little's Law. However, Little's Law might not be valid over very small measurement intervals or intervals in which the disk I/O is quite bursty.

These derived disk performance counters should be relied on only if you have a good understanding of the underlying problems of interpretations.

As an alternative, you can always rely on the disk counters that are based on direct measurements. These include the % Idle Time, Disk Transfers/sec, Avg. Disk secs/Transfer, and Current Disk Queue Length counters.

Interpretation of the % Disk Time and Avg. Disk Queue Length counters is also difficult when the underlying physical disk entity contains multiple disks. What the host operating system regards as a physical disk entity might, in fact, be a collection of physical disks—or portions of physical disks—that are configured using an array controller to appear as a single LUN. If the underlying physical disk entity contains multiple disks capable of performing disk operations in parallel—a function common to most array controllers—the physical disk entity should not be viewed as a single server. Under these circumstances, the measured values for the Current Disk Queue Length and derived values of the Avg. Disk Queue Length reflect a single queue serviced by multiple disks. If multiple disks are in the physical disk entity, you should calculate the average queue length *per disk*. Table 3-27 describes the Physical Disk(*n*)\Avg. Disk Queue Length counter.

Table 3-27 Physical Disk(*n*)\Avg. Disk Queue Length Counter

Counter Type	Compound counter.
Description	The estimated average number of physical disk requests that are either in service or are waiting for service at the disk.

Table 3-27 **Physical Disk(*n*)\Avg. Disk Queue Length Counter**

Measurement Notes	Avg. Disk Queue Length is derived using Little's Law by multiplying Physical Disk(*n*)\Avg. Disk secs/Transfer by Physical Disk(*n*)\Disk Transfers/sec. This counter estimates the average number of requests that are in service or queued during the measurement interval.
Usage Notes	A secondary indicator of physical disk I/O queuing that requires careful interpretation. ■ For very short measurement intervals or for intervals in which the I/O activity is quite bursty, very high values of the Avg. Disk Queue Length should be interpreted cautiously. The use of Little's Law to derive the average disk queue length might not be valid for those measurement intervals. ■ Little's Law requires the equilibrium assumption that the number of I/O arrivals equals completion during the interval. For short measurement intervals, compare the Current Disk Queue Length to the value observed at the end of the previous measurement interval. If the values are significantly different, the use of Little's Law to estimate the queue length during the interval is suspect. ■ Correlate this derived value with measured values of the Current Disk Queue Length for the same measurement intervals. ■ Avg. Disk Read Queue Length and Avg. Disk Write Queue Length are derived similarly: *Physical Disk(n)\Avg. Disk Read Queue Length = Physical Disk(n)\Avg. Disk secs/Read × Physical Disk(n)\Disk Reads/sec* *Physical Disk(n)\Avg. Disk Write Queue Length = Physical Disk(n)\Avg. Disk secs/Write × Physical Disk(n)\Disk Write /sec* Interpretation of these values is subject to the same warnings listed in the Caution earlier in this section. ■ Values of the Avg. Disk Queue Length counter should be interpreted based on an understanding of the nature of the underlying physical disk entity. What appears to the host operating system as a single physical disk might, in fact, be a collection of physical disks that appear as a single LUN. Array controllers are often used to create Virtual LUNs that are backed by multiple physical disks. With array controllers, multiple disks in the array can be performing concurrent operations. Under these circumstances, the physical disk entity should no longer be viewed as a single server. ■ If multiple disks are in the underlying physical disk entity, calculate the Avg. Disk Queue Length per physical disk. ■ % Disk Read Time, % Disk Time, and % Disk Write Time are derived using the same formulas, except that the values they report are capped at 100 percent.

Table 3-27 Physical Disk(*n*)\Avg. Disk Queue Length Counter

Performance	Secondary indicator to determine whether the disk is a potential bottleneck.
Capacity Planning	Not applicable.
Operations	Not applicable.
Alert Threshold	Do not Alert on this counter value.
Related Measures	Physical Disk(*n*)\% Idle Time Physical Disk(*n*)\Avg. Disk secs/Transfer Physical Disk(*n*)\Disk Transfers/sec Physical Disk(*n*)\Current Disk Queue Length Physical Disk(*n*)\% Disk Time

The Explain text for the % Disk Time counters is misleading. These counters do not measure disk utilization or how busy the disk is, as the Explain text seems to imply. Use the % Idle Time measurement instead to derive a valid measure of disk utilization, as described earlier.

The % Disk Read Time, % Disk Time, and % Disk Write Time counters are derived using the same application of Little's Law formula, except that the values are reported as percentages and the percentage value is capped at 100 percent. For example, if the Avg. Disk Queue Length value is 0.8, the % Disk Time Counter reports 80 percent. If the Avg. Disk Queue Length value is greater than 1, % Disk Time remains at 100 percent.

Because the % Disk Time counters are derived using Little's Law, similar to the way the Avg. Disk Queue Length counters are derived, they are subject to the same interpretation issues. Table 3-28 describes the Physical Disk(*n*)\% Disk Time counter.

Table 3-28 Physical Disk(*n*)\% Disk Time Counter

Counter Type	Compound counter.
Description	The average number of physical disk requests that are either in service or are waiting for service at the disk, expressed as a percentage.
Measurement Notes	% Disk Time is derived using Little's Law by multiplying Physical Disk(*n*)\Avg. Disk secs/Transfer by Physical Disk(*n*)\Disk Transfers/sec. The calculation is then reported as a percentage and capped at 100 percent.

Table 3-28 Physical Disk(*n*)\% Disk Time Counter

Usage Notes	This derived value should be used cautiously, if at all. ■ This counter duplicates the Avg. Disk Queue Length calculation, which estimates the average number of requests that are in service or queued during the measurement interval. % Disk Time reports the same value, as a percentage, as the Avg. Disk Queue Length for disks with an average queue length <= 1. For disks with a calculated average queue length > 1, % Disk Time always reports 100 percent. ■ % Disk Read Time, % Disk Time, and % Disk Write Time are derived using the same formulas as the Avg. Disk Queue Length counters, except they report values as percentages and the values are capped at 100 percent. ■ % Disk Read Time and % Disk Write Time are derived using similar formulas: *Physical Disk(n)\% Disk Read Time = 100 × min(1,(Physical Disk(n)\Avg. Disk secs/Read × Physical Disk(n)\Disk Reads/sec))* *Physical Disk(n)\% Disk Write Time = 100 × min(1,(Physical Disk(n)\Avg. Disk secs/Write × Physical Disk(n)\Disk Write/sec)).* Interpretation of these values is subject to the same Warning. ■ Values of the % Disk Time counters should also be interpreted based on an understanding of the nature of the underlying physical disk entity. What appears to the host operating system as a single physical disk might, in fact, be a collection of physical disks that appear as a single LUN. Array controllers are often used to create virtual LUNs that are backed by multiple physical disks. With array controllers, multiple disks in the array can be performing concurrent operations. Under these circumstances, the physical disk entity should no longer be viewed as a single server. ■ If multiple disks are in the underlying physical disk entity, calculate the % Disk Time per physical disk.
Performance	Not applicable. Use the % Idle Time and Avg. Disk Queue Length counters instead.
Capacity Planning	Not applicable.
Operations	Not applicable.
Alert Threshold	Do not Alert on this counter value. Use the % Idle Time and Avg. Disk Queue Length counters instead.
Related Measures	Physical Disk(*n*)\% Idle Time Physical Disk(*n*)\Avg. Disk secs/Transfer Physical Disk(*n*)\Disk Transfers/sec Physical Disk(*n*)\Current Disk Queue Length Physical Disk(*n*)\Avg. Disk Queue Length

Split I/Os

Split I/Os are physical disk requests that are split into multiple requests, usually due to disk fragmentation. The Physical Disk object reports the rate that physical disk I/Os are split into multiple physical disk requests so that you can easily determine when disk performance is suffering because of excessive file system fragmentation. Table 3-29 describes the Physical Disk(*n*)\Split IO/sec counter.

Table 3-29 Physical Disk(*n*)\Split IO/sec Counter

Counter Type	Interval difference counter (rate/second).
Description	The rate physical disk requests were split into multiple disk requests during the interval. Note that when a split I/O occurs, the I/O Manager measurement layers count both the original I/O request and the split I/O request as split I/Os, so the split I/O count accurately reflects the number of I/O operations initiated by the I/O Manager.
Usage Notes	A primary indicator of physical disk fragmentation. ■ Defragmenting disks on a regular basis helps improve disk performance because sequential operations run several times faster than random disk requests on most disks. On disks with built-in actuator-level buffers, sequential operations can run 10 times faster than random disk requests. ■ A split I/O might also result when data is requested in a size that is too large to fit into a single I/O. ■ Calculate split I/Os as a percentage of Disk Transfers/sec: *Physical Disk(n)\% Split IOs = Physical Disk(n)\Split IO/sec ÷ Physical Disk(n)\Disk Transfers/sec* When the number of split I/Os is 10–20 percent or more of the total Disk Transfers, check to see whether the disk is very fragmented. ■ Split I/Os usually take longer for the disk to service, so also watch for a correlation with Physical Disk(*n*)\Avg. Disk secs/Transfer. Higher values of Avg. Disk secs/Transfer also contribute to greater disk utilization (1–% Idle Time).
Performance	Secondary indicator that helps you determine how often you need to run disk defragmentation software.
Capacity Planning	Not applicable.
Operations	Poor disk response time slows application response time.
Alert Threshold	Alert when split I/Os > 20 percent of Disk Transfers/sec.
Related Measures	Physical Disk(*n*)\Disk Transfers/sec Physical Disk(*n*)\Avg. Disk secs/Transfer Physical Disk(*n*)\% Idle Time

Defragmenting disks on a regular basis or when the number of split I/Os is excessive will normally improve disk performance, because disks are capable of processing sequential operations much faster than they process random requests. Be sure to check the Analysis Report produced by the Defragmentation utility. If the report indicates that the files showing the most fragmentation are the ones in constant use or the ones being modified regularly, the performance boost gained from defragmenting the disk might be short-lived. For more advice about using disk defragmentation utilities effectively, see Chapter 5, "Performance Troubleshooting."

Disk Space Usage Measurements

Two important disk space usage measurements are available in the Logical Disk object: % Free Space and Free Megabytes. Because running out of disk space is almost always catastrophic, monitoring the Free Space available on your logical disks is critical. Because disk space tends to be consumed gradually, you usually don't have to monitor disk free space as frequently as you do for many of the other performance indicators you will gather. Monitoring disk free space hourly or even daily is usually sufficient. Note that the Performance Library responsible for computing the Free Megabytes and % Free Space counters, Perfdisk.dll, refreshes these measurements at a slower pace because the counter values themselves tend to change slowly. By default, these disk space measurements are refreshed once every 5 minutes, independent of the Performance Monitor data collection interval. Using Performance Monitor to gather these counters at a rate faster than approximately 5 minutes per sample, you will retrieve counter values that are duplicates, reflecting the slow rate of data gathering by the Perfdisk.dll Performance Library. Both the high overhead associated with calculating Free Megabytes on very large file systems and the normally slow rate at which these counter values change are the important factors that this slower rate of data gathering reflects. Table 3-30 describes the Logical Disk(*n*)\Free Megabytes counter.

Note Because of the high overhead associated with calculating % Free Space and Free Megabytes on very large file systems, and the normally slow rate at which these counter values change, these counters are normally measured only once every 5 minutes. If you need more frequent measurements to track file system growth, you can add a binary Registry field at HKLM\System\CurrentControlSet\Services\Perfdisk\Performance\VolumeRefreshInterval and change the VolumeRefreshInterval to a more suitable value. Code the number of seconds you would like to wait between recalculations of the Logical Disk % Free Space and Free Megabytes metrics. The default VolumeRefreshInterval is 300 seconds.

Table 3-30 Logical Disk(*n*)\Free Megabytes Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The amount of unallocated space on the logical disk, reported in megabytes.
Measurement Notes	This is the same value reported by Windows Explorer on the Logical Disk Property sheets. ■ This counter value tends to change slowly. ■ Because calculating free megabytes for very large file systems is time-consuming, the I/O Manager measurement layers recalculate the value of the counter approximately once every 5 minutes. When you gather this measurement data more frequently than the rate at which it is recalculated, you will obtain static values of the counter that reflect this slow rate of updating.
Usage Notes	A primary indicator of logical disk space capacity used. ■ Divide by % Free Space or multiply by the reciprocal of free space (1/% Free Space).
Performance	Not applicable.
Capacity Planning	Trending and forecasting disk space usage over time.
Operations	Running out of space on the file system is usually catastrophic.
Alert Threshold	Alert on this counter value or when Logical Disk(*n*)\% Free Space < 10 percent.
Related Measures	Logical Disk(*n*)\% Free Space.

You can use statistical forecasting techniques to extrapolate from the historical trend of disk space usage to anticipate when you are likely to run out of disk space. See Chapter 4, "Performance Monitoring Procedures," for an example that uses linear regression to forecast workload growth—a simple statistical technique that can readily be adapted for long-term disk capacity planning.

Managing Network Traffic

Network traffic is instrumented at the lowest level hardware interface and at each higher level in the TCP/IP stack. At the lowest level, both packets and byte counts are accumulated. At the IP level, datagrams sent and received are counted. Statistics for both IP version 4 and IP version 6 are provided. At the TCP level, counters exist for segments sent and received, and for the number of initiated and closed connections. At the network application level, similar measures of load and traffic are in, for example, HTTP requests, FTP requests, file Server requests, network client Redirector requests, as well other application-level statistics. In some cases, application response time measures might also be available in some form.

Network Interface Measurements

At the lowest level of the TCP/IP stack, the network interface driver software layer provides instrumentation on networking hardware performance. Network interface statistics are gathered by software embedded in the network interface driver layer. This software counts the number of packets that are sent and received, and also tracks the size of their data payloads. Multiple instances of the Network Interface object are generated, one for every network interface chip or card that is installed, plus the Loopback interface, if that is defined. Note that network packets that were retransmitted as a result of collisions on an Ethernet segment are not directly visible to the host software measurement layer. Ethernet packet collision detection and recovery is performed on board the network adapter card, transparently to all host networking software. Table 3-31 describes the Network Interface(*n*)\Bytes Total/sec counter.

Table 3-31 Network Interface(*n*)\Bytes Total/sec Counter

Counter Type	Interval difference counter (rate/second).
Description	Total bytes per second transmitted and received over this interface during the interval. This is the throughput (in bytes) across this interface.
Measurement Notes	This counter tracks packets sent and received and accumulates byte counts from packet headers as they are transmitted or received. Packets are retransmitted on an Ethernet segment because collisions are not included in this count.
Usage Notes	The primary indicator of network interface traffic. ■ Calculate network interface utilization: *Network Interface(n)\% Busy = Network Interface(n)\Bytes Total/sec ÷ Network Interface(n)\Current Bandwidth* ■ Network packets that were retransmitted as a result of collisions on an Ethernet segment are not counted. Collision detection and recovery is entirely performed on board the NIC, transparently to the host networking software. ■ The Current Bandwidth counter reflects the actual performance level of the network adaptor, not its rated capacity. If a gigabit network adapter card on a segment is forced to revert to a lower speed, the Current Bandwidth counter will reflect the shift from 1 Gbps to 100 Mbps, for example. ■ The maximum achievable bandwidth on a switched link should be close to 90–95 percent of the Current Bandwidth counter.

Table 3-31 Network Interface(*n*)\Bytes Total/sec Counter

Performance	Primary indicator to determine whether the network is a potential bottleneck.
Capacity Planning	Trending and forecasting network usage over time.
Alert Threshold	Alert when Total Bytes/sec exceeds 90 percent of line capacity.
Related Measures	Network Interface(*n*)\Bytes Received/sec Network Interface(*n*)\Bytes Sent/sec Network Interface(*n*)\Packets Received/sec Network Interface(*n*)\Packets Sent/sec Network Interface(*n*)\Current Bandwidth

TCP/IP Measurements

Both the IP and TCP layers of the TCP/IP stack are instrumented. Windows Server 2003 supports both TCP/IP version 4 and version 6, and there are separate performance objects for each, depending on which versions of the software are active. At the IP level, Datagrams/sec is the most important indicator of network activity, which can also be broken out into Datagrams Received/sec and Datagrams Sent/sec. Additional IP statistics are available on packet fragmentation and reassembly. Table 3-32 describes the IPv*n*\Datagrams/sec counter.

Table 3-32 IPv*n*\Datagrams/sec Counter

Counter Type	Interval difference counter (rate/second).
Description	Total IP datagrams per second transmitted and received during the interval.
Measurement Notes	The IP layer of the TCP/IP stack counts datagrams sent and received.
Usage Notes	The primary indicator of IP traffic. Identical sets of counters are available for IPv4 and IPv6.
Performance	Secondary indicator to determine whether the network is a potential bottleneck.
Capacity Planning	Trending and forecasting network usage over time.
Operations	Sudden spikes in the amount of IP traffic might indicate the presence of an intruder.
Alert Threshold	Build alerts for important machines linked to the network backbone based on extreme deviation from historical norms.
Related Measures	IPv*n*\Datagrams Received/sec IPv*n*\Datagrams Sent/sec Network Interface(*n*)\Packets/sec

For the TCP protocol, which is session- and connection-oriented, connection statistics are also available. It is useful to monitor and track TCP connections both for security reasons to detect Denial of Service attacks, and for capacity planning. Table 3-33 describes the TCPv*n*\Connections Established counter.

Table 3-33 TCPv*n*\Connections Established Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The total number of TCP connections in the ESTABLISHED state at the end of the measurement interval.
Measurement Notes	The TCP layer counts the number of times a new TCP connection is established.
Usage Notes	The primary indicator of TCP session connection behavior. Identical counters are maintained for TCPv4 and TCPv6. The number of TCP connections that can be established is constrained by the size of the Nonpaged pool. When the Nonpaged pool is depleted, no new connections can be established.
Performance	Secondary indicator to determine whether the network is a potential bottleneck.
Capacity Planning	Trending and forecasting growth in the number of network users over time. The administrator should tune TCP registry entries like *MaxHashTableSize* and *NumTcTablePartitions* based on the number of network users seen on average.
Operations	Sudden spikes in the number of TCP connections might indicate a Denial of Service attack.
Alert Threshold	Build alerts for important machines linked to the network backbone based on extreme deviation from historical norms.
Related Measures	TCPPv*n*\Segments Received/sec TCPPv*n*\Segments Sent/sec Network Interface(*n*)\Packets/sec Memory\Nonpaged Pool Bytes

From a capacity planning perspective, TCP Connections Established measures the number of network clients connected to the server. Additionally, characterizing each session by its workload demand is useful. TCP activity is recorded by segments, which then get broken into packets by the IP layer that are compatible with the underlying hardware. Segments Received/sec corresponds to the overall request rate from networking clients to your server. Table 3-34 describes the counter.

Table 3-34 TCPv*n*\Segments Received/sec Counter

Counter Type	Interval difference counter (rate/second).
Description	The number of TCP segments received across established connections, averaged over the measurement interval.
Measurement Notes	The TCP layer counts the number of times TCP segments are received.
Usage Notes	The primary indicator of TCP network load. ■ Identical counters are maintained for TCPv4 and TCPv6. ■ Calculate the average number of segments received per connection: *TCPvn\Segments Received/sec ÷ TCPPvn\Connections Established/sec* This can be used to forecast future load as the number of users grows. ■ When server request packets are received from network clients, depending on the networking application, the request is usually small enough to fit into a single Ethernet message and IP Datagram. For HTTP and server message block (SMB) requests, for example, TCPv*n*\Segments Received/sec≅IPv*n*\Datagrams Received/sec because HTTP and SMB requests are usually small enough to fit into a single packet.
Performance	Secondary indicator to determine whether the network is a potential bottleneck.
Capacity Planning	Trending and forecasting network usage over time.
Operations	Sudden spikes in the amount of TCP requests received might indicate the presence of an intruder.
Alert Threshold	Build alerts for important machines linked to the network backbone based on extreme deviation from historical norms.
Related Measures	TCPPv*n*\Connections Established/sec TCPPv*n*\Segments Sent/sec IPv*n*\Datagrams Received/sec Network Interface(*n*)\Packets/sec

If Windows Server 2003 machines are serving as networking hubs or gateways, IPv*n*\Datagrams Received/sec and TCPv*n*\Segments Received track the number of requests received from networking clients. For capacity planning, either of these indicators of load can be used to characterize your machine's workload in terms of network traffic per user. When you are running machines dedicated to a single server application—a dedicated IIS Web server, for example—you can also characterize processor usage per user connection, along with processor usage per request. Then as the number of users increases in your forecast, you can also project the network and processor resources that are required to service that projected load. Because of the extensive use of memory-resident caching in most server applications, characterizing the disk I/O rate per user or request isn't easy, because the disk I/O rate could remain relatively flat as the request load increases because of effective caching.

Networking Error Conditions

In all areas of network monitoring, pay close attention to any reported error incidents. These include Network Interface(*n*)\Packets Received Errors, Network Interface(*n*)\Packets Outbound Errors, IP\Datagrams Outbound No Route, IP\Datagrams Received Address Errors, IP\Datagrams Received Discarded, TCP\Segments Retransmitted/sec and TCP\Connection Failures. Configure alerts to fire when any of these networking error conditions are observed.

Maintaining Server Applications

When the TCP layer is finished processing a segment received from a networking client, the request is passed upwards to the networking application that is plugged into the associated TCP port. Some of the networking applications you will be managing might include:

- File Server
- Print server
- Web server
- SQL Server
- Terminal Server
- Exchange Mail and Messaging server
- COM+ Server applications

These and other networking applications provide additional performance measurements that can be used to characterize their behavior. This section highlights some of the most important performance counters available from these networking applications that aid in their configuration and tuning.

Thread Pooling

To aid in scalability, most server applications use some form of *thread pooling*. In general, thread pooling means these server applications perform the following actions:

- Define a pool that contains Worker threads that can handle incoming requests for service.
- Queue work requests as they arrive, and then release and dispatch Worker threads to take a work request off the queue and complete it.

The maximum size of the thread pool defined by the server application is usually a function of the size of RAM and the number of processors. There will be many instances when the heuristic used to set the maximum size of thread pool is inadequate for your specific workload. When that happens, these server applications frequently also have configuration and tuning options that allow you to do either of the following actions, or both:

- Increase the number of worker threads in the thread pool
- Boost the dispatching priority of worker threads in the thread pool

Looking at these thread pooling server applications externally, from the point of view of their process address space, Process(*n*)\Thread Count identifies the total number of threads that are created, including the worker threads created in the thread pool. You will frequently find that the Process(*n*)\Thread Count is a large, relatively static number. However, inside the application, the situation is much more dynamic. Worker threads are activated as work requests are received, up to the maximum size of the thread pool, and they might be decommissioned later after they are idle.

Performance counters that are internal to the server application let you know how many of the worker threads that are defined are in use. If you find that the following two conditions are true for these thread pooling applications, you might find that increasing the maximum size of the thread pool will boost throughput and improve the responsiveness of the server application:

- Running worker threads for sustained periods of time at or near the limit of the maximum size of the pool
- Neither the processor (or processors) or memory appears to be saturated

In addition to monitoring the number of active worker threads, you might also find it useful to track the client request rate and any counters that indicate when internal requests are delayed in the request queue. For more information about thread pooling in server applications, see Chapter 6, "Advanced Performance Topics."

You might also find that increasing the maximum size of the thread pool makes no improvement, merely resulting in higher context switch rates and increased CPU utilization (with no increase in throughput). In this case, you should reduce the maximum size of the thread pool to its original size (or to an even lower value).

File Server

The file Server service that is available on all Windows Server 2003 machines is a good example of a thread pooling application. It defines separate thread pools for each processor, and then uses processor affinity to limit the amount of interprocessor communication that occurs. The file Server also provides extensive instrumentation that is available in two performance objects: Server and Server Work Queues. You can monitor the number of active worker threads in each Server Work Queue, the rate of requests that are processed, the request queue length, and a number of error indicators, along with other measurements.

The Server object contains overall file server statistics, including the number of file server sessions, the rate of requests, and several error indicators. The most important error indicator is the Server\Work Item Shortages counter, which tracks the number of times a client request was rejected because of a resource shortage.

Table 3-35 Server\Work Item Shortages Counter

Counter Type	Interval difference counter (rate/second).
Description	The number of times a shortage of work items caused the file Server to reject a client request. This error usually results in session termination.
Measurement Notes	The file Server counts the number of times a work item shortage occurs.

Table 3-35 **Server\Work Item Shortages Counter**

Usage Notes	A primary indicator that the File Server service is short of resources. ■ As server message blocks (SMBs) are received from clients by the Server service, the request is stored in a work item and assigned to an available worker thread from the thread pool to process. If worker threads are not available or cannot process requests fast enough, the queue of Available Work Items can become depleted. When no more work items are available, the Server service cannot process the SMB request and terminates the session with an error response. ■ If memory is available, add the values for the *InitWorkItems* or *MaxWorkItems* parameters to the Registry key at HKLM\SYSTEM\CurrentControlSet\Services\lanmanserver\parameters to increase the number of work items that are allocated.
Performance	Primary indicator to determine whether the number of Server Work items defined is a potential bottleneck.
Capacity Planning	Not applicable.
Operations	File Server clients whose sessions are terminated because of a work item shortage must restore their session manually.
Alert Threshold	Alert on any nonzero value of this counter.
Related Measures	Server Work Queues(*n*)\Active Threads Server Work Queues(*n*)\Available Threads Server Work Queues(*n*)\Available Work Items Server Work Queues(*n*)\Borrowed Work Items Server Work Queues(*n*)\Queue Length Server Work Queues(*n*)\Work Item Shortages.

At the Server Work Queue level, you can drill down into more detail. One Server Work Queue is defined for blocking requests, which are operations that require an I/O to disk. Server implements an I/O completion port for worker threads that handle these operations. Besides the Blocking Queue is one Server Work Queue defined per processor, each with dedicated thread pool. The per processor Server Work Queues are identified using the processor instance name. SMBs received from network clients are assigned to the Server Work Queue associated with the processor where the network interface interrupt was serviced. Threads from the thread pool are activated as necessary to process incoming requests for service.

The file Server is well-instrumented. There are counters at the Server Work Queue level for the number of Active Threads currently engaged in processing SMB requests and Available Threads that could be scheduled if new work requests arrive. You should verify that Available Threads is not equal to zero for any sustained period that might cause the Work Item queue to back up. Table 3-36 describes the Server Work Queues(*n*)\Available Threads counter.

Table 3-36 Server Work Queues(*n*)\Available Threads Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	A sampled value that reports the number of available threads from the per-processor Server Work Queue that are available to process incoming SMB requests.
Measurement Notes	The file Server counts the number of free worker threads.
Usage Notes	A primary indicator that the file Server service is short of worker threads. ■ When Available Threads is zero, incoming SMB requests must be queued. If requests arrive faster than they can be processed because there are no Available Threads, the queue where pending work items are stored might back up. ■ If there are no Available Work Items, the server attempts to borrow them from another processor Work Item queue. Borrowing work items forces the Server to lock the per-processor Work Item queue to facilitate interprocessor communication, which tends to slow down work item processing. ■ If Available Threads is at or near zero for any sustained period, the Queue Length of waiting requests is > 5, and processor resources are available—% Processor Time for the associated processor instance < 80 percent—you should add the value for *MaxThreadsPerQueue* to the Registry key at HKLM\SYSTEM\CurrentControlSet\Services\lanmanserver\parameters to increase the number of threads that are created in the per-processor thread pools. ■ Per processor thread pools are defined so that there are multiple instances of the Server Work queues performance object on a multiprocessor. ■ The values of the Active Threads and Available Threads counters for the Blocking queue can be safely ignored because the Blocking queue is managed differently from the per-processor thread pools. Resources to process Blocking queue requests are allocated on demand. The Server service also utilizes the I/O completion port facility to process I/O requests more efficiently.
Performance	Primary indicator to determine whether the number of worker threads defined for the per-processor Server Work queues is a potential bottleneck.
Capacity Planning	Not applicable.

Table 3-36 Server Work Queues(*n*)\Available Threads Counter

Operations	File Server clients whose sessions are terminated because of a work item shortage must restore their session manually.
Alert Threshold	Alert on any zero value of this counter.
Related Measures	Server\Work Item Shortages Server Work Queues(*n*)\Active Threads Server Work Queues(*n*)\Available Work Items Server Work Queues(*n*)\Borrowed Work Items Server Work Queues(*n*)\Queue Length Server Work Queues(*n*)\Work Item Shortages

Each per-processor Server Work queue also reports on the number of queued requests that are waiting for a worker thread to become available. Table 3-37 describes the Server Work Queues(*n*)\Queue Length counter.

Table 3-37 Server Work Queues(*n*)\Queue Length Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	A sampled value that reports the number of incoming SMB requests that are queued for processing, waiting for a worker thread to become available. There are separate per-processor Server Work queues to minimize interprocessor communication delays.
Measurement Notes	The file Server reports the number of SMB requests stored in work items that are waiting to be assigned to an available worker thread for servicing.
Usage Notes	A secondary indicator that the file Server service is short of worker threads. ■ Pay close attention to the per-processor work item queues, and watch for indications that the queues are backing up. ■ Work items for the Blocking Queue are created on demand, so the Blocking queue is managed differently from the per-processor work queues. The Blocking queue is seldom a bottleneck.
Performance	Primary indicator to determine whether client SMB requests are delayed for processing at the file Server. A secondary indicator that the per-processor Work Item queue is backed up because of a shortage of threads or processing resources.
Capacity Planning	Not applicable.
Operations	File Server clients whose sessions are terminated because of a work item shortage must restore their session manually.
Alert Threshold	Alert when the Queue Length > 5 for any processor Work Item queue.
Related Measures	Server\Work Item Shortages Server Work Queues(*n*)\Active Threads Server Work Queues(*n*)\Available Threads Server Work Queues(*n*)\Available Work Items Server Work Queues(*n*)\Borrowed Work Items Server Work Queues(*n*)\Work Item Shortages

Print Servers

Printing is performed by worker threads associated with the Print Spooler service, which executes as the Spoolsv.exe process. A tuning parameter allows you to boost the priority of print spooler threads if you need to boost the performance of background printing services. The print spooler also supplies performance statistics in the Print Queue object. Key performance counters include both print jobs and printed pages. As you would for other server applications, set up alerts for error conditions. Any nonzero occurrences of the Print Queue\Not Ready Errors, Out of Paper Errors, and Job Errors that occur should generate alerts so that operations staff can intervene promptly to resolve the error conditions.

Web-Based Applications

The Web server and FTP server functions in IIS are also structured as thread pooling applications to assist in scalability. The IIS Web server contains many performance-oriented parameters and settings, a complete discussion of which is beyond the scope of this book.

More Info For more information about performance-oriented parameters and settings for IIS Web server, see the "Internet Information Services (IIS) 6.0" topic under "Internet and E-mail services" in the "Help and Support" documentation, and the "Internet Information Services (IIS) 6.0 Help" in the Internet Information Services Manager. Also see Chapter 13, "Optimizing IIS 6.0 Performance," in the *Microsoft Internet Information Services (IIS) 6.0 Resource Kit* from Microsoft Press.

IIS provides a number of performance objects, depending on which specific services are defined, each with corresponding measures that report on the transaction load. These objects and some of their most important measures of transaction load are identified in the Table 3-38.

Table 3-38 IIS Critical Measurements

Object	Counter	Notes
SMTP Server	Bytes Received/sec	The Exchange Server Internet Mail Connector (IMC) uses the IIS SMTP Server facility to communicate with other e-mail servers using the SMTP protocol.
	Bytes Sent/sec	
	Messages Received/sec	
	Messages Sent/sec	

Table 3-38 **IIS Critical Measurements**

Object	Counter	Notes
FTP Service	Bytes Received/sec	There is one instance of the FTP Service object per FTP site, plus an *_Total* instance.
	Bytes Sent/sec	
	Total Files Received	
	Total Files Sent	
Web Service	Bytes Received/sec	There is one instance of the Web Service object per Web site, plus an *_Total* instance.
	Bytes Sent/sec	
	Get Requests/sec	
	Post Requests/sec	
	ISAPI Extension Requests/sec	
NNTP Service	Articles Received/sec	
	Articles Sent/sec	
	Bytes Received/sec	
	Bytes Sent/sec	
Active Server Pages	Requests/sec	
ASP.NET	Requests/sec	
ASP.NET Applications	Requests/sec	There is one instance of the ASP.NET Applications object per ASP.NET application, plus an *_Total* instance.

IIS defines a pool of worker threads, which are then assigned dynamically to perform the various tasks requested from the different installed Web applications. ASP and ASP.NET applications also rely on additional thread pools. For example, ASP application thread pools are governed by the *AspProcessorThreadMax* property, which is stored in the metabase. In the case of ASP.NET applications, the *maxWorkerThreads* and *maxIOThreads* properties from the *processModel* section of the Machine.config file determine the size of the thread pool. Thread pool configuration and tuning for Web server and other server applications is discussed in Chapter 6, "Advanced Performance Topics."

Both Active Server Pages and ASP.NET provide many additional metrics, including some important response time indicators. Table 3-39 describes the Active Server Pages\Request Execution Time and ASP.NET\Request Execution Time counters.

Table 3-39 Active Server Pages\Request Execution Time and ASP.NET\Request Execution Time Counters

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The execution time in milliseconds of the ASP or ASP.NET transaction that completed last.
Measurement Notes	This is the same value that is available in the IIS log as the Time Taken field. It is the Time Taken for the last ASP Request that completed execution. If you gather this counter at a rate faster than the ASP Requests/sec transaction arrival rate, you will gather duplicate counter values.
Usage Notes	A primary indicator of ASP and ASP.NET application service time. Properly viewed as a sample measurement of ASP or ASP.NET transaction service time. Because it is a sampled value and it is impossible to tell what specific transaction was completed last, you should not use this counter for operational alerts. Calculate ASP application response time by adding Active Server Pages\Request Queue Time: *Active Server Pages\Request Response Time = Active Server Pages\Request Execution Time + Active Server Pages\ Request Queue Time* Calculate ASP.NET application response time by adding ASP.NET\Request Queue Time: *ASP.NET \Request Response Time = ASP.NET \Request Execution Time + ASP.NET \Request Queue Time*
Performance	A primary indicator of ASP or ASP.NET application service time.
Capacity Planning	Not applicable.
Operations	Not applicable.
Alert Threshold	Do not alert on this counter.
Related Measures	Active Server Pages\Requests/sec Active Server Pages\Request Queue Time Active Server Pages\Requests Executing Active Server Pages\Requests Queued ASP.NET\Requests/sec ASP.NET\Request Queue Time ASP.NET\Requests Executing ASP.NET\Requests Queued

Active Server Pages\Request Execution Time and ASP.NET\Request Execution Time are indicators of ASP and ASP.NET application service time, respectively. But both counters need to be treated as *sampled* measurements, or a single observation. You need, for example, to accumulate several hundred sample measurements during periods of peak load to be able to estimate the *average* Request Execution Time accurately over that interval. Table 3-40 describes the Active Server Pages\Request Queue Time and ASP.NET\Request Queue Time counters.

Table 3-40 Active Server Pages\Request Queue Time and ASP.NET\Request Queue Time Counters

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The queue time in milliseconds of the ASP or ASP.NET transaction that completed last.
Measurement Notes	This is the Queue Time delay value for the last ASP or ASP.NET request that completed execution. If you gather this counter at a rate faster than the ASP Requests/sec transaction arrival rate, you will gather duplicate counter values.
Usage Notes	A primary indicator of ASP or ASP.NET application queue time. Properly viewed as a sample measurement of ASP or ASP.NET transaction queue time. Because it is a sampled value and it is impossible to tell what specific transaction completed last, you should not use this counter for operational alerts. Calculate ASP.NET application response time by adding ASP.NET \Request Execution Time: *ASP.NET \Request Response Time = ASP.NET \Request Execution Time + ASP.NET \Request Queue Time*
Performance	A primary indicator of ASP or ASP.NET application queue time.
Capacity Planning	Not applicable.
Operations	Not applicable.
Alert Threshold	Do not alert on this counter.
Related Measures	Active Server Pages\Requests/sec Active Server Pages\Request Execution Time Active Server Pages\Requests Executing Active Server Pages\Requests Queued ASP.NET\Requests/sec ASP.NET \Request Execution Time ASP.NET\Requests Executing ASP.NET\Requests Queued

Adding ASP.NET Request Execution Time and Request Queue Time yields the response time of the last ASP.NET transaction. This derived value also needs to be treated as a sample measurement, or a single observation. In a spreadsheet, for example, accumulate several hundred sample measurements of Request Execution Time and Request Queue Time during periods of peak load, and then summarize them to estimate the average Request Response Time during the period. Table 3-41 describes the Active Server Pages\Requests Executing and ASP.NET\Requests Executing counters.

Table 3-41 Active Server Pages\Requests Executing and ASP.NET\Requests Executing Counters

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of ASP or ASP.NET requests that are currently being executed.
Usage Notes	A primary indicator of ASP application concurrency. Each active ASP or ASP.NET request is serviced by a worker thread. If all ASP or ASP.NET threads are currently busy when a new request arrives, the request is queued. This is an instantaneous counter that reports the number of ASP or ASP.NET threads currently occupied with active requests.
Performance	A primary indicator of ASP or ASP.NET application concurrency.
Capacity Planning	Not applicable.
Operations	Not applicable.
Alert Threshold	Alert when this ASP\Requests Executing ≥ (AspProcessorThreadMax – 2) × #_of _processors. Alert when this ASP.NET\Requests Executing ≥ (ProcessorThreadMax – 2) × #_of _processors.
Related Measures	Active Server Pages\Requests/sec Active Server Pages\Request Queue Time Active Server Pages\Requests Execution Time Active Server Pages\Requests Queued ASP.NET\Requests/sec ASP.NET\Request Queue Time ASP.NET \Requests Execution Time ASP.NET Requests Queued

The *ProcessorThreadMax* property is an upper limit on the number of ASP.NET Requests that can be in service at one time. If all available ASP.NET threads are currently busy when a new ASP.NET request arrives, the request is queued and must wait until one of the requests in service completes and an ASP.NET worker thread becomes available.

Tip The *ProcessorThreadMax* parameter can have a significant affect on the scalability of your ASP applications. When ASP.NET\Requests Executing is observed at or near *ProcessorThreadMax* multiplied by the number of processors, consider increasing the value of *ProcessorThreadMax*. For more details, see Chapter 6, "Advanced Performance Topics."

Consider, for example, an ASP.NET application that services 10 requests per second. If the average response of ASP.NET applications is 4 seconds, Little's Law predicts that the number of ASP.NET requests in the system equals the arrival rate multiplied by the response time, or 10 × 4, or 40, which is how many ASP.NET\Requests Executing you would expect to see. If *ProcessorThreadMax* is set to its default value, which is 20, and IIS is running on a 2-way multiprocessor, the maximum value you could expect to see for ASP.NET\Requests Executing is 40. Absent other obvious processor, memory, or disk resource constraints on ASP.NET application throughput, observing no more than 40 Requests Executing during periods of peak load might mean that the *ProcessorThreadMax* is an artificial constraint on ASP.NET throughput. For a more detailed discussion of server thread pooling applications and their scalability, see Chapter 6, "Advanced Performance Topics." Table 3-42 describes the Active Server Pages\Requests Queued and ASP.NET\Requests Queued counters.

Table 3-42 Active Server Pages\Requests Queued and ASP.NET\Requests Queued Counters

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of ASP or ASP.NET requests that are currently queued for execution, pending the availability of an ASP or ASP.NET worker thread.
Usage Notes	A primary indicator of ASP or ASP.NET concurrency constraint. Each active request is serviced by a worker thread. The number of worker threads available to process ASP requests is governed by the *AspProcessorThreadMax* property in the IIS metabase. *AspProcessorThreadMax* defaults to 25 threads per processor in IIS 6.0. The number of worker threads available to process ASP.NET requests is governed by the *ProcessorThreadMax* property in the Machine.config settings file. *ProcessorThreadMax* defaults to 20 threads per processor in .NET version 1. If all worker threads are currently busy when a new request arrives, the request is queued. This is an instantaneous counter that reports the number of ASP or ASP.NET threads currently occupied with active requests. Estimate average ASP Request response time using Little's Law: *Active Server Pages\Request response time = (Active Server Pages\Requests Executing + Active Server Pages\Requests Queued) ÷ Active Server Pages\Requests/sec* Estimate average ASP.NET request response time using Little's Law: *ASP.NET\Request response time = (ASP.NET\Requests Executing + ASP.NET\Requests Queued) ÷ ASP.NET\Requests/sec*

Table 3-42 Active Server Pages\Requests Queued and ASP.NET\Requests Queued Counters

Performance	A primary indicator of ASP and ASP.NET application concurrency.
Capacity Planning	Not applicable.
Operations	Not applicable.
Alert Threshold	Alert when this counter ≥ 5 × #_of _processors.
Related Measures	Active Server Pages\Requests/sec, Active Server Pages\Request Queue Time Active Server Pages\Requests Execution Time Active Server Pages\Requests Queued ASP.NET\Requests/sec ASP.NET\Request Queue Time ASP.NET\Requests Execution Time ASP.NET\Requests Queued

The ASP.NET counters (and their Active Server Pages corresponding counters) can also be used to estimate response time on an interval basis using Little's Law. Together, ASP.NET\Requests Executing and ASP.NET\Requests Queued measure the number of ASP.NET requests currently in the system at the end of the interval. Assuming the equilibrium assumption is not violated—that ASP.NET transaction arrivals roughly equal completions over the interval—then divide the number of requests in the system by the arrival rate, ASP.NET\Requests/sec, to calculate average ASP.NET application response time. This value should correlate reasonably well with the sampled ASP.NET response time that you can calculate by adding ASP.NET\Requests Execution Time and ASP.NET\Requests Queue Time over an interval in which you have accumulated sufficient samples.

Similar ASP.NET measurements are available at the .NET application level.

Tip Any time your Web site reports ASP.NET\Requests Executing at or near the *ProcessorThreadMax* × the number of processors maximum, consider boosting the number of ASP.NET threads.

You might find that increasing *ProcessorThreadMax* makes no improvement, merely resulting in higher context switch rates and increased CPU utilization (with no increase in throughput). In this case, reduce the number to its original size (or to an even lower value).

This is a simplified view of Web application thread pooling. With Web application gardens and ASP and ASP.NET applications running in isolated processes, the configuration and tuning of the Web server application thread pool grows more complicated. See Chapter 6, "Advanced Performance Topics," for more details.

Terminal Services

Terminal Services provides many unique capabilities. With Terminal Services, for example, a single point of installation of a desktop application can be made available centrally to multiple users. Users sitting at Terminal Server client machines can run programs remotely, save files, and use network resources just as though the applications were installed locally. Terminal Server can also deliver Windows desktop applications to computers that might not otherwise be able to run the Windows operating system.

Similar to other server applications, there might be many Terminal Server clients that depend on good performance from your Terminal Server machine or machines. Effective performance monitoring procedures are absolutely necessary to ensure good service is provided to Terminal Server clients.

When a Terminal Server client logs on to Windows Server 2003 configured to run Terminal services, it creates a Terminal Server *session*. Table 3-43 describes the Terminal Services\Total Sessions counter.

Table 3-43 Terminal Services\Total Sessions Counter

Counter Type	Instantaneous (sampled once during each measurement period).
Description	The total number of Terminal Server sessions, including both active and inactive sessions.
Usage Notes	The primary indicator of total Terminal Server clients. It includes both inactive and active sessions. Terminal Server creates and maintains a desktop environment for each active session. This requires private copies of the following processes: Explorer, Winlogon, Smss.exe, Lsass.exe, and Csrss.exe. Additional private copies of any processes that the Terminal Server launches from the desktop are also created. Private copies of the desktop processes are retained for duration of the session, regardless of whether the session is inactive or active.
Performance	Not applicable.
Capacity Planning	Trending and forecasting Terminal Server usage over time.
Operations	Terminal Server capacity constraints can affect multiple Terminal Server clients.
Alert Threshold	Do not alert on this counter.
Related Measures	Terminal Services\Active Sessions Terminal Services\Inactive Sessions

Every Terminal Server session is provided with a Windows desktop environment, which is supported by private instances of several critical processes: the Explorer desktop shell, a Winlogon process for authentication, a private copy of the Windows Client/Server Subsystem, Csrss.exe, the Lsass.exe, and Smss.exe security subsystem processes. In addition, Terminal Server creates process address spaces associated with the desktop application that the Terminal Server client is running remotely. A Terminal Server supporting a large number of Terminal Server clients must be able to sustain a large number of process address space and execution threads.

In practice, this means that large Terminal Server deployments on 32-bit machines can encounter severe 32-bit virtual memory addressing constraints. With enough Terminal Server clients on a 32-bit server machine, virtual memory shortages in the system area can occur, specifically in the system Paged pool or the pool of available system PTEs.

More Info For more information about virtual memory shortages, see the "Windows Server 2003 Terminal Server Capacity and Scaling" white paper at *http://www.microsoft.com/windowsserver2003/techinfo/overview/tsscaling.mspx*, and the discussion in Chapter 5, "Performance Troubleshooting," which addresses 32-bit virtual memory constraints.

The Memory\Pool Paged Bytes and Memory\Free System Page Table Entries counters should be tracked on Terminal Server machines to determine whether 32-bit virtual memory shortages are a concern. Note that systems with 64-bit virtual addressing, such as x64 and IA64 systems, provide more Terminal Server capacity than 32-bit systems.

A related concern, discussed at length in the "Windows Server 2003 Terminal Server Capacity and Scaling" white paper, is a potential shortage of physical memory for the system file cache because of contention for RAM with the system's Paged pool and pool of System PTEs. The file cache shares the range of system virtual memory available to the operating system with the Paged pool and the System PTE pool. Direct evidence that the size of the system file cache might be constrained is obtained by monitoring the Cache\Copy Read Hits % counter. For the sake of delivering good performance to Terminal Server clients, the Cache\Copy Read Hits % counter should be consistently above 95 percent. In fact, the authors of the "Windows Server 2003 Terminal Server Capacity and Scaling" white paper recommend that Cache\Copy Read Hits % counter remain at a 99 percent level for optimal performance.

Chapter 4

Performance Monitoring Procedures

In this chapter:

Understanding Which Counters to Log 314
Daily Server Monitoring Procedures. 316
Using a SQL Server Repository. 365
Capacity Planning and Trending 379
Counter Log Scenarios 388
Troubleshooting Counter Collection Problems 395

The regular performance monitoring procedures you implement need to be able to serve multiple purposes, including the following:

- Detecting and diagnosing performance problems
- Verifying that agreed-upon service levels are being met
- Supporting proactive capacity planning initiatives to anticipate and relieve impending resource shortages before they impact service levels

In this chapter, sample performance-monitoring procedures are described that will help you meet these important goals. These sample procedures, which center on a daily performance data-gathering process that you can easily implement, are generalized and thus are appropriate for both small- and large-scale enterprises. However, you will likely need to customize them to some degree to suit your specific environment.

These sample procedures will also help you start diagnosing common server performance problems. Additional recommendations will address performance alerts, management reporting, and capacity planning.

The final section of this chapter documents procedures that you should follow when you experience a problem using the Performance Monitor to gather specific performance statistics. For example, applications that install or uninstall performance counters can sometimes do so incorrectly, in which case you can use the procedures described here to restore the performance counter infrastructure integrity. Doing so will ensure that Performance Monitor correctly reports performance statistics.

The sample procedures in this chapter are designed to help you anticipate a wide variety of common performance problems. They rely on the Log Manager and Relog automated command-line tools that were discussed in Chapter 2, "Performance Monitoring Tools." Log Manager and Relog will help you log counters from the local machine to a local disk and gather the most important performance counters, which were highlighted in Chapter 3, "Measuring Server Performance."

These sample automated procedures allow you to detect and diagnose many performance problems. They rely on background data-gathering sessions that you may analyze at length later after a problem is reported. Still, the amount of data it is suggested you collect continuously for daily performance monitoring will not always be adequate to solve every performance-related problem. Sometimes you will need to augment these background data collection procedures with focused real-time monitoring. Use real-time monitoring when you need to gather more detailed information about specific situations.

More Info Trying to diagnose any complex performance problem is often challenging. Trying to diagnose a complex performance problem in real time using the System Monitor is even more difficult. Unless you know precisely what you are looking for and the problem itself is persistent, it can be difficult to use the System Monitor in real time to identify the cause of a performance problem. In a real-time monitoring session, you have so many counters and instances of counters to look at and analyze that the problem might disappear before you are able to identify it. Problem diagnosis is the focus of Chapter 5, "Performance Troubleshooting."

Understanding Which Counters to Log

A daily performance monitoring procedure that is effective in helping you detect, diagnose, and resolve common performance problems must gather large quantities of useful performance data, which you can then edit, summarize, and use in management reporting and capacity planning. The first decision you will have to make is about which performance counters, among all those that are available, you should gather on a regular basis.

Background Performance Monitoring

A daily performance monitoring regimen that sets up background counter log sessions to gather performance data on a continuous basis allows you to detect and resolve common performance problems when they arise. Because it is impossible to know in advance which key resources are saturated on a machine experiencing performance problems, in an ideal world you would collect performance data on *all* the key resources, such as the processor, memory, disk, and network. However, overhead considerations dictate that you can never collect all the available performance information about all resources and all workloads all the time on any sizable machine running Microsoft Windows Server 2003. Thus, you must be selective about which data you are going to gather and how often you will collect it. Striking a balance between the amount of data you will gather and analyze and the costs associated with that process is important.

To detect and diagnose common performance problems involving overloaded resources, you need to gather a wide range of detailed performance data on processor, memory, disk, and network utilization and the workloads that are consuming those resources. This data should include any performance counters that indicate error conditions, especially errors resulting from a shortage of internal resources. This chapter discusses some of the key error indicators you should monitor.

Management Reporting

Normally, much less detailed information is required for service-level and other forms of management reporting. Thus, the level of detail provided by daily performance monitor procedures appropriate for detecting and resolving common performance problems is more than adequate for management reporting. In fact, to build management reporting procedures that scale efficiently across a large organization, you typically would find it helpful to summarize your daily performance counter logs first, prior to reporting. Summarizing the daily counter logs will reduce the size of the files that need to be transferred around the network and improve the efficiency of the reporting procedures.

Service-level reporting focuses on resource consumption and measures of load such as logons, sessions, transaction rates, and messages received. Service-level reporting is often of interest to a wider audience of system management professionals, many of whom might not be intimately familiar with the way Windows Server 2003 and its major server applications function. Consequently, it is important to avoid reporting too much technical detail in management reports aimed at this wider audience. Rather, focus service-level reporting on reporting a few key measures of resource utilization and load that are widely understood.

Capacity Planning

Finally, to implement proactive capacity planning, in which you identify workload growth trends, reliably predict workload growth, and forecast future requirements, you track and accumulate historical data on key resource usage and consumption levels over time. You will probably need to do this for only a small number of key computer components such as the processor, the networks and the disks, and a few key applications. The counter log data that feeds your management reporting processes will be edited and summarized again to support capacity planning, at which point the emphasis shifts to building an historical record of computer resource usage data.

You can reliably predict the future with reasonable accuracy only when you have amassed a considerable amount of historical data on the patterns of workload growth. For example, for every unit of time that you want to predict future workload growth, you need, at a minimum, an historical record equal to twice that amount of time. Thus, capacity planning requires that you maintain a record of resource usage over long periods of time. Typically, only when you have accumulated at least 2–3 years of data can you make reasonably accurate forecasts of capacity requirements that will feed decision making for an annual budget cycle.

In planning for future capacity requirements, seasonal patterns of activity often have a major impact. Seasonal variations in many production workloads commonly occur in monthly and annual cycles. For example, higher rates of financial transaction processing are often associated with month-end and year-end closings. You will need to provide computer capacity that is sufficient to absorb these month-end and year-end peak loads. In a retail sales organization, you are likely to find that patterns of consumer purchases are tied to holiday gift giving, when you can expect much higher transaction volume. It goes without saying that these peak transaction rates must be accommodated somehow. You will be able to factor in seasonal variations in workload demand only after you accumulate historical data reflecting multiple cycles of that seasonal activity.

Daily Server Monitoring Procedures

This section details a model daily performance monitoring procedure, which is part of a comprehensive program of proactive performance management. This procedure includes the following daily activities:

- Automatically gathering an in-depth view of system performance using counter logs
- Monitoring key system and server application error indicators
- Setting up alerts that automatically trigger the collection of detailed, diagnostic counter logs
- Developing management reports on key performance metrics that can be viewed by interested parties in your organization
- Retaining summarized performance statistics to support capacity planning
- Managing the counter logs that are created automatically during these processes

Remember that the model performance monitoring practices and procedures discussed here will require tailoring for use in your environment, based on your site-specific requirements. For instance, each IT organization tends to have unique management reporting requirements impacting the type and quantity of reports generated and the data included on those reports. The practices and procedures here represent a good place for you to start building an effective performance management function within your IT organization.

Daily Counter Logs

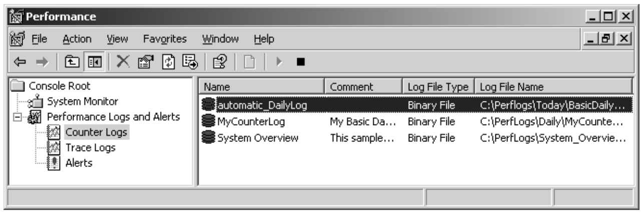
The first step in monitoring machines running Windows Server 2003 is to establish automated data logging using the Log Manager (logman) command-line utility.

The command shown in Listing 4-1 establishes a daily performance logging procedure using a settings file that defines the performance counters you want to gather. (A sample settings file is described later.) The command also references a command file to be executed when the daily counter log files are closed. (A sample script to perform typical post-processing is also illustrated.)

Listing 4-1 Establishing Daily Performance Logging

```
logman create counter automatic_DailyLog -cf "c:\perflogs\basic-counters-
setting-file.txt" -o C:\Perflogs\Today\BasicDailyLog -b 1/1/2004 00:00:00
-cnf 24:00:00 -si 1:00 -f BIN -v mmddhhmm -rc c:\perflogs\post-process.vbs
```

After you execute this command, the counter log you defined is visible and should look similar to the display shown in Figure 4-1. If you use the counter log's graphical user interface, you can confirm the properties used to establish the logging session, as illustrated.



Name	Comment	Log File Type	Log File Name
automatic_DailyLog		Binary File	C:\Perflogs\Today\BasicDaily...
MyCounterLog	My Basic Da...	Binary File	C:\PerfLogs\Daily\MyCounte...
System Overview	This sample...	Binary File	C:\PerfLogs\System_Overvie...

Figure 4-1 Properties used to establish the logging session

As documented in Chapter 2, “Performance Monitoring Tools,” the Log Manager utility allows you to configure background counter log data-gathering sessions. Table 4-1 parses the Log Manager command parameters that are used in Listing 4-1 and explains what it is they accomplish.

Table 4-1 Parsing the Log Manager Command Parameters

Log Manager Parameter	Explanation
-cf "c:\perflogs\basic-counters-setting-file.txt"	The counters to log are specified in a counters settings file. An example of a basic-counters-setting-file is provided in the section “A Sample Counter Settings File” in this chapter.
-b 1/1/2004 00:00:00 -cnf 24:00:00	Logging is initiated automatically by the System Monitor logging service as soon as your machine reboots and runs continuously for a 24-hour period.
-si 1:00	Data samples are collected once per minute.
-f BIN	Data logging is performed continuously to a binary log file. The binary format is used for the sake of efficiency and to save on the amount of disk space consumed.
-v mmddhhmm	Automatic versioning is used to create unique daily counter log file names.
-rc c:\perflogs\post-process.bat	At the end of a data logging session, when the log file is closed, a script is launched. This script performs file management and other post-processing. This post processing includes deleting older copies of the counter log files that were created previously, and summarizing the current log file for daily reporting. A sample post-processing script is provided in “Automated Counter Log Processing” in this chapter.

The Log Manager *-v* parameter allows you to generate unique file names for the counter logs created daily. The Performance Logs and Alerts snap-in supports an additional file versioning option that specifies the date in *yyyymmdd* format. If you prefer to have counter log file names automatically versioned using a *yyyymmdd* format, you can use the Performance snap-in afterward to manually modify the counter log properties to append the year, month, and day to the counter log file name.

When you are satisfied that the counter logging session you created is correctly specified, issue the command shown in Listing 4-2 to start logging data.

Listing 4-2 Starting Logging Data

```
logman start counter automatic_DailyLog
```

Logging Local Counters to a Local Disk

The daily performance monitoring procedure recommended here generates counter logs that contain local counters that are written in binary format to a local disk. The example writes counter log files to a local disk folder named C:\Perflogs\Today, although any suitable local disk folder will do. Binary log file format is recommended because it is more efficient and consumes less overhead.

Logging local counters to a local disk permits you to implement a uniform performance monitoring procedure across all the machines in your network, enabling you to scale these procedures to the largest server configurations, no matter what the network topology is.

The daily performance monitoring procedures documented here assume that local counter log data is written to a local disk folder, although other configurations are possible. Counter logs, for example, can be used to gather data from remote computers. Gathering data remotely is appropriate when you cannot get physical access to the remote machine. In those circumstances, it is simpler to perform performance data gathering on one centrally located machine that is designed to pull counter data from one or more remote machines. However, such a procedure is inherently less robust than having counter logs running locally, because a data collection problem on *any* remote machine can impact all machines that the centralized process is designed to monitor. The network impact of monitoring remote machines must also be understood. This topic is discussed in greater detail in the “Counter Log Scenarios” section.

When you log counter log data to a local disk, you are required to manage the counter log files that are created on a regular basis so that they do not absorb an inordinate amount of local disk space. Without some form of file aging and cleanup, the counter logs you generate daily can be expected to absorb as much as 30–100 MB of local disk space on your servers each day they are active. In the section “Automated Counter Log Processing,” a sample post-processing script is provided that uses Microsoft Visual Basic Scripting Edition (VBScript) and Windows Script Host (WSH) and can perform this daily file management and cleanup.

Using the Relog command-line utility, you can transform binary format files later into any other form. For example, you can create summarized files that can then be transferred to a consolidation server on the network. You can also create comma-delimited text files for use with programs like Microsoft Excel, which can generate useful and attractive charts and graphs. Using Relog, you can also build and maintain a SQL Server database of consolidated counter log data from a number of servers that will serve the needs of capacity planners.

Logging to a Network Share

Instead of generating counter logs in binary format to a local disk folder, many people prefer to write counter logs to a network-shared folder. Because it can simplify file management, this approach is often preferred. If you opt for this method, however, note the following considerations, which might affect the scalability of your performance monitoring procedures at sites where a large number of servers exist:

- Be careful not to overload the network. If your daily counter log consumes 50 MB of disk space per day, spread over a 24-hour period, that consumption amounts to only about 600 bytes per server per second of additional load that your network must accommodate. To determine the approximate load on the network to perform remote data logging, multiply by the number of servers that will be logging to the same network shared folder.
- Make sure that your counter log session runs under a User ID that has permission to access the shared network folder. This User ID also must also be a member of the built-in Performance Log Users group. To add User credentials to a Log Manager session, use the *-u* parameter to specify UserName and Password. Using Performance Logs and Alerts in the Performance Monitor console, you must set the Run As parameter on the General properties page for your counter log.
- Ensure that the Windows Time Service is used to synchronize the clocks on all the servers that are writing counter log files to a shared network folder.

> **More Info** For a description of how to use the Windows Time Service to synchronize the clocks on the machines on your Windows Server 2003 network, see the documentation entitled "Windows Time Service Technical Reference" posted on TechNet at http://www.microsoft.com/resources/documentation/WindowsServ/2003/all/techref/en-us/W2K3TR_times_intro.asp.

- Embed the computer name in the file names of the counter log files so that you can easily identify the machine they come from.

These considerations can be easily accomplished using a WSH script. For example, the VBScript shown in Listing 4-3 returns the local computer name in a variable named *LogonServer*.

Listing 4-3 Identifying the Source Machine

```
Set WshShell = CreateObject("Wscript.Shell")
Set objEnv = WshShell.Environment("Process")
LogonServer = objENV("COMPUTERNAME")
```

Next, you can construct a file name with the local computer name embedded in it using the script code shown in Listing 4-4.

Listing 4-4 Constructing a File Name

```
Const PerflogFolderYesterday = "Z:\SharedPerflogs\Yesterday"
Const LogType = "blg"
Const LogParms = " -b 1/1/2004 00:00:00 -cnf 24:00:00 -si 1:00 -f BIN
-v mmddhhmm -rc c:\perflogs\post-process.vbs"
DailyFileName = PerflogFolderYesterday & "\" & LogonServer & "." &
"basicDailyLog" & "." & LogType
LogSessionName = LogonServer & "-daily-log"
```

Then you can execute the Logman utility from inside the script using WSH Shell's *Exec* method, as illustrated in Listing 4-5.

Listing 4-5 Executing the Logman Utility

```
Const logSettingsFileName = "Z:\SharedPerflogs\log-counters-setting-file.txt"
command = "logman create counter " & LogSessionName & " -o " & DailyFileName & " -cf
" & logSettingsFileName & " " & LogParms
Set WshShell = Wscript.CreateObject("WScript.Shell")
Set execCommand = WshShell.Exec(command)
Wscript.Echo execCommand.StdOut.ReadAll
```

The final line of the script obtains the *ReadAll* property of the *StdOut* stream, which contains any messages generated by the Logman utility from the Shell Object's *Exec* method, allowing you to determine whether the utility executed successfully.

Additional considerations for performing remote logging and writing counter logs to a remote disk are discussed in "Counter Log Scenarios."

Baseline Measurements

Baseline performance data is a detailed performance profile of your machine that you store for future use in case you need to determine what has changed in your environment. You can compare the previous baseline set of measurements to current data to detect any major changes in the workload. Sometimes, small incremental changes that occur slowly add up to major changes over time. By using performance reports that are narrowly focused in time, you can easily miss seeing the extent and scope of these incremental changes. One good way to assess the extent of these changes over time is to compare a detailed view of the current environment with one of your more recent sets of baseline measurements.

The daily performance monitoring procedure recommended here is suitable for establishing a set of baseline measurements for your machine. Every six months or so, copy this counter log file and set it aside. Copy the counter log to a secure location where you can store it long-term. It is also a good idea to save a baseline counter log before and after any major system hardware or software configuration change.

To be useful as a baseline set of measurements, all counter data you gather should be kept in its original binary format. You might want to save these baseline measurement sets to offline archived storage so that you can keep them around for several years without incurring much cost.

Using a Counter Settings File

Using a counter settings file to specify the counters that you want to gather makes it easy to create and maintain uniform performance monitoring procedures across all your machines with similar requirements. For example, you might create a counter settings file for use on all your machines that are running Microsoft SQL Server. Similarly, your Web server machines running Internet Information Services (IIS) and .NET Framework applications would require a different counter settings file for best results.

Any counter settings file that you create is likely to include a base set of counters that reports utilization of the machine's principal resources—processors, memory, disks, and network interfaces. A simple counter settings file that would form the heart of almost any application-specific counter settings that you will need to create might look the example in Listing 4-6.

Listing 4-6 Simple Counter Settings File

```
\LogicalDisk(*)\% Free Space
\LogicalDisk(*)\Free Megabytes
\PhysicalDisk(*)\*
\Cache\*
\Processor(*)\*
\Memory\*
\System\*
\Network Interface(*)\*
\IPv4\*
\TCPv4\*
```

Listing 4-6 gathers Logical Disk free space statistics and all counters from the Cache, Memory, Network Interface, Physical Disk, Processor, and System objects. The Logical Disk free space measurements enable you to determine when your file system is running out of capacity.

Adding application-specific counters The counter settings file shown in Listing 4-6 lacks any application-specific performance data. This simple counter settings file can be enhanced significantly by adding counters associated with applications that the server runs. (See Table 4-2.)

Table 4-2 Adding Counters

Role That Your Windows Server 2003 Machine Serves	Gather Counters from These Objects	Gather Counters from These Processes
Domain Controller	*NTDS*	*lsass*, *smss*
Remote Access Server	*RAS Total*, *RAS Port*	*svchost*
Database Server	*SQL Server:General Statistics*, *SQL Server:Databases*, *SQL Server:Buffer Manager*, *SQL Server:Cache Manager*, *SQL Server:SQL Statistics*, *SQL Server:Locks*	*sqlserver*, *sqlagent*
Web Server	*Internet Information Services Global*, *FTP Service*, *Web Service*, *Web Service Cache*	*inetinfo*, *svchost*
File and Print Server	*Server*, *Server Work Queues*, *Print Queue*, *NBT Connection*	*svchost*, *spoolsv*
Terminal Server	*Terminal Services*, *Terminal Services Session*	*svchost*; *tssdis*
Exchange Server	*MSExchangeAL*, *MSExchangeDSAccess Caches*, *MSExchangeDSAccess Contexts*, *MSExchangeDSAccess Processes*, *Epoxy*, *MSExchangeIS Mailbox*, *Database ==> Instances*, *MSExchange Transport Store Driver*, *MSExchangeIS Transport Driver*, *MSExchangeSRS*, *MSExchange Web Mail*, *MSExchangeIMAP4*, *MSExchangePOP3*, *MSExchangeMTA*, *MSExchangeMTA Connections*, *SMTP Server*, *SMTP NTFS Store Driver*	*store*, *dsamain*
Application Server	*MSMQ Session*, *MSMQ Service*, *MSMQ Queue*	*dllhost*

To limit the size of the counter log files that are generated daily, do not create a single settings file that contains *all* the application-specific counters in Table 4-2 and their associated process-level data. Instead, create a series of application-specific settings files.

Tip When you do not know what server applications are installed and active on a machine, the typeperf command-line utility, documented in Chapter 2, "Performance Monitoring Tools," can be used. It generates a settings file that provides a full inventory of the extended application counters available for collection on the machine.

A sample counter settings file Following the guidelines given in the preceding section, a counter settings file for a File and Print Server might look like the sample in Listing 4-7.

Listing 4-7 Example Counter Settings File for a File and Print Server

```
\LogicalDisk(*)\% Free Space
\LogicalDisk(*)\Free Megabytes
\LogicalDisk(*)\Current Disk Queue Length
\PhysicalDisk(*)\Current Disk Queue Length
\PhysicalDisk(*)\Avg. Disk Queue Length
\PhysicalDisk(*)\Avg. Disk sec/Transfer
\PhysicalDisk(*)\Avg. Disk sec/Read
\PhysicalDisk(*)\Avg. Disk sec/Write
\PhysicalDisk(*)\Disk Transfers/sec
\PhysicalDisk(*)\Disk Reads/sec
\PhysicalDisk(*)\Disk Writes/sec
\PhysicalDisk(*)\Disk Bytes/sec
\PhysicalDisk(*)\Disk Read Bytes/sec
\PhysicalDisk(*)\Disk Write Bytes/sec
\PhysicalDisk(*)\Avg. Disk Bytes/Transfer
\PhysicalDisk(*)\Avg. Disk Bytes/Read
\PhysicalDisk(*)\Avg. Disk Bytes/Write
\PhysicalDisk(*)\% Idle Time
\PhysicalDisk(*)\Split IO/Sec
\Server\Bytes Total/sec
\Server\Bytes Received/sec
\Server\Bytes Transmitted/sec
\Server\Sessions Timed Out
\Server\Sessions Errored Out
\Server\Sessions Logged Off
\Server\Sessions Forced Off
\Server\Errors Logon
\Server\Errors Access Permissions
\Server\Errors Granted Access
\Server\Errors System
\Server\Blocking Requests Rejected
```

```
\Server\Work Item Shortages
\Server\Pool Nonpaged Bytes
\Server\Pool Nonpaged Failures
\Server\Pool Nonpaged Peak
\Server\Pool Paged Bytes
\Server\Pool Paged Failures
\Server Work Queues(*)\Queue Length
\Server Work Queues(*)\Active Threads
\Server Work Queues(*)\Available Threads
\Server Work Queues(*)\Available Work Items
\Server Work Queues(*)\Borrowed Work Items
\Server Work Queues(*)\Work Item Shortages
\Server Work Queues(*)\Current Clients
\Server Work Queues(*)\Bytes Transferred/sec
\Server Work Queues(*)\Total Operations/sec
\Server Work Queues(*)\Context Blocks Queued/sec
\Cache\Data Maps/sec
\Cache\Data Map Hits %
\Cache\Data Map Pins/sec
\Cache\Pin Reads/sec
\Cache\Pin Read Hits %
\Cache\Copy Reads/sec
\Cache\Copy Read Hits %
\Cache\MDL Reads/sec
\Cache\MDL Read Hits %
\Cache\Read Aheads/sec
\Cache\Lazy Write Flushes/sec
\Cache\Lazy Write Pages/sec
\Cache\Data Flushes/sec
\Cache\Data Flush Pages/sec
\Processor(*)\% Processor Time
\Processor(*)\% User Time
\Processor(*)\% Privileged Time
\Processor(*)\Interrupts/sec
\Processor(*)\% DPC Time
\Processor(*)\% Interrupt Time
\Memory\Page Faults/sec
\Memory\Available Bytes
\Memory\Committed Bytes
\Memory\Commit Limit
\Memory\Write Copies/sec
\Memory\Transition Faults/sec
\Memory\Cache Faults/sec
\Memory\Demand Zero Faults/sec
\Memory\Pages/sec
\Memory\Pages Input/sec
\Memory\Page Reads/sec
\Memory\Pages Output/sec
\Memory\Pool Paged Bytes
\Memory\Pool Nonpaged Bytes
\Memory\Page Writes/sec
```

```
\Memory\Pool Paged Allocs
\Memory\Pool Nonpaged Allocs
\Memory\Free System Page Table Entries
\Memory\Cache Bytes
\Memory\Cache Bytes Peak
\Memory\Pool Paged Resident Bytes
\Memory\System Code Total Bytes
\Memory\System Code Resident Bytes
\Memory\System Driver Total Bytes
\Memory\System Driver Resident Bytes
\Memory\System Cache Resident Bytes
\Memory\% Committed Bytes In Use
\Memory\Available KBytes
\Memory\Available MBytes
\Memory\Transition Pages RePurposed/sec
\Paging File(*)\% Usage
\Paging File(*)\% Usage Peak
\System\Context Switches/sec
\System\System Up Time
\System\Processor Queue Length
\System\Processes
\System\Threads
\Process(svchost,*)\% Processor Time
\Process(svchost,*)\% User Time
\Process(svchost,*)\% Privileged Time
\Process(svchost,*)\Virtual Bytes Peak
\Process(svchost,*)\Virtual Bytes
\Process(svchost,*)\Page Faults/sec
\Process(svchost,*)\Working Set Peak
\Process(svchost,*)\Working Set
\Process(svchost,*)\Page File Bytes Peak
\Process(svchost,*)\Page File Bytes
\Process(svchost,*)\Private Bytes
\Process(svchost,*)\Thread Count
\Process(svchost,*)\Priority Base
\Process(svchost,*)\Elapsed Time
\Process(svchost,*)\ID Process
\Process(svchost,*)\Pool Paged Bytes
\Process(svchost,*)\Pool Nonpaged Bytes
\Print Queue(*)\Total Jobs Printed
\Print Queue(*)\Bytes Printed/sec
\Print Queue(*)\Total Pages Printed
\Print Queue(*)\Jobs
\Print Queue(*)\References
\Print Queue(*)\Max References
\Print Queue(*)\Jobs Spooling
\Print Queue(*)\Max Jobs Spooling
\Print Queue(*)\Out of Paper Errors
\Print Queue(*)\Not Ready Errors
\Print Queue(*)\Job Errors
\Print Queue(*)\Enumerate Network Printer Calls
```

```
\Print Queue(*)\Add Network Printer Calls
\Network Interface(*)\Bytes Total/sec
\Network Interface(*)\Packets/sec
\Network Interface(*)\Packets Received/sec
\Network Interface(*)\Packets Sent/sec
\Network Interface(*)\Current Bandwidth
\Network Interface(*)\Bytes Received/sec
\Network Interface(*)\Packets Received Unicast/sec
\Network Interface(*)\Packets Received Non-Unicast/sec
\Network Interface(*)\Packets Received Discarded
\Network Interface(*)\Packets Received Errors
\Network Interface(*)\Packets Received Unknown
\Network Interface(*)\Bytes Sent/sec
\Network Interface(*)\Packets Sent Unicast/sec
\Network Interface(*)\Packets Sent Non-Unicast/sec
\Network Interface(*)\Packets Outbound Discarded
\Network Interface(*)\Packets Outbound Errors
\Network Interface(*)\Output Queue Length
\IPv4\Datagrams/sec
\IPV4\Datagrams Received/sec
\IPV4\Datagrams Received Header Errors
\IPV4\Datagrams Received Address Errors
\IPV4\Datagrams Forwarded/sec
\IPV4\Datagrams Received Unknown Protocol
\IPV4\Datagrams Received Discarded
\IPV4\Datagrams Received Delivered/sec
\IPV4\Datagrams Sent/sec
\IPV4\Datagrams Outbound Discarded
\IPV4\Datagrams Outbound No Route
\IPV4\Fragments Received/sec
\IPV4\Fragments Re-assembled/sec
\IPV4\Fragment Re-assembly Failures
\IPV4\Fragmented Datagrams/sec
\IPV4\Fragmentation Failures
\IPV4\Fragments Created/sec
\TCPV4\Segments/sec
\TCPV4\Connections Established
\TCPV4\Connections Active
\TCPV4\Connections Passive
\TCPV4\Connection Failures
\TCPV4\Connections Reset
\TCPV4\Segments Received/sec
\TCPV4\Segments Sent/sec
\TCPV4\Segments Retransmitted/sec
```

Depending on such factors as the number of physical processors installed, Physical Disks attached, the number of Logical Disks defined, and the number of Network Interface adaptors, the volume of counter data generated by the counter settings file in Listing 4-7 would generate could range from 30 MB per day for a small machine to 100 MB or more per day on a very large machine.

Gathering error indicators Many of the counters included in the sample counter settings file in Listing 4-7 are indicators of specific error conditions. These error conditions are not limited to resource shortages—some reflect improperly configured system and networking services, for example. Others might indicate activity associated with unauthorized users attempting to breach the machine's security. The value of including these counters in your daily performance monitoring procedures is that they allow you to pinpoint the times when system services encounter these error conditions.

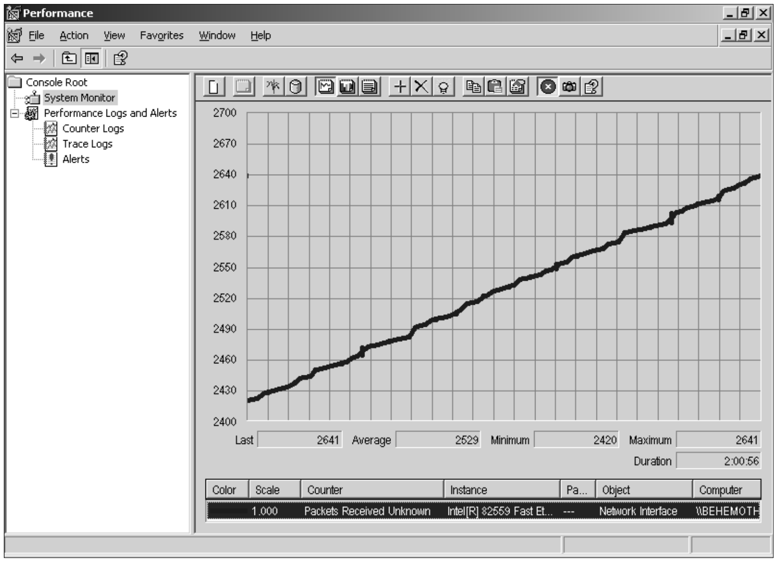
Listing 4-8 shows counters that were included in the settings file in Listing 4-7; these counters are included primarily as error indicators on a File and Print Server. This set of counters includes error indicators for File Server sessions, printer error conditions, and generic networking errors.

Listing 4-8 Counters That Are Error Indicators for a File and Print Server

```
\Server\Sessions Timed Out
\Server\Sessions Errored Out
\Server\Sessions Logged Off
\Server\Sessions Forced Off
\Server\Errors Logon
\Server\Errors Access Permissions
\Server\Errors Granted Access
\Server\Errors System
\Server\Blocking Requests Rejected
\Server\Work Item Shortages
\Server\Pool Nonpaged Bytes
\Server\Pool Nonpaged Failures
\Server\Pool Paged Failures
\Print Queue(*)\Out of Paper Errors
\Print Queue(*)\Not Ready Errors
\Network Interface(*)\Packets Received Discarded
\Network Interface(*)\Packets Received Errors
\Network Interface(*)\Packets Received Unknown
\Network Interface(*)\Packets Outbound Discarded
\Network Interface(*)\Packets Outbound Errors
\IPV4\Datagrams Received Header Errors
\IPV4\Datagrams Received Address Errors
\IPV4\Datagrams Received Unknown Protocol
\IPV4\Datagrams Received Discarded
\IPV4\Datagrams Outbound Discarded
\IPV4\Datagrams Outbound No Route
\IPV4\Fragment Re-assembly Failures
\IPV4\Fragmentation Failures
\TCPV4\Connection Failures
\TCPV4\Connections Reset
```

Counters, such as the ones featured in Listing 4-8 that record the number of error conditions that have occurred, are usually instantaneous, or raw counters. However, the counter values they contain are cumulative values. Because the number of error conditions that should be occurring is small, using an interval difference counter to track errors would result in reporting error rates per second that are so small that they would report zero values. So, to avoid reporting the valuable metrics as zero values, these counters report cumulative error counts.

The one drawback of this approach is that you cannot use the Alerts facility of the Performance Monitor to notify you when error conditions occur. The chart in Figure 4-2 plots values for one Network Interface error condition counter over the course of a 2-hour monitoring session. Notice that the curve marking the values of the Network Interface\Packets Received Unknown counter increases steadily during the monitoring interval. Whatever error condition is occurring is occurring regularly. Because the counter maintains the cumulative number of error conditions that have occurred since the system was last booted, once an error condition occurs, the counter value remains a nonzero value. You can see that once the alert on one of these counters is triggered, the alert will continue to occur at every measurement interval that follows. Such alerts will overrun the Application event log.

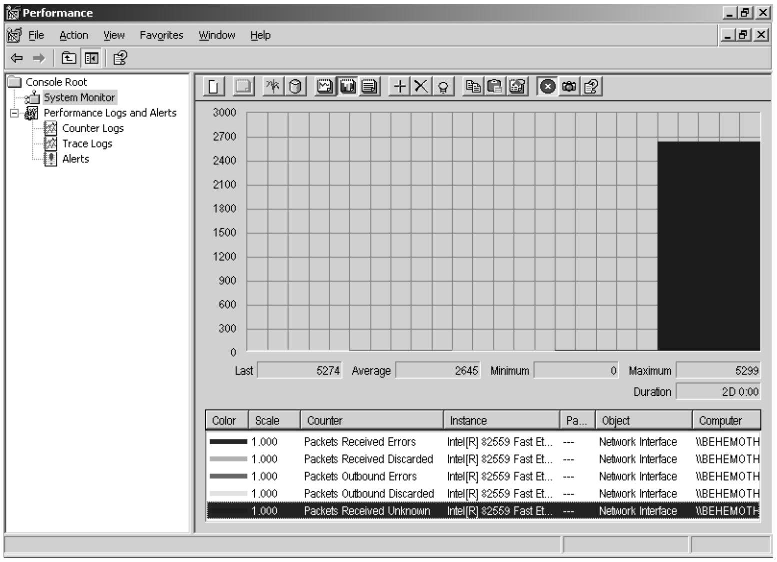


Last 2641 Average 2529 Minimum 2420 Maximum 2641
Duration 2:00:56

Color	Scale	Counter	Instance	Pa...	Object	Computer
	1.000	Packets Received Unknown	Intel[R] 82559 Fast Et...	---	Network Interface	\\BEHEMOTH

Figure 4-2 Values for one Network Interface error condition counter over the course of a 2-hour monitoring session

An easy and quick way to check to whether these error conditions are occurring is to develop a report using the Histogram view, which allows you to identify all the nonzero error conditions. An example of this approach is shown in Figure 4-3, which shows that only the Packets Received Unknown error indicator occurs over a two-day monitoring interval



Last 5274 Average 2645 Minimum 0 Maximum 5299
Duration 2D 0:00

Color	Scale	Counter	Instance	Pa...	Object	Computer
▬	1.000	Packets Received Errors	Intel[R] 82559 Fast Et...	---	Network Interface	\\BEHEMOTH
▬	1.000	Packets Received Discarded	Intel[R] 82559 Fast Et...	---	Network Interface	\\BEHEMOTH
▬	1.000	Packets Outbound Errors	Intel[R] 82559 Fast Et...	---	Network Interface	\\BEHEMOTH
▬	1.000	Packets Outbound Discarded	Intel[R] 82559 Fast Et...	---	Network Interface	\\BEHEMOTH
▬	1.000	Packets Received Unknown	Intel[R] 82559 Fast Et...	---	Network Interface	\\BEHEMOTH

Figure 4-3 Histogram view showing error condition counters that have nonzero values

If you remove all the error condition counters from the Histogram that report zero values, you can then switch to the Chart view to investigate the remaining nonzero counters. Using the Chart view, you can determine when these error conditions occurred and the rate at which they occurred.

Using Alerts Effectively

Because of the resulting overhead and the size of the counter logs, it is rarely possible to gather all the performance statistics you would need all the time. Gathering process-level data tends to contribute most to the size of the counter log files you generate—even smaller machines running Windows Server 2003 typically have numerous processes running and thus large counter log files. In the sample counter settings file in Listing 4-7, this potential problem was addressed by collecting process-level performance counters for a only few processes at a time. In Listing 4-7 and in the examples listed in Table 4-1, process-level performance data was selected only for processes that were central to the server application running.

However, this selective garnering of data is not a wholly satisfactory solution because process-level performance data is required to detect runaway processes that are monopolizing the processor or are leaking virtual memory. The solution to this quandary is to use the Alerts facility of the Performance Monitor console to trigger a Log Manager counter log data collection session automatically when an alert has tripped its threshold. Implementing counter logging procedures in this fashion allows you to gather very detailed information about problems automatically, without resorting to gathering all the performance data all the time.

Follow these simple steps to set up a Log Manager counter log data collection session that starts automatically when an alert fires:

1. Define the Log Manager counter log you plan to use to gather detailed performance data about a specific condition.
2. Define the alert condition that will be used to trigger the counter log session.
3. In the definition of the alert action, start the counter log session you defined in step 1.

For example, to help detect that a process is leaking virtual memory, define a counter log session that will gather data at the process level on virtual memory allocations. Then define an alert condition that will be triggered whenever virtual memory allocations reach a critical level. Finally, define an alert action that will start the counter log session you defined in step 1 (of the preceding procedure) when the alert condition occurs.

The next section walks you through a sample implementation of an alert that will fire when there is a virtual memory shortage and then initiate a counter log data gathering session that will enable you to determine which process is responsible for the virtual memory shortage. In Chapter 5, "Performance Troubleshooting," additional techniques for detecting and diagnosing virtual memory leaks are discussed.

Caution You need to be careful that your performance alerts do not result in flooding the Application event log with an excessive number of messages. You should periodically review the Application event log to ensure that your alerts settings are not generating too many event log entries. Note that programs like the Microsoft Operations Manager can actively manage your event logs and consolidate and suppress duplicate event log entries so that duplicate alert messages are not generated for the same condition over and over.

Triggering Counter Logs Automatically

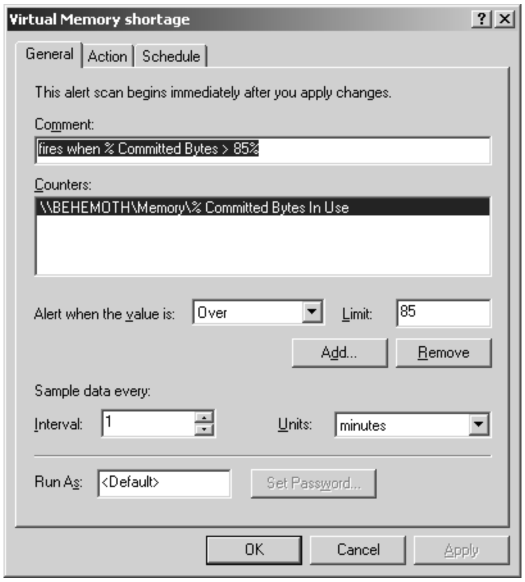
This section walks you step by step through a procedure that will generate diagnostic counter logs whenever there is a virtual memory shortage that might be caused by a runaway process leaking virtual memory.

Step 1: Define the counter log settings These are the counter log settings you want to use when the alert trips its threshold condition. For example, use the following command:

```
Logman create counter MemoryTroubleshooting -v mmddhhmm -c "\Memory\Available Bytes"
"\Memory\% Committed Bytes in Use" "\Memory\Cache Bytes" "\Memory\Pool Nonpaged Bytes"
"\Memory\Pool Paged Bytes" "\Process(*)\Pool Nonpaged Bytes" "\Process(*)\Pool Paged
Bytes" "\Process(*)\Pool Virtual Bytes" "\Process(*)\Pool Private Bytes"
"\Process(*)\Page Faults/sec" "\Process(*)\Pool Working Set" -si 00:30 -o
"c:\Perflogs\Alert Logs\MemoryTroubleshooting" -max 5
```

This command defines a MemoryTroubleshooting counter log session, which gathers virtual memory allocation counters at a process level, along with a few system-wide virtual memory allocation counters. In this example, these counters are sampled at 30-second intervals. Specifying the *-max* parameter shuts down data collection when the counter log reaches 5 MB. Using the Performance Monitor console, you could also limit the duration of the data collection session to a value of, for example, 10–15 minutes. Notice that the counter logs will be created in a C:\Perflogs\Alert Logs\ folder, distinct from the folder in which you create your regular daily performance logs. This separate folder enables you to more easily distinguish alert-triggered counter logging sessions from other counter logs that you create, and also makes it easier for you to manage them based on different criteria. For example, this MemoryTroubleshooting log is a detailed view that is limited to data on virtual memory allocations. You would never need to summarize this type of counter log, nor would you normally need to keep a detailed historical record on these types of problems.

Step 2: Define the alert General Properties You define the counter value threshold test (or tests) that trigger the specific alert condition using the tabs of the Virtual Memory Alert Properties page. In our ongoing example, to create an alert that is triggered by a virtual memory shortage, the threshold test is the value of the Memory\% Committed Bytes In Use exceeding 85 percent, as illustrated in Figure 4-4.


Virtual Memory shortage

General | Action | Schedule

This alert scan begins immediately after you apply changes.

Comment: fires when % Committed Bytes > 85%

Counters: \\BEHEMOTH\Memory\% Committed Bytes In Use

Alert when the value is: Over Limit: 85

Sample data every:

Interval: 1 Units: minutes

Run As: <Default>

Figure 4-4 The Memory\% Committed Bytes in Use over 85%

The rate at which the alert scan samples the performance counter (or counters) you select determines how frequently alert messages can be generated. In this example, the alert scan is scheduled to run once per minute. On a virtual memory-constrained machine in which the value of the Memory\% Committed Bytes In Use counter consistently exceeds the 85 percent alert threshold, this alert condition is triggered once every minute. Depending on the alert action you choose, this frequency might prove excessive.

Tip Where possible, define alert conditions that are triggered no more frequently than several times per hour under normal circumstances. Adjust those alert conditions that are triggered more frequently than 5–10 times per hour, even under severe conditions, so that they fire less frequently.

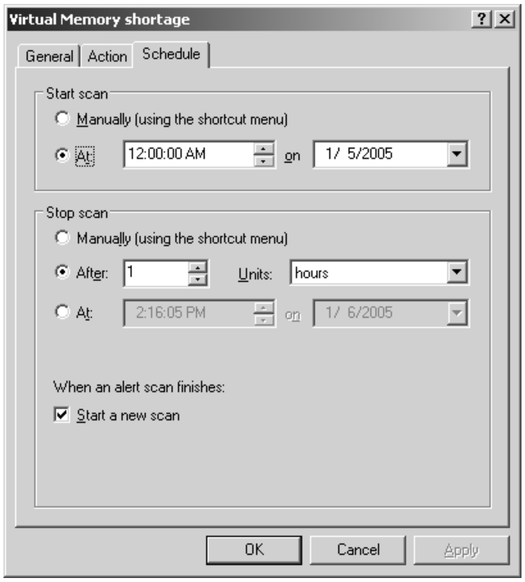
Alerts that occur too frequently annoy recipients. Psychologically, alert conditions that are triggered too frequently lose their effectiveness as notifications of significant events. Ultimately, they become treated as commonplace events that are safe to ignore. These human responses to the condition are understandable, but highly undesirable if the alert threshold truly represents an exceptional condition worthy of attention and additional investigation.

You can easily control the frequency for triggering alerts in either of the following ways:

1. Adjust the threshold condition so that the alert fires less frequently.
2. Slow down the rate at which the alert scan runs.

In addition, you might choose to limit alert actions to those that are calculated not to annoy anyone.

Step 3: Define the alert schedule The alert schedule parameters determine the duration of an alert scan. For best results, limit the duration of alert scans to 1–2 hours. For continuous monitoring, be sure to start a new scan when the current scan finishes, as illustrated in Figure 4-5.



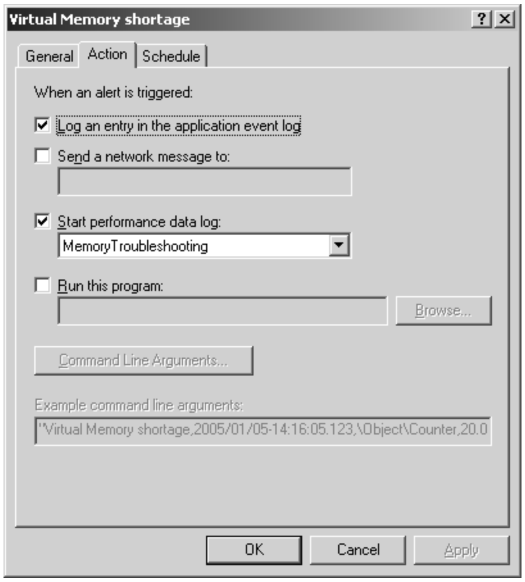
Start scan
- ○ Manually (using the shortcut menu)
- ● At: 12:00:00 AM on 1/ 5/2005

Stop scan
- ○ Manually (using the shortcut menu)
- ● After: 1 Units: hours
- ○ At: 2:16:05 PM on 1/ 6/2005

When an alert scan finishes:
- ☑ Start a new scan

Figure 4-5 Use the Start A New Scan check box

Step 4: Define the alert action The alert action parameters determine what action the alert facility will take when the alert condition is true. In this example, you want the alert facility to initiate the counter log you defined in step 1 to gather more detailed information about application performance at the process level at the time the alert was triggered. Note that the counter log you specify will be started just once per alert scan. In contrast, event log entries are generated at each sampling interval where the alert condition is true. Alert messages are also generated at each sampling interval where the alert condition is true. If you choose to run a program (Figure 4-6), the Alerts facility will schedule it to run only once per alert scan.



When an alert is triggered:

- [✓] Log an entry in the application event log
- [ ] Send a network message to:
- [✓] Start performance data log: MemoryTroubleshooting
- [ ] Run this program:

Example command line arguments:
"Virtual Memory shortage,2005/01/05-14:16:05.123,\Object\Counter,20.0

Figure 4-6 You can choose to run a program

The counter log or specified program is started immediately following the first sample interval of the alert scan in which the Alert condition is true. The duration of the counter logging session is determined by the schedule parameters you specified at the time you defined the counter log session. As noted earlier, using the command-line interface to the Log Manager, you can limit the size of the counter log file that will be created. Using the Performance Monitor console interface, you can limit the duration of the counter log session. A detailed counter log that allows you to explore the condition of the machine in depth over the next 10–30 minutes is usually effective. Obviously, if the alert conditions are triggered frequently enough and the detailed counter log sessions are relatively long, there is a risk that you will gather much more data about a potential problem than you can effectively analyze later. You might also use too much disk space on the machine that is experiencing the problem.

General Alerting Procedures

You will want to define similar alert procedures to launch counter logs that will allow you to investigate periods of excessive processor utilization, physical memory shortages, and potential disk performance problems. Table 4-3 summarizes basic Alert procedures that are valuable for your production servers. Refer to the discussion in

Chapter 3 on establishing configuration-dependent alert thresholds for the excessive paging alert condition.

Table 4-3 Settings for General Alert Conditions

Condition	Threshold Tests	Scan Frequency (In Seconds)	Additional Counters to Log	Log Sample Interval (In Seconds)
Excessive CPU utilization; potential process in an infinite loop	Processor(*)\% Processor Time > 98%	10–30	Processor(*)*; Process(*)\% Processor Time, Process(*)\% Privileged Time; Process(*)\% User Time	10–20 for 10–20 minutes
Process leaking virtual memory	Memory\% Committed Bytes In Use > 85%	10–30	Memory*; Process(*)\Private Bytes, Process(*)\Virtual Bytes, Process(*)\Pool Nonpaged Bytes, Process(*)\Pool Paged Bytes, Process(*)\Page File Bytes	15–30 for 10–30 minutes
Excessive paging to disk	Memory\Available Kbytes < <*threshold*>; Memory\Pages/sec > <*threshold*>	10–30	Memory*; Physical Disk(*n*)\% Idle Time, Physical Disk(*n*)\Avg. Disk Secs/Transfer, Physical Disk(*n*)\Transfers/sec; Process(*)\Page Faults/sec	10–20 for 10–20 minutes
Poor disk performance	Physical Disk(*n*)\Avg. Disk Secs/Transfer > 20; Physical Disk(*n*)\Transfers/sec > 200; Physical Disk(*n*)\Current Disk Queue Length > 5	15	Physical Disk(*n*)\% Idle Time, Physical Disk(*n*)\Avg. Disk Secs/Transfer, Physical Disk(*n*)\Transfers/sec	10 for 10 minutes

The alert threshold tests illustrated in Figure 4-3 show representative values that are good for many machines. For more specific recommendations on Alert thresholds for these counters, see the extended discussion in Chapter 3, "Measuring Server Performance."

Application Alerts

You might also want to define additional alert scans that are based on application-specific alerting criteria. In each situation, focus on alert conditions associated with excessive resource consumption, indicators of resource shortages, measurements showing large numbers of requests waiting in the queue, and other anomalies related to application performance. For some applications, it is worthwhile to generate alerts both when transaction rates are heavy and when transaction rates are unusually light, perhaps indicating the application stopped responding and transactions are blocked from executing.

Table 4-4 lists suggested settings for alerts for several popular server applications. The values you use in alert condition threshold tests for application servers usually varies with each site, as illustrated.

More Info For additional assistance in setting application-specific alert thresholds, consult the Microsoft Operations Framework documentation online, at the TechNet Products and Technologies section, at http://www.microsoft.com/technet/.

Table 4-4 Sample Settings for Application-Specific Alert Conditions

Application	Condition	Threshold Tests	Additional Objects to Log
Domain Controllers	Excessive Active Directory requests	NTDS\LDAP Searches/sec > <*threshold*>	NTDS*
Active Server Pages (or .NET Framework ASPX applications)	Excessive ASP request queuing	ASP\Requests Queued > <*threshold*>	ASP; Internet Information System Global; Web Service; Process(w3wp)*
File Server	Resource shortages	Server Work Queues(*)\Queue Length > <*threshold*>	Server; Server Work Queues; Cache; Memory; Processor; Process(svchost)\% Processor Time
SQL Server	Excessive database transaction rates	SQL Server:Databases(*n*)\Transactions/sec > <*threshold*>	SQL Server:Buffer Manager; SQL Server:Cache Manager; SQL Server:Memory Manager; SQL Server:Locks; SQL Server:SQL Statistics;SQL Server:Databases

Daily Management Reporting

Management reporting is a way to ensure that all parties at your installation who are interested in server performance have access to key information and reports that describe the behavior of your machines running Windows Server 2003. Management reporting should not be confused with the detailed and exploratory data analysis performed by experienced performance analysts when there are performance problems to diagnose. Management reporting focuses on a few key metrics that can be readily understood by a wide audience. Keep the charts and graphs you produce relatively simple and uncluttered.

The metrics that technical managers and other interested parties request the most include:

- Utilization measures of key resources like processors, memory, disks, and the network
- Availability and transaction rates of key applications such as Web servers, database servers, and Exchange mail and messaging servers
- Transaction service and response times of key applications, when they are available

Summarizing Daily Counter Logs

Because management reports should provide a high-level view of performance, you will want to generate daily files containing summarized performance data using the Relog utility. The detailed daily counter logs you generate are not the most efficient way to produce management reports. A daily counter log that gathers 1-minute samples continuously over the course of a 24-hour period accumulates 1440 observations daily of each performance counter you are logging. (The actual number of samples that would be generated daily is 1441 because one extra sample is needed at the outset to gather the starting values of all counters.) That is much more data than the System Monitor can squeeze onto a Chart view, which is limited to plotting 100 data points across its time-based x-axis. To create a Chart view for a 24-hour period, the System Monitor is forced to distill 14 separate measurements and plot summary statistics instead, which can easily create a distorted view of the performance statistics.

Using the Relog utility to create a summarized version of the daily counter logs you are gathering will simplify the process of creating useful daily management reports. For example, issuing the following command creates a compact version of one of your daily counter logs, summarized to convenient 15-minute intervals:

```
relog basicDailyLog_20031228.blg -o <computer-name>.basicDailyLog.blg
-f BIN -t 15
```

These summarized versions of your daily counter logs are very handy for building management reports quickly and efficiently.

Using summarized data in your management reports eliminates any distortions that can be introduced when the Chart view is forced to drop so many intermediate observations. Summarizing a 24-hour period to fifteen-minute intervals yields slightly fewer than 100 observations per counter, a number that fits neatly in a System Monitor Chart view. Note that summarizing the measurement data to this degree will smooth out many of the highs and lows evident in the original, detailed counter log. When you are investigating a performance problem, you will want to access the original counter log data, along with any detailed counter logs for the interval that were generated automatically by your alert procedures.

Tip The detailed counter logs that you create daily are suitable for generating management reports that focus on a peak 1- or 2-hour period. When peak hour transaction rates are two or more times heavier than average transaction rates, management reports that concentrate on these peak periods are extremely useful. When performance bottlenecks are evident only during these peak loads, reports that concentrate on these narrow periods of time are very helpful.

Consolidating performance data from multiple servers If you are responsible for the performance of a large number of servers, you will probably want to gather the counter logs from many of these servers so that you can report on them from a single location. To save on disk space at a central location, you might want to perform this consolidation using summarized counter logs, rather than the bulky, detailed daily counter logs that you collect initially. This consolidation can be performed daily using a series of automated procedures, as follows:

1. Use the Relog utility on the local server machine to create a summarized version of the daily counter log files that you produce.
2. Embed the computer name of the machine where the counter log originated into the summarized file name that Relog produces as output.
3. Copy the summarized counter log file to a central location.
4. At the consolidation server, use Relog to combine all the counter logs into a single output file that can be used for daily reporting.

Later in this section, examples of scripts that you can use to automate these daily processing functions are provided. As an alternative to consolidating counter logs gathered across many servers at a central location during a post-processing stage, you might consider creating all your counter logs at a centralized location at the outset. As discussed in “Logging to a Network Share” in this chapter, this option has scalability implications for very large server farms, so it should be implemented with caution. For a more detailed discussion of these issues, see the section entitled “Logging Local Counters to a Local Disk” earlier in this chapter.

When you execute the Relog utility to create summarized daily counter logs suitable for management reporting and archiving, you might choose to edit the performance metrics being retained even further, eliminating counters that you do not intend to report on in your management reports. When you execute the Relog utility to create summarized daily counter logs, you can reference a counter log settings file that will perform this editing.

For example, you can invoke Relog as follows, where relog-counters-setting-file.txt is a narrower subset of the original basic-counters-setting-file.txt that you used to generate the full daily counter logs.

```
relog basicDailyLog_20031228.blg -o <computer-name>.basicDailyLog.blg -cf relog-
counters-setting-file.txt -f BIN -t 15
```

Listing 4-9 shows the contents of a recommended Relog counter settings file suited to creating summarized files for daily management reporting.

Listing 4-9 relog-counters-setting-file.txt

```
\LogicalDisk(*)\% Free Space
\LogicalDisk(*)\Free Megabytes
\LogicalDisk(*)\Current Disk Queue Length
\PhysicalDisk(*)\Current Disk Queue Length
\PhysicalDisk(*)\Avg. Disk Queue Length
\PhysicalDisk(*)\Avg. Disk sec/Transfer
\PhysicalDisk(*)\Avg. Disk sec/Read
\PhysicalDisk(*)\Avg. Disk sec/Write
\PhysicalDisk(*)\Disk Transfers/sec
\PhysicalDisk(*)\Disk Reads/sec
\PhysicalDisk(*)\Disk Writes/sec
\PhysicalDisk(*)\Disk Bytes/sec
\PhysicalDisk(*)\Disk Read Bytes/sec
\PhysicalDisk(*)\Disk Write Bytes/sec
\PhysicalDisk(*)\Avg. Disk Bytes/Transfer
\PhysicalDisk(*)\Avg. Disk Bytes/Read
\PhysicalDisk(*)\Avg. Disk Bytes/Write
\PhysicalDisk(*)\% Idle Time
\Processor(*)\% Processor Time
\Processor(*)\% User Time
\Processor(*)\% Privileged Time
\Processor(*)\Interrupts/sec
\Processor(*)\% DPC Time
\Processor(*)\% Interrupt Time
\Memory\Page Faults/sec
\Memory\Available Bytes
\Memory\Committed Bytes
```

```
\Memory\Commit Limit
\Memory\Transition Faults/sec
\Memory\Cache Faults/sec
\Memory\Demand Zero Faults/sec
\Memory\Pages/sec
\Memory\Pages Input/sec
\Memory\Page Reads/sec
\Memory\Pages Output/sec
\Memory\Pool Paged Bytes
\Memory\Pool Nonpaged Bytes
\Memory\Page Writes/sec
\Memory\Cache Bytes
\Memory\Pool Paged Resident Bytes
\Memory\System Code Resident Bytes
\Memory\System Driver Resident Bytes
\Memory\System Cache Resident Bytes
\Memory\% Committed Bytes In Use
\Memory\Available KBytes
\Memory\Available MBytes
\Memory\Transition Pages RePurposed/sec
\System\Context Switches/sec
\System\Processor Queue Length
\Process(sqlserver)\% Processor Time
\Process(sqlserver)\% User Time
\Process(sqlserver)\% Privileged Time
\Process(sqlserver)\Virtual Bytes
\Process(sqlserver)\Page Faults/sec
\Process(sqlserver)\Working Set
\Process(sqlserver)\Private Bytes
\Process(sqlserver)\Elapsed Time
\Process(sqlserver)\Pool Paged Bytes
\Process(sqlserver)\Pool Nonpaged Bytes
\Process(inetinfo)\% Processor Time
\Process(inetinfo)\% User Time
\Process(inetinfo)\% Privileged Time
\Process(inetinfo)\Virtual Bytes
\Process(inetinfo)\Page Faults/sec
\Process(inetinfo)\Working Set
\Process(inetinfo)\Private Bytes
\Process(inetinfo)\Elapsed Time
\Process(inetinfo)\Pool Paged Bytes
\Process(inetinfo)\Pool Nonpaged Bytes
\RAS Total\Bytes Transmitted/Sec
\RAS Total\Bytes Received/Sec
\RAS Total\Total Errors/Sec
\Print Queue(*)\Total Jobs Printed
\Print Queue(*)\Bytes Printed/sec
\Print Queue(*)\Total Pages Printed
\Print Queue(*)\Jobs
\Network Interface(*)\Bytes Total/sec
\Network Interface(*)\Packets/sec
\Network Interface(*)\Packets Received/sec
\Network Interface(*)\Packets Sent/sec
\Network Interface(*)\Current Bandwidth
```

```
\Network Interface(*)\Bytes Received/sec
\Network Interface(*)\Bytes Sent/sec
\Network Interface(*)\Output Queue Length
\IP\Datagrams/sec
\IP\Datagrams Received/sec
\IP\Datagrams Sent/sec
\TCP\Segments/sec
\TCP\Connections Established
\TCP\Connections Active
\TCP\Connection Failures
\TCP\Connections Reset
\TCP\Segments Received/sec \TCP\Segments Sent/sec
\TCP\Segments Retransmitted/sec
```

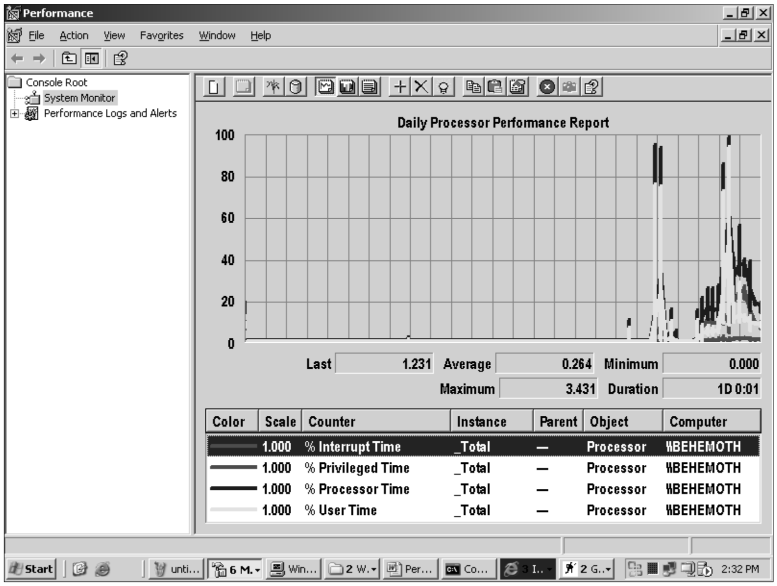
Sample Management Reports

The following section illustrates the management reports that you are likely to find most useful for presenting summarized performance statistics. These examples primarily use the System Monitor console's Chart view to present the performance data. You can, of course, always build more elaborate reporting charts and graphs than are available from the System Monitor console by using tools like Microsoft Excel or other charting programs.

Note that the sample management reports presented here showcase the key performance counters you should display. They also present uncluttered charts and graphs that are easy to read and decipher. The counter logs used to generate these charts were gathered from a relatively quiet machine performing little or no work during much of the reporting interval. The intent here is to focus your attention on the *presentation* of the data, not on the data itself. If you are interested in seeing examples of interesting charts and reports illustrating machines experiencing performance problems, you can find many relevant examples in Chapter 5, "Performance Troubleshooting."

For information about how to use the System Monitor Automation Interface to generate management reports like these automatically, see the section entitled "System Monitor Automation Interface" in Chapter 6, "Advanced Performance Topics."

Processor utilization Figure 4-7 illustrates a basic Chart view template that is suitable for many management reports. The report shows overall processor utilization and also breaks out processor utilization into its component parts. A large, easy-to-read font was selected, x- and y-axis gridlines were added, and a descriptive title was used. After you create report templates, selecting the counters you want and adding the proper presentation elements, you can save your settings as a Microsoft Management console .msc settings file that you can reuse.

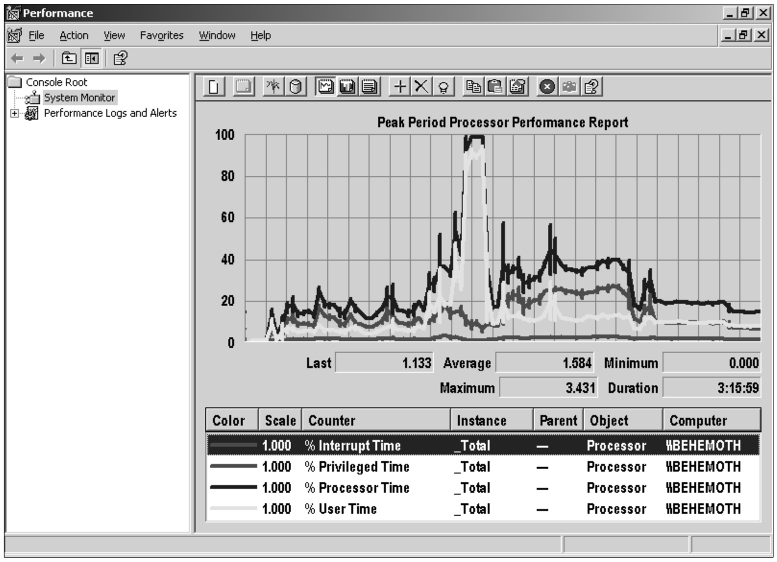


Last 1.231 Average 0.264 Minimum 0.000
Maximum 3.431 Duration 1D 0:01

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	1.000	% Interrupt Time	_Total	—	Processor	\\BEHEMOTH
▬	1.000	% Privileged Time	_Total	—	Processor	\\BEHEMOTH
▬	1.000	% Processor Time	_Total	—	Processor	\\BEHEMOTH
▬	1.000	% User Time	_Total	—	Processor	\\BEHEMOTH

Figure 4-7 Daily Processor Utilization Report

A similar chart that zooms in on a two-hour peak processing period is illustrated in Figure 4-8. Reusing the chart templates for similar management reports simplifies your task of explaining what these various charts and graphs mean.


Performance
File Action View Favorites Window Help
Console Root
System Monitor
Performance Logs and Alerts
Peak Period Processor Performance Report
100
80
60
40
20
0

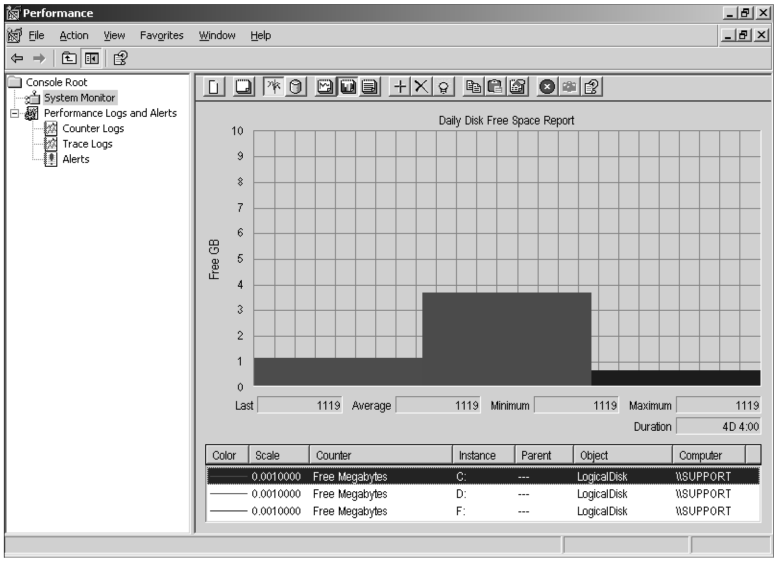
Last 1.133 Average 1.584 Minimum 0.000
Maximum 3.431 Duration 3:15:59

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	1.000	% Interrupt Time	_Total	—	Processor	\\BEHEMOTH
▬	1.000	% Privileged Time	_Total	—	Processor	\\BEHEMOTH
▬	1.000	% Processor Time	_Total	—	Processor	\\BEHEMOTH
▬	1.000	% User Time	_Total	—	Processor	\\BEHEMOTH

Figure 4-8 Peak Hour Processor Utilization Report

The daily processor utilization management report illustrated in Figure 4-7 used a summarized daily counter log file created using the Relog utility as input. The peak hour report in Figure 4-8 uses the full, unsummarized daily counter log with the Time Window adjusted to show approximately two hours of peak load data.

Available disk space The Histogram view illustrated in Figure 4-9 works well for reporting data from counters that tend to change very slowly, such as Logical Disk(*)\Free Megabytes. Using the Histogram view, you can show the amount of free space available on a large number of server disks very succinctly.

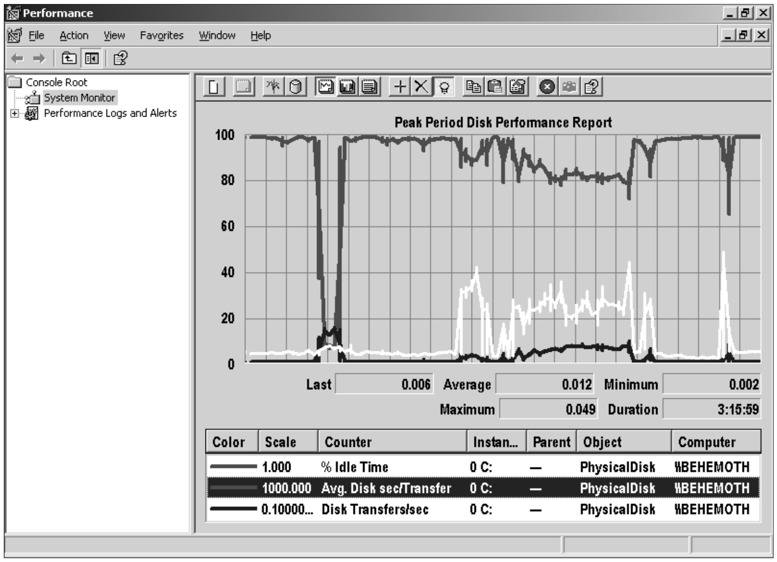


Last	Average	Minimum	Maximum	Duration
1119	1119	1119	1119	4D 4:00

Color	Scale	Counter	Instance	Parent	Object	Computer
———	0.0010000	Free Megabytes	C:	---	LogicalDisk	\\SUPPORT
———	0.0010000	Free Megabytes	D:	---	LogicalDisk	\\SUPPORT
———	0.0010000	Free Megabytes	F:	---	LogicalDisk	\\SUPPORT

Figure 4-9 Disk Free Space Report

Disk performance Figure 4-10 illustrates the readability problems that occur when you are forced to graph multiple counters against a single y-axis. The judicious use of scaling values makes this possible, of course. However, reporting metrics that require different scaling values to permit them all to be displayed against a single y-axis frequently confuses viewers. Figure 4-10 illustrates this problem, using disk performance measurements of idle time, device activity rates, and disk response time. Together, these three metrics accurately characterize physical disk performance. However, each individual counter usually requires using a separate scaling factor, and this tends to create confusion.


Performance
File Action View Favorites Window Help
Console Root
System Monitor
Performance Logs and Alerts
Peak Period Disk Performance Report
100
80
60
40
20
0

Last 0.006 Average 0.012 Minimum 0.002
Maximum 0.049 Duration 3:15:59

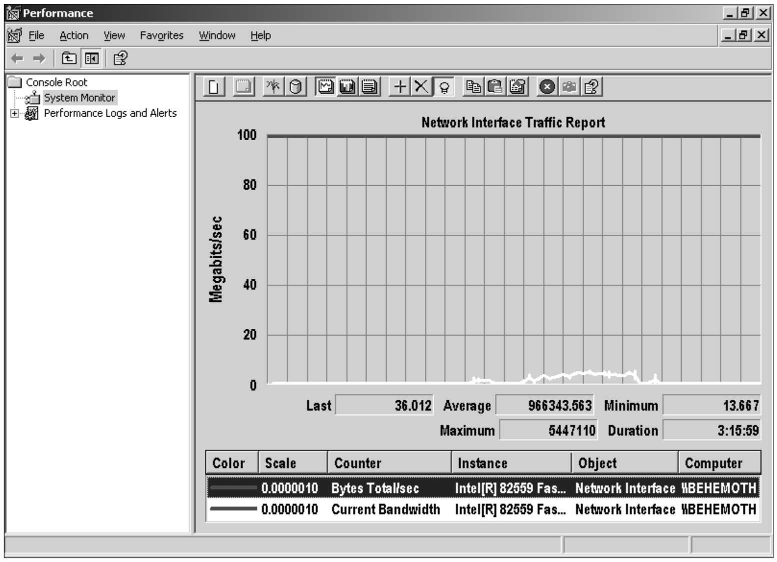
Color	Scale	Counter	Instan...	Parent	Object	Computer
▬	1.000	% Idle Time	0 C:	—	PhysicalDisk	\\BEHEMOTH
▬	1000.000	Avg. Disk sec/Transfer	0 C:	—	PhysicalDisk	\\BEHEMOTH
▬	0.10000...	Disk Transfers/sec	0 C:	—	PhysicalDisk	\\BEHEMOTH

Figure 4-10 Measuring disk performance using three metrics

The % Idle Time counter neatly falls neatly within the default y-axis that ranges from zero through 100, but the other metrics do not. Physical Disk Transfers/sec frequently exceeds 100 per second, so this counter cannot always be properly graphed against the same y-axis scale as % Idle Time. In addition, disk response time, measured in milliseconds, has to be multiplied by a scaling factor of 1000 to be displayed against the same y-axis as % Idle Time. Placing several counters that all use different scaling factors on the same chart invites confusion.

Nevertheless, this practice is sometimes unavoidable. Specifying an appropriate y-axis label can sometimes help minimize confusion. The alternative of providing three separate graphs, one for each measurement, has limited appeal because the content in an individual counter value chart is substantially diluted.

Network traffic Network traffic can be reported from the standpoint of each network interface, or summarized across all existing network interfaces. Because the Network Interface counters do not report network utilization directly, a report template like the one illustrated in Figure 4-11, which shows both the total bytes transferred across the network interface and the current bandwidth rating of the card, makes it relatively easy to visualize what the utilization of the interface is. Change the default y-axis maximum so that the Current Bandwidth counter represents a 100 percent utilization level at the top of the scale. Then the Bytes Total /sec counter for the network interface instance can be visualized as a fraction of the interface bandwidth. Be sure to label the y-axis appropriately, as in Figure 4-11.



Last	36.012	Average	966343.563	Minimum	13.667
		Maximum	5447110	Duration	3:15:59

Color	Scale	Counter	Instance	Object	Computer
	0.0000010	Bytes Total/sec	Intel[R] 82559 Fas...	Network Interface	\\BEHEMOTH
	0.0000010	Current Bandwidth	Intel[R] 82559 Fas...	Network Interface	\\BEHEMOTH

Figure 4-11 Network Interface Traffic Report

Historical Data for Capacity Planning

Capacity planning refers to the practices and procedures you institute to avoid performance problems as your workloads grow and change. When you maintain an historical record of computer resource usage by important production workloads, you can forecast future resource requirements by extrapolating from historical trends.

Capacity planning processes also benefit from growth forecasts that people associated with your organization's key production workloads can provide for you. Often, these growth forecasts are projections based on adding more customers who are users of these production systems. You must then transform these growth projections into a set of computer resource demands based on the resource usage profile of existing customers. Usually, a capacity plan brings together both kinds of forecasting information to project future workload requirements and the kinds of computer resources required to run those workloads.

This section focuses on procedures you can use to harness the daily performance counter logs you generate to create and maintain a database containing an historical record of computer resource usage data. You can then use this data to support capacity planning and other system management functions. This historical record of computer performance data is known as a *performance database*, or *PDB*. This section also describes procedures you can use to build a SQL Server–based performance database using the command-line tools discussed in Chapter 2, "Performance Monitoring Tools." A later section of this chapter, entitled "Using a SQL Server Repository," discusses using tools like Microsoft Excel to analyze the data in this SQL Server PDB and provides an example of using Excel to forecast future capacity requirements.

Why SQL Server?

Microsoft SQL Server makes an ideal performance database. If you are an inexperienced user of SQL Server, you might be reluctant to implement a SQL Server performance database. Rest assured that when you use command-line tools such as Logman and Relog, you can quite easily use SQL Server for this purpose. Microsoft SQL Server is an advanced Relational Database Management System (RDBMS) that is well suited to this task, particularly to handling the large amounts of capacity planning data you will eventually accumulate. Using SQL Server, you do not have to be responsible for managing lots of individual counter log files. Instead, you can consolidate all your historical information in one place in one or more sets of SQL Server Tables.

Another advantage of using SQL Server as the repository for your capacity planning data is that you are not limited to using the System Monitor for reporting. Once your counter log data is loaded into SQL Server, you can use a variety of data access, reporting, and analysis tools. One of the most popular and powerful of these reporting and analysis tools is Microsoft Excel. The section of this chapter entitled "Using a SQL Server Repository" discusses using Microsoft Excel to analyze, report, and forecast performance data that has been stored in a SQL Server PDB.

Creating Historical Summary Logs

Capacity planning requires summarized data that is accumulated and stored over long periods of time so that historical trends become evident. This section shows how the Relog utility can be used to summarize data and accumulate this counter log data to support capacity planning.

The summarized daily counter log files that are used for management reporting contain too much detail to be saved for weeks, months, and years. Consider running Relog again on the summarized daily counter log files that your daily performance monitoring procedure produces to edit these counter logs and summarize them further. Listing 4-10 provides an example.

Listing 4-10 Editing and Summarizing Daily Count Logs

```
relog <computer-name>.basicDailyLog.blg -o <computer-name>.dailyHistoryLog.blg -cf
summary-counters-setting-file.txt -f BIN -t 4
```

In this code, because the counter log files defined as input were already summarized to 15-minute intervals, the *-t 4* subparameter performs additional summarization to the 1-hour level.

Only a small number of counters are valuable for long-term capacity planning. The counter log settings file referenced by this Relog command drops many counter values that have little or no value long term. An example counter log settings file for a File and Print Server that is suitable for capacity planning is illustrated in Listing 4-11.

Listing 4-11 Counter Log Settings File for a Capacity Planning Database

```
\LogicalDisk(*)\% Free Space
\LogicalDisk(*)\Free Megabytes
\PhysicalDisk(*)\Avg. Disk sec/Transfer
\PhysicalDisk(*)\Avg. Disk sec/Read
\PhysicalDisk(*)\Avg. Disk sec/Write
\PhysicalDisk(*)\Disk Transfers/sec
\PhysicalDisk(*)\Disk Reads/sec
\PhysicalDisk(*)\Disk Writes/sec
\PhysicalDisk(*)\Disk Bytes/sec
\PhysicalDisk(*)\Disk Read Bytes/sec
\PhysicalDisk(*)\Disk Write Bytes/sec
\PhysicalDisk(*)\Avg. Disk Bytes/Transfer
\PhysicalDisk(*)\Avg. Disk Bytes/Read
\PhysicalDisk(*)\Avg. Disk Bytes/Write
\PhysicalDisk(*)\% Idle Time
\Processor(*)\% Processor Time
\Processor(*)\% User Time
\Processor(*)\% Privileged Time
\Processor(*)\Interrupts/sec
\Processor(*)\% DPC Time
\Processor(*)\% Interrupt Time
\Memory\Page Faults/sec
\Memory\Available Bytes
\Memory\Committed Bytes
\Memory\Commit Limit
\Memory\Transition Faults/sec
\Memory\Cache Faults/sec
\Memory\Demand Zero Faults/sec
\Memory\Pages/sec
\Memory\Pages Input/sec
\Memory\Page Reads/sec
\Memory\Pages Output/sec
\Memory\Pool Paged Bytes
\Memory\Pool Nonpaged Bytes
\Memory\Page Writes/sec
\Memory\Cache Bytes
\Memory\Pool Paged Resident Bytes
\Memory\System Cache Resident Bytes
\Memory\% Committed Bytes In Use
```

```
\Memory\Available KBytes
\Memory\Available MBytes
\Memory\Transition Pages RePurposed/sec
\Process(svchost,*)\% Processor Time
\Process(svchost,*)\% User Time
\Process(svchost,*)\% Privileged Time
\Process(svchost,*)\Virtual Bytes
\Process(svchost,*)\Page Faults/sec
\Process(svchost,*)\Working Set
\Print Queue(*)\Bytes Printed/sec
\Print Queue(*)\Total Pages Printed
\Print Queue(*)\Jobs
\Network Interface(*)\Bytes Total/sec
\Network Interface(*)\Bytes Received/sec
\Network Interface(*)\Bytes Sent/sec
\IPv4\Datagrams/sec
\IPv4\Datagrams Received/sec
\IPv4\Datagrams Sent/sec
\TCPv4\Segments/sec
\TCPv4\Segments Received/sec
\TCPv4\Segments Sent/sec
```

As just mentioned, because only a small number of counters are valuable for capacity planning, the counter log settings file can be even more concise than the example in Listing 4-11. In this listing, only high-level statistics on processor, memory, disk, and network utilization are kept, along with some server application-related process statistics and server application-specific statistics related to the printer workload throughput.

Accumulating historical data The Relog command procedure (Listing 4-10) constructs a binary counter log from your daily file and summarizes it to 1-hour intervals. One-hour intervals are suitable for longer-term capacity planning, forecasting, and trending. For capacity planning, you will want to build and maintain a summarized, historical counter log file that will contain information spanning weeks, months, and even years. You can use the Append option of the Relog utility to accomplish this. For instance, the following command uses Relog to append the daily summarized counter log to a counter log file that contains accumulated historical data using the *-a* append parameter:

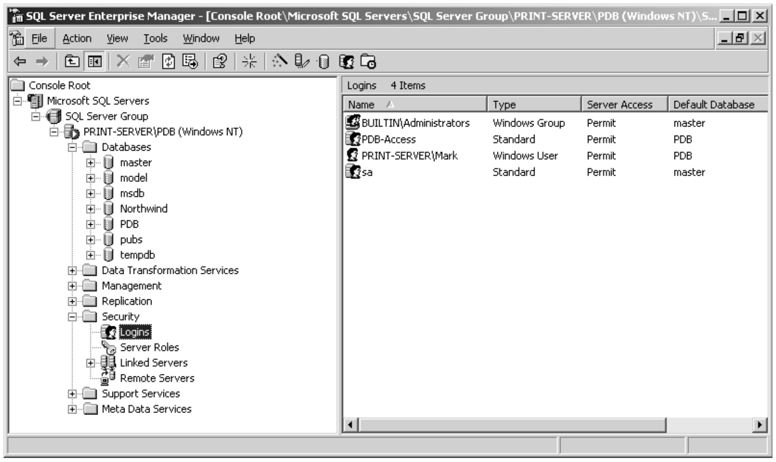
```
relog <computer-name>.dailyHistoryLog.blg -o
<computer-name>.historyPerformanceLog.blg -f BIN -a
```

Note that the *-a* option assumes that the output counter log file already exists.

Creating a SQL Server PDB

Before you are ready to use command-line tools like Logman and Relog to populate a SQL Server performance database, you must install an instance of SQL Server and define and configure the SQL Server database that you intend to use as a PDB. You can install a separate instance of SQL Server for your use exclusively as a PDB, or share an instance of SQL Server with other applications. Once you select the instance of SQL Server that you intend to use, follow these steps to define a performance database:

1. **Define the database where you intend to store the performance data.** Using the SQL Enterprise Manager console, define a new database and allocate disk space for it and its associated database recovery log. For the purpose of this example, the database that will be used has been called PDB. Once the database is created, you can access the system tables that are built automatically.
2. **Define the database security rules.** Security is a major aspect of SQL Server database administration. Until you specifically define their access rights, no external users can log on to gain access to the information stored in the PDB database. You must define at least one new database logon and grant that user access to the PDB database, as illustrated in Figure 4-12. In this example, the new database logon is called PDB-Access, which is defined by default to have access to the PDB database.

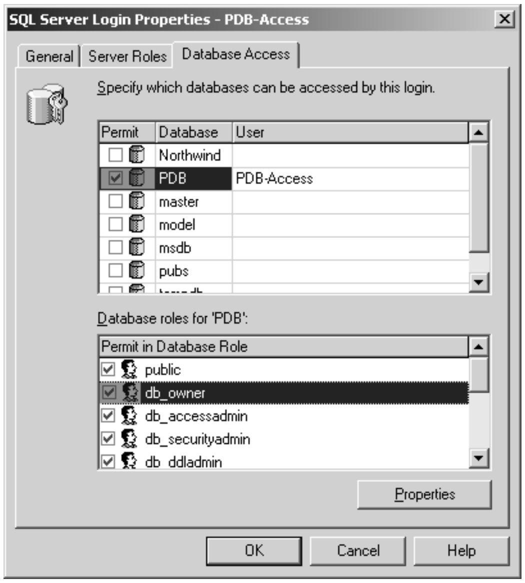


Logins 4 Items

Name	Type	Server Access	Default Database
BUILTIN\Administrators	Windows Group	Permit	master
PDB-Access	Standard	Permit	PDB
PRINT-SERVER\Mark	Windows User	Permit	PDB
sa	Standard	Permit	master

Figure 4-12 Defining a new database logon

Note that the SQL Server instance you use must be set up to use Microsoft Windows NT Authentication only. In addition, you must define a named user with the proper credentials, in this example a user named PDB-Access. At the level of the PDB database, you must also define the permissions for the logon you just defined. Figure 4-13 illustrates the PDB database user properties for the PDB-Access logon. Here PDB-Access has been granted permission to use all database security roles that are available. At a minimum, give this logon *db_owner* and *db_datawriter* authority. The *db_owner* role allows it to define, add, and change database tables, and *db_datawriter* gives the user permission to update and add data to the defined tables.



Specify which databases can be accessed by this login.

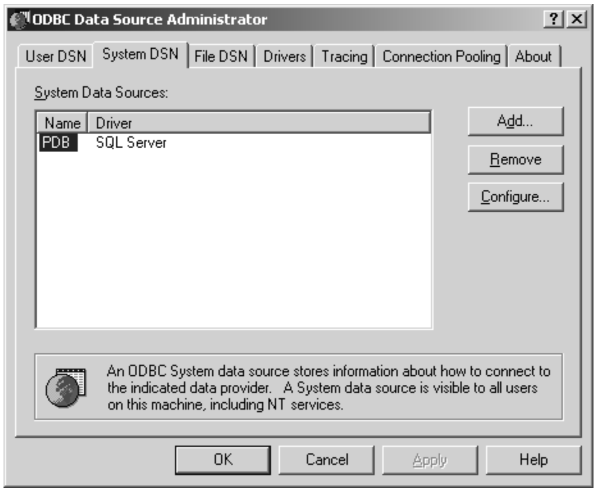
Permit	Database	User
☐	Northwind	
☑	PDB	PDB-Access
☐	master	
☐	model	
☐	msdb	
☐	pubs	

Database roles for 'PDB':

Permit in Database Role	
☑	public
☑	db_owner
☑	db_accessadmin
☑	db_securityadmin
☑	db_ddladmin

Figure 4-13 Database user properties for the PDB-Access logon

3. **Define the ODBC connection that will be used to access the PDB database.** To allow programs like the Relog utility, the Performance Logs and Alerts service, the System Monitor console for reporting, and Microsoft Excel for data mining and analysis to access the PDB database, you must define a System DSN connection. Using the ODBC Administrator, add a new System DSN connection that uses the SQL Server driver to allow access to the PDB database that you just defined, as illustrated in Figure 4-14.



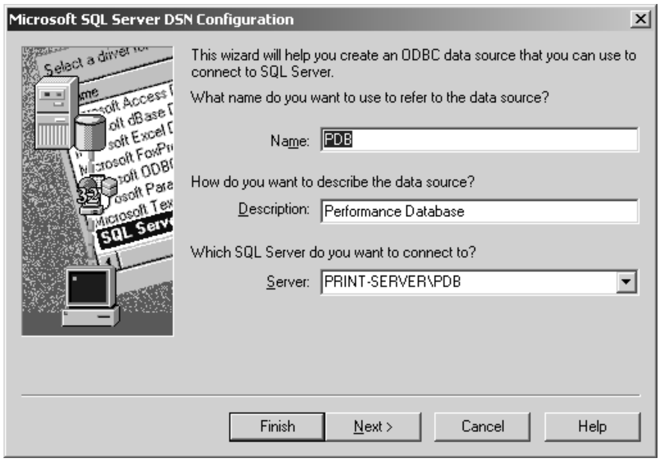
System Data Sources:

Name	Driver
PDB	SQL Server

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

Figure 4-14 Using the SQL Server driver to allow access to the PDB database

Doing this launches the ODBC connection wizard, which will enable you to configure this connection, as illustrated in Figure 4-15.



This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name: PDB

How do you want to describe the data source?

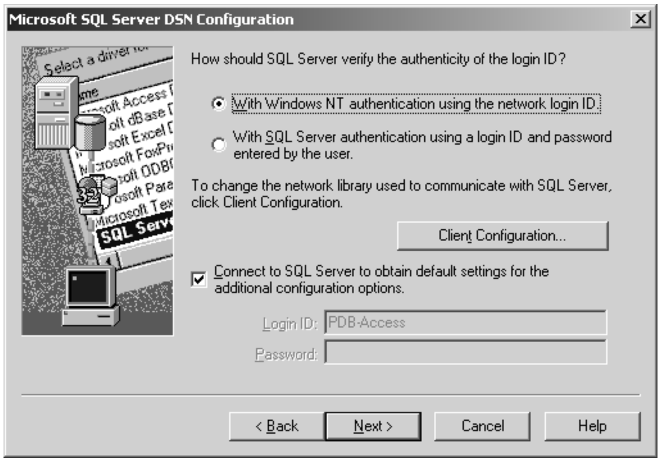
Description: Performance Database

Which SQL Server do you want to connect to?

Server: PRINT-SERVER\PDB

Figure 4-15 Using the ODBC connection wizard

You must supply a connection name, as illustrated, and point the connection to the SQL Server instance and database. In this example, the connection name is PDB. Then you must supply information about the security of the connection, as illustrated in Figure 4-16.


Microsoft SQL Server DSN Configuration

How should SQL Server verify the authenticity of the login ID?

[✓ With Windows NT authentication using the network login ID.]

With SQL Server authentication using a login ID and password entered by the user.

To change the network library used to communicate with SQL Server, click Client Configuration.

Client Configuration...

[✓] Connect to SQL Server to obtain default settings for the additional configuration options.

Login ID: PDB-Access

Password:

Figure 4-16 Verifying the authenticity of the connection

The Connect To SQL Server To Obtain Default Settings For The Additional Configuration Options check box, which is shown as selected in Figure 4-16, allows you to fully test the permissions on the connection when the connection definition is complete. Continue to fill out the forms provided by the ODBC connection wizard and click Finish to test the connection.

Populating the Repository

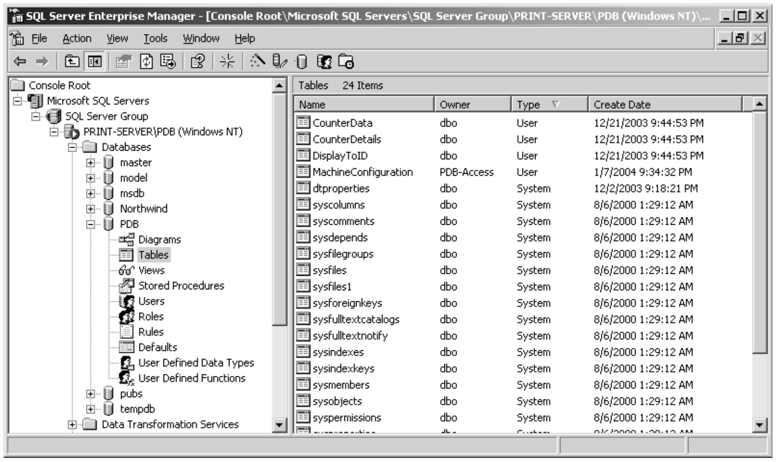
After the ODBC connection is defined, you can start loading counter log data into the PDB database you created. Because you plan to accumulate a large amount of historical performance information in the PDB, even the summarized daily counter log, which you created to build daily management reports, contains much more data than you will need for capacity planning purposes. Similar to the processing you performed when maintaining an historical binary format counter log, as discussed in "Creating Historical Summary Logs," you probably want to summarize this data even further and drop any counters that will not prove worthwhile to retain over long periods of time.

For example, the following Relog command takes one or more daily counter log files that are summarized to create a history file and inserts the output into the PDB database:

```
relog <computer-name>.historyPerformanceLog.blg -o "SQL:PDB!ByHour" -f SQL -cf
c:\perflogs\summary-counters-setting-file.txt
```

The output file specification, *-o "SQL:PDB!ByHour*", identifies the ODBC connection named PDB and defines a subsection of the PDB database that is called *ByHour*. You can define multiple capacity planning databases within the PDB database by identifying them using separate names.

After this command executes, it creates the PDB database tables that are used to store the counter log data. Using the SQL Enterprise Manager console, you can verify that the counter log tables have been created properly. Figure 4-17 illustrates the state of the PDB database following execution of the Relog command that references the SQL Server PDB ODBC connection.



Name	Owner	Type	Create Date
CounterData	dbo	User	12/21/2003 9:44:53 PM
CounterDetails	dbo	User	12/21/2003 9:44:53 PM
DisplayToID	dbo	User	12/21/2003 9:44:53 PM
MachineConfiguration	PDB-Access	User	1/7/2004 9:34:32 PM
dtproperties	dbo	System	12/2/2003 9:18:21 PM
syscolumns	dbo	System	8/6/2000 1:29:12 AM
syscomments	dbo	System	8/6/2000 1:29:12 AM
sysdepends	dbo	System	8/6/2000 1:29:12 AM
sysfilegroups	dbo	System	8/6/2000 1:29:12 AM
sysfiles	dbo	System	8/6/2000 1:29:12 AM
sysfiles1	dbo	System	8/6/2000 1:29:12 AM
sysforeignkeys	dbo	System	8/6/2000 1:29:12 AM
sysfulltextcatalogs	dbo	System	8/6/2000 1:29:12 AM
sysfulltextnotify	dbo	System	8/6/2000 1:29:12 AM
sysindexes	dbo	System	8/6/2000 1:29:12 AM
sysindexkeys	dbo	System	8/6/2000 1:29:12 AM
sysmembers	dbo	System	8/6/2000 1:29:12 AM
sysobjects	dbo	System	8/6/2000 1:29:12 AM
syspermissions	dbo	System	8/6/2000 1:29:12 AM

Figure 4-17 The state of the PDB database following execution of the Relog command

The counter log data is stored in three SQL Server tables—CounterData, CounterDetails, and DisplayToID. The format of these SQL Server tables reflects the data model that the performance monitoring facilities and services use to maintain a SQL Server repository of counter log data. This data model is discussed in detail below in the section entitled Using a SQL Server Repository.

Automated Counter Log Processing

In the preceding sections of this chapter, procedures were outlined to perform the following functions:

- Gather daily performance counter logs
- Gather diagnostic counter logs in response to alerts
- Summarize the daily performance counter logs to build management reports
- Populate a repository inside SQL Server that can be used in capacity planning

These are functions that need to be performed automatically on all the Windows Server 2003 machines requiring continuous performance monitoring. This section provides a sample WSH script to automate these daily procedures. This script is written in VBScript, and you can easily tailor it to your environment. It will reference the counter log settings files that were shown in the preceding sections, which create summarized counter log files and reference the PDB database in SQL Server defined in the previous section.

The complete sample post-processing script is shown in Listing 4-12. It is heavily commented and sprinkled with *Wscript.Echo* messages; this information will allow you to modify the script easily to suit your specific requirements. To enable these *Wscript.Echo* diagnostic messages, remove the Microsoft Visual Basic comment indicator, which is a single quotation mark character (').

Listing 4-12 Sample Post-Processing Script

```
‘VBscript for post-processing daily performance Counter Logs
‘
‘Initialization

CreatedTodayDate = Now

Const OverwriteExisting = True
Const LogType = "blg"
Const OldLogType = "Windows Backup File"
Const PMFileType = "Performance Monitor File"

Const relogSettingsFileName = "c:\perflogs\relog-counters-setting-file.txt"
Const relogParms = "-f BIN -t 15"
Const relogSummaryParms = "-f BIN -t 4"
Const relogHistoryParms = "-f BIN -a"
Const HistorySettingsFileName = "c:\perflogs\summary-counters-setting-file.txt"

Const PerflogFolderToday = "C:\Perflogs\Today"
Const PerflogFolderYesterday = "C:\Perflogs\Yesterday"
Const PerflogFolderAlerts = "C:\Perflogs\Alert Logs"
Const PerflogFolderHistory = "C:\Perflogs\History"

Set WshShell = CreateObject("Wscript.Shell")
Set objEnv = WshShell.Environment("Process")
LogonServer = objENV("COMPUTERNAME")
DailyFileName = PerflogFolderYesterday & "\" & LogonServer & "." & _
  "basic_daily_logDailyLog" & "." & LogType
SummaryFileName = PerflogFolderYesterday & "\" & LogonServer & "." & _
  "basic_history_logHistoryLog" & "." & LogType
HistoryFileName = PerflogFolderHistory & "\" & LogonServer & "." & _
  "history_performance_logPerformanceLog" & "." & LogType
‘WScript.Echo DailyFileName
‘WScript.Echo SummaryFileName
‘WScript.Echo HistoryFileName
```

```
Const AlertDaysOld = 3 ‘Number of days to keep Alert-generated
  Counter Log files

Set objFSO1 = CreateObject("Scripting.FileSystemObject")

If objFSO1.FolderExists(PerflogFolderYesterday) Then
  Set objYesterdayFolder = objFSO1.GetFolder(PerflogFolderYesterday)
Else
  Wscript.Echo "Yesterday folder does not exist. Will create " &
    PerflogFolderYesterday
  Set objYesterdayFolder = objFSO1.CreateFolder(PerflogFolderYesterday)
End If

Set objYesterdayFolder = objFSO1.GetFolder(PerflogFolderYesterday)
Set objTodayFolder = objFSO1.GetFolder(PerflogFolderToday)
Set objAlertsFolder = objFSO1.GetFolder(PerflogFolderAlerts)

‘Wscript.Echo "Begin Script Body"
objYesterdayFolder.attributes = 0
Set fc1 = objYesterdayFolder.Files

‘Wscript.Echo "Look for Yesterday's older backup files..."
   For Each f1 in fc1
     ‘Wscript.Echo "Found " & f1.name & " in " & PerflogFolderYesterday
     ‘Wscript.Echo "File type is " & f1.type
     If f1.type = OldLogType Then
       ‘Wscript.Echo "Old files of type " & f1.type & " will be deleted."
        filename = PerflogFolderYesterday & "\" & f1.name
       ‘Wscript.Echo "Delete " & filename
       objFSO1.DeleteFile(filename)
     End If
   Next

‘Wscript.Echo "Look for Yesterday's .blg files..."
   For Each f1 in fc1
      ‘Wscript.Echo "Found " & f1.name & " in " & PerflogFolderYesterday
      If f1.type = PMFileType Then
         NewName = PerflogFolderYesterday & "\" & f1.name & "." & "bkf"
         ‘Wscript.Echo f1.name & " will be renamed to " & NewName
        filename = PerflogFolderYesterday & "\" & f1.name
         ‘Wscript.Echo "Rename " & filename
         objFSO1.MoveFile filename, NewName
      End If
    Next

objYesterdayFolder.attributes = 0

‘Wscript.Echo "Look at Today's files..."
‘Wscript.Echo "Today is " & CStr(CreatedTodayDate)
```

```
   Set fc2 = objTodayFolder.Files
   For Each f2 in fc2
      filename = PerflogFolderToday & "\" & f2.name
      FileCreatedDate = CDate (f2.DateCreated)
      'Wscript.Echo filename & " was created on " & CStr (FileCreatedDate)

     If DateDiff ("d", CreatedTodayDate, FileCreatedDate) = 0 Then
         'Wscript.Echo "Skipping the file currently in use: " & filename

      Else
          'Wscript.Echo filename & " is " & CStr(DateDiff("d", FileCreatedDate,
CreatedTodayDate)) & " day(s) old."
         'Wscript.Echo "Copying " & filename & " to " & PerflogFolderYesterday
         objFSO1.CopyFile filename, PerflogFolderYesterday & "/"
         relogfiles = relogfiles & PerflogFolderYesterday & "\" & f2.name & " "
      End If
   Next

'Wscript.Echo "Today's files to send to relog: " & relogfiles

command = "relog " & relogfiles & "" -o " & DailyFileName & " -cf " _
  & relogSettingsFileName & " " & relogParms
'Wscript.Echo "Relog command string: " & command

Set WshShell = Wscript.CreateObject("WScript.Shell")
Set execCommand = WshShell.Exec(command)
Wscript.Echo execCommand.StdOut.ReadAll

command = "relog " & DailyFileName & _
  " -o " & SummaryFileName & " -cf " & HistorySettingsFileName _
  & " " & relogSummaryParms
'Wscript.Echo "Relog command string: " & command

Set WshShell = Wscript.CreateObject("WScript.Shell")
Set execCommand = WshShell.Exec(command)
Wscript.Echo execCommand.StdOut.ReadAll

If (objFSO1.FileExists(HistoryFileName)) Then
  command = "relog " & HistoryFileName & " " & SummaryFileName & _
    " -o " & HistoryFileName & " " & relogHistoryParms
  'Wscript.Echo "Relog command string: " & command
  Set WshShell = Wscript.CreateObject("WScript.Shell")
  Set execCommand = WshShell.Exec(command)
  Wscript.Echo execCommand.StdOut.ReadAll
Else
  objFSO1.CopyFile SummaryFileName, HistoryFileName
End If

'Copy the summarized daily file to a Counter Log data consolidation server
'   objFSO1.CopyFile DailyFileName, <somewhere>
```

```
‘Wscript.Echo "Deleting files after processing"
 For Each f2 in fc2
      filename = PerflogFolderToday & "\" & f2.name
      FileCreatedDate = CDate (f2.DateCreated)

     If DateDiff ("d", CreatedTodayDate, FileCreatedDate) = 0 Then
         ‘Wscript.Echo "Skipping the file currently in use: " & filename
       Else
         ‘Wscript.Echo "Deleting " & filename &  " from " & PerflogFolderToday
         objFSO1.DeleteFile(filename)
     End If
  Next

Set fc3 = objAlertsFolder.Files

‘Wscript.Echo "Look for older Alert-generated log files ..."
   For Each f3 in fc3
      ‘Wscript.Echo "Found " & f3.name & " in " & PerflogFolderAlerts
      filename = PerflogFolderAlerts & "\" & f3.name
      FileCreatedDate = CDate (f3.DateCreated)
      ‘Wscript.Echo filename & " is " & CStr(DateDiff("d", FileCreatedDate,
  CreatedTodayDate)) & " day(s) old."

     If DateDiff ("d", FileCreatedDate, CreatedTodayDate) < AlertDaysOld Then
        ‘Wscript.Echo "Skipping recently created Alert Counter Log file: " &
  filename
       Else
         ‘Wscript.Echo "Deleting " & filename &  " from " &
  PerflogFolderToday
         objFSO1.DeleteFile(filename)
     End If
  Next
```

This sample script is designed to be launched automatically by the Performance Logs and Alerts service when smlogsvc closes one daily counter log and opens the next. You can also use the Task Scheduler to launch this script automatically whenever you decide to perform these post-processing steps.

The sections that follow discuss in detail the logic used by the post-processing script, in case you need to customize it.

Script Initialization

The Initialization section of this sample script sets initial values for a number of constants that are used. By changing the initial values that these variables are set to, you can easily change the behavior of the script to conform to your environment without having to alter much of the script’s logic. For example, the following lines set initial values for four string variables that reference the folders containing the counter logs that the script will manage:

```
Const PerflogFolderToday = "C:\Perflogs\Today"
Const PerflogFolderYesterday = "C:\Perflogs\Yesterday"
Const PerflogFolderAlerts = "C:\Perflogs\Alert Logs"
Const PerflogFolderHistory = "C:\Perflogs\History"
```

It assumes that *PerflogFolderToday* points to the folder where the daily counter logs are being written by the Performance Logs and Alerts service. It also assumes that *Perflog-FolderAlerts* points to the folder where counter logs that are triggered by hourly alert scans are written. The script will move yesterday's counter logs from C:\Per-flogs\Today to C:\Perflogs\Yesterday, creating C:\Perflogs\Yesterday if it does not already exist. The script will also delete counter logs in the C:\Perflogs\Alert Logs folder if they are older than three days. Another constant named *AlertDaysOld* contains the aging criteria for the Alert Logs folder. If you prefer to keep diagnostic counter logs that were automatically generated by alert thresholds being tripped for 10 days, simply change the initialization value of the *AlertDaysOld* variable as follows:

```
Const AlertDaysOld = 10
```

The script uses the *WshShell* object to access the built-in *COMPUTERNAME* environment variable. This variable is used to create a file name for the summarized counter log file that relog will create from any detail counter logs that are found. Having the computer name where the counter log originated embedded in the file name makes for easier identification.

```
Set WshShell = CreateObject("Wscript.Shell")
Set objEnv = WshShell.Environment("Process")
LogonServer = objENV("COMPUTERNAME")
DailyFileName = PerflogFolderYesterday & "\" & LogonServer & "." &
  "basic_daily_log" & "." & LogType
'WScript.Echo DailyFileName
```

Following initialization of these and other constants, the script initializes a *FileSystem-Object*, which it uses to perform various file management operations on the designated folders and the counter log files they contain:

```
Set objFSO1 = CreateObject("Scripting.FileSystemObject")
```

The *FileSystemObject* is then used to identify the file folders the script operates on. For example, the following code initializes a variable named *fc1*, which is a collection that contains the files in the C:\Perflog\Yesterday folder:

```
Set objYesterdayFolder = objFSO1.GetFolder(PerflogFolderYesterday)
Set fc1 = objYesterdayFolder.Files
```

Cleaning Up Yesterday's Backup Files

The body of the script begins by enumerating the files in the C:\Perflogs\Yesterday folder where older counter logs are stored. At this point, any files in the C:\Perflogs\Yesterday folder that match the *OldLogType* are deleted. The next section of the script renames any .blg files that it finds in the C:\Perflogs\Yesterday folder with the *OldLogType* suffix, which is appended to the file name using this section of code:

```
For Each f1 in fc1
   If f1.type = PMFileType Then
         filename = PerflogFolderYesterday & "\" & f1.name
      NewName = PerflogFolderYesterday & "\" & f1.name & "." & "bkf"
      objFSO1.MoveFile filename, NewName
   End If
 Next
```

The sample script initializes the value of *OldLogType* to associate it with *Windows Backup File*, which uses a file name suffix of .bkf. You can change this to any value appropriate for your environment. (Alternately, you could parse the file name string to look for this file extension.)

The effect of this processing on the files in the C:\Perflogs\Yesterday folder is that the folder will contain counter logs from yesterday and backup versions of the counter logs used the day before yesterday. This provides a margin of safety that allows you to go back in time at least one full day when something goes awry with any aspect of the daily counter log post-processing procedure.

Managing Yesterday's Daily Counter Logs

Next, the script enumerates the counter log files that exist in the C:\Perflogs\Today folder. This should include the counter log binary file where current measurement data is being written, along with any earlier counter logs that are now available for post-processing. The script determines whether the counter log files in the C:\Perflogs\Today folder are ready for processing by comparing their creation date to the current date. Files in the C:\Perflogs\Today folder from the previous day are then copied to the C:\Perflogs\Yesterday folder using the *DateDiff* function. Meanwhile, the names of the files being copied are accumulated in a string variable called *relogfiles* for use later in the script.

Creating Summarized Counter Logs

After all the older counter logs in the C:\Perflogs\Today folder are processed, the script executes the Relog utility using the *Exec* method of the *WshShell* object:

```
command = "relog " & relogfiles & "-o " & DailyFileName & " -cf " _
  & relogSettingsFileName & " " & relogParms
Set WshShell = Wscript.CreateObject("WScript.Shell")
Set execCommand = WshShell.Exec(command)
```

This creates a binary counter log denoted by the *DailyFileName* string variable, which is summarized to 15-minute intervals.

The line that follows the invocation of the Relog utility gives you access to any output messages that Relog produces, if you remove the comment character and enable the call to *Wscript.Echo*:

```
'Wscript.Echo execCommand.StdOut.ReadAll
```

Once this summarized daily counter log is created using the Relog utility, you will probably want to add a processing step in which you copy the daily log to a counter log consolidation server somewhere on your network. Replace the following comment lines, which reserve a space for this processing, with code that performs the file transfer to the designated consolidation server in your environment:

```
'  Copy the summarized daily file to a Counter Log data consolidation server
'   objFSO1.CopyFile DailyFileName, <somewhere>
```

The script issues a second Relog command to summarize the daily log to 1-hour intervals. Next, it uses Relog again to append this counter log to an historical summary log file in the C:\Perflogs\History folder that is also summarized hourly. At the conclusion of this step, a file named <*computer-name*>.history_performance_log.blg, which contains all the accumulated historical data you have gathered, summarized to 1-hour intervals, is in the C:\Perflogs\History folder. Because this historical summary file will continue to expand at the rate of about 300–500 KB per day, you might want to archive it periodically for long-term storage in another location. For instance, after one year, the historical summarized counter log will grow to approximately 100–200 MB per machine.

Alternatively, at this point in the procedure, you could chose to store the historical, summarized counter log data in a SQL Server performance database. If you decide to use a SQL Server repository for historical counter data, first insert the following initialization code:

```
Const relogSQLParms = "-f SQL -t 4"
Const relogSQLDB = """SQL:PDB!ByHour"""
Const SQLSettingsFileNane = "c:\perflogs\summary-counters-setting-file.txt"
```

Next, replace the last two Relog commands with a single Relog command to insert the hourly counter log data that is generated directly into SQL Server, as follows:

```
command = "relog " & DailyFileName & " -o " & relogSQLDB & " -cf " & SQLSettingsFileNane
& " " & relogSQLParms
```

Tip For maximum flexibility, you might want to build and maintain both summarized data and historical data in binary format logs that are easy to transfer from computer to computer in your network and to a consolidated historical PDB using SQL Server.

The script then contains a comment that instructs you to copy the summarized daily counter log file that was just created to a consolidation server somewhere on your network, where you will use it in your management reporting process:

```
'   objFSO2.CopyFile DailyFileName, <somewhere>
```

Next, the script returns to the C:\Perflogs\Today folder and deletes the older files that were previously copied to C:\Perflogs\Yesterday and processed by Relog.

Managing Counter Logs Automatically Generated by Alerts

Finally, the script enumerates all the files in the C:\Perflogs\Alert Logs folder and deletes any that are older than the value set in the *AlertDaysOld* constant, which by default is set to 3 days. The rationale for deleting older counter logs in the Alert Logs folder is that the crisis has probably passed. Like the C:\Perflogs\Today folder where you are writing daily counter logs, if you neglect to perform some sort of automatic file management on the C:\Perflogs\Alert Logs folder, eventually you are going to run out of disk space on the machine.

Scheduled Monthly Reports and Archiving

Additional relogging can be done on a scheduled basis using the Scheduled Tasks utility. Tasks scheduled to run by a service other than the Performance Logs and Alerts service should use active log files very cautiously. Having an active log file open by another process might prevent the log service from closing it properly, or prevent the command file that is executed after the current log is closed from functioning correctly.

At the end of the month, the log files that were collected each day throughout the month can be consolidated into a single archival summary log file. In most cases, this will provide sufficient detail while not consuming unnecessary disk space.

The end-of-the-month processing consists of two main functions:

1. Reducing the number of samples
2. Consolidating the daily files into a single log file representing the month

For best processing performance, first compress the individual files and then concatenate them. Because Relog tries to join all the input files together before processing them, working with smaller input files makes the resulting join much quicker. Listing 4-13 is an example of typical monthly processing using a command file.

Listing 4-13 Monthly Processing Example

```
Rem ************************************************************************
Rem *  arg 1 is the name of the month that the performance data logs
Rem *  are being compiled for.
Rem *
Rem *  arg 2 is the directory path in which to put the output file
Ren *
Ren *  NOTE: This procedure should not run when a daily log is being
Ren *  processed. It should run after the last daily log of the month has
Rem *  been processed and moved to the SAVE_DIR
Rem *
Rem ************************************************************************
set LOG_DIR <directory path containing these files>
set SAVE_DIR <directory path where daily log files are saved>
set TEMP_DIR <directory path where compressed daily log files are saved>
echo Archiving logs in %SAVE_DIR% at %date% %time%>> %LOG_DIR%\SAMPLE_RELOG.LOG
Rem  compress each daily log and store the output files in the TEMP_DIR
for %a in (%SAVE_DIR%) do %LOG_DIR%\Montlhy_Compress.bat "%a" "%TEMP_DIR%"
if errorlevel 1 goto COMPRESS_ERROR
Rem  concatenate the compressed log files into monthly summary file
relog %TEMP_DIR%\*.blg -o %2\%1.blg
if errorlevel 1 goto RELOG_ERROR
Rem clear out the temp directory to remove the temp files
Del /q %TEMP_DIR%\*.*
Rem  clear out the original files if you want to free up the space
Rem  or you may wish to do this manually after insuring the files were
Rem  compressed correctly
Del /q %SAVE_DIR%\*.*
exit :COMPRESS_ERROR
echo Compress error at %date% %time%>> %LOG_DIR%\SAMPLE_RELOG.LOG
exit
:RELOG_ERROR
echo Relog error at %date% %time%>> %LOG_DIR%\SAMPLE_RELOG.LOG
exit
```

In the command file in Listing 4-13, the log files stored in the SAVE_DIR directory are compressed with another command file and then merged together.

Listing 4-14 Command File Invoked by the Monthly Processing Job

```
Rem ***************************************************************************
Rem *  arg 1 is the filename of the daily performance data log file to
Rem *  compress for subsequent concatenation.
Rem *
Rem *  arg 2 is the directory path in which to put the output file
Ren *
Rem ***************************************************************************
set LOG_DIR <directory path containing these files>
set SAVE_DIR <directory path where daily log files are saved>
set TEMP_DIR <directory path where compressed daily log files are saved>
echo Compressing file: %1 at %date% %time%>> %LOG_DIR%\SAMPLE_RELOG.LOG
Rem  compress each daily log and store the output files in the TEMP_DIR
Relog %1 -config %LOG_DIR%\COMPRESS_CFG.TXT -o %2\%~nx1 >>
%LOG_DIR%\SAMPLE_RELOG.LOG
```

The command file in Listing 4-14 is separate from the main command file so that the file name parsing features of the command interpreter can be used.

After the data collected during the month is consolidated into a condensed summary file, monthly reports can be produced using the summary file. In fact, the same counter log configuration files could be used for a monthly summary of the same data reported in the daily reports. The only difference is in using a different input file—the monthly summary log file instead of the daily performance data log file.

Defining Log Configurations for Multiple Servers

Configuration files can also be used to establish uniform counter log collection procedures across multiple computers. The following configuration file illustrates the basic log settings that might be used in a server configuration consisting of many servers. Note that additional counters can be added if necessary.

```
[name]
Basic Performance Data Log

[sample]
15

[format]
bin

[--max]

[--append]
```

```
[version]
nnnnnn

[runcmd] <insert the full path name of command file to run when this log file is
closed>

[counters]
\Processor(_Total)\% Processor Time
\LogicalDisk(_Total)\% Disk Time
\Memory\Pages/sec
\Network Interface (*)\Bytes Total/sec
```

Using a SQL Server Repository

The SQL Server support for counter log data is flexible and provides a good way to build and maintain a long-term repository of counter log data for both analysis and capacity planning. This section discusses the use of a SQL Server repository for reporting counter log performance data. It discusses the database schema that is employed to store the counter log data. It then describes how to use data mining tools like Microsoft Excel to access counter log data in a SQL Server repository and report on it. Finally, an example is provided that retrieves counter log data from SQL Server using Excel and produces a workload growth–based forecast, performing a statistical analysis of historical usage trends.

When counter log data is stored in a SQL Server repository, it is organized into three database tables. In general, data in database tables is arranged into rows and columns. Rows are equivalent to records in a file, whereas columns represent fields. Each counter log table is indexed using *key* fields. Keys uniquely identify rows in the table and allow you to *select* specific rows directly without having to read through the entire table. Each counter log table contains *foreign keys*, which are the key fields that allow you to link to another, related table. If two tables share a common key field, you can *join* them, which is a logical operation that combines data from two tables into one. The language used to select and join database tables is known as SQL, which is based on relational algebra. Fortunately, you do not have to know how to use the SQL language to access and manipulate the counter log tables inside SQL Server. You can rely on tools like Microsoft Excel, Microsoft Query, and Microsoft Access that allow you to access SQL Server data without ever having to code SQL statements. Nevertheless, when you are using one of these tools to access counter log data stored in SQL Server, it is helpful to have an understanding of how the counter log is organized into database tables.

Although you can store any type of counter log in a SQL Server database, the database is ideal for storing summarized counter log data that is consolidated from multiple machines for longer-term trending, forecasting, and capacity planning. The procedures described in this chapter focus on this type of usage.

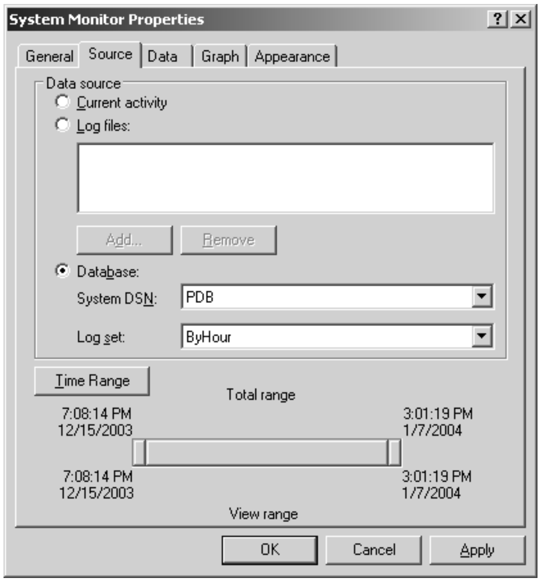
If you are creating a SQL Server repository of summarized counter log data for capacity planning, you will use procedures like the ones described earlier in "Populating the Repository," in which you relog the data to create concise summaries and send the output from the Relog utility to SQL Server. For example, the following Relog command takes one or more daily counter log files summarized to 15-minute intervals, summarizes them even further to 1-hour intervals, and inserts the output into a SQL Server database named PDB:

```
relog <summaryfile-list> -o "SQL:PDB!ByHour" -f SQL -t 4 -cf c:\perflogs\summary-
counters-setting-file.txt
```

The output file specification, *-o "SQL:PDB!ByHour"*, identifies the ODBC connection named PDB and also identifies a log set of the PDB database, called *ByHour*.

Using the System Monitor Console with SQL Server

You can continue to use the System Monitor console to report on counter log data that is stored in a SQL Server performance database. In the Chart view, click the View Log Data toolbar button (which looks like a disk drive), and then select Database as the data source, as illustrated in Figure 4-18.

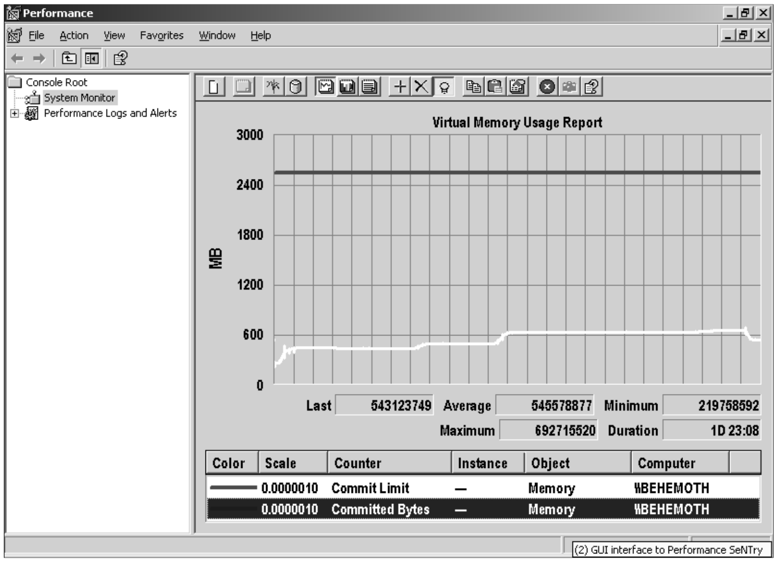


Database: [✓]
System DSN: PDB
Log set: ByHour

Total range: 7:08:14 PM 12/15/2003 – 3:01:19 PM 1/7/2004
View range: 7:08:14 PM 12/15/2003 – 3:01:19 PM 1/7/2004

Figure 4-18 Database selected on the Source tab

Select the System DSN and the Log Set, and then click Apply. At this point, you will be able to create charts and histograms for any of the historical data that is stored in the SQL Server database. For example, Figure 4-19 shows a Chart view of Committed Bytes compared to the machine's Commit Limit over a 3-day span using counter log data retrieved from a SQL Server database.


Performance
File
Action
View
Favorites
Window
Help
Console Root
System Monitor
Performance Logs and Alerts
Virtual Memory Usage Report
3000
2400
1800
1200
600
0
MB
(2) GUI interface to Performance SeNTry

Last	543123749	Average	545578877	Minimum	219758592
		Maximum	692715520	Duration	1D 23:08

Color	Scale	Counter	Instance	Object	Computer
▬	0.0000010	Commit Limit	—	Memory	\\BEHEMOTH
▬	0.0000010	Committed Bytes	—	Memory	\\BEHEMOTH

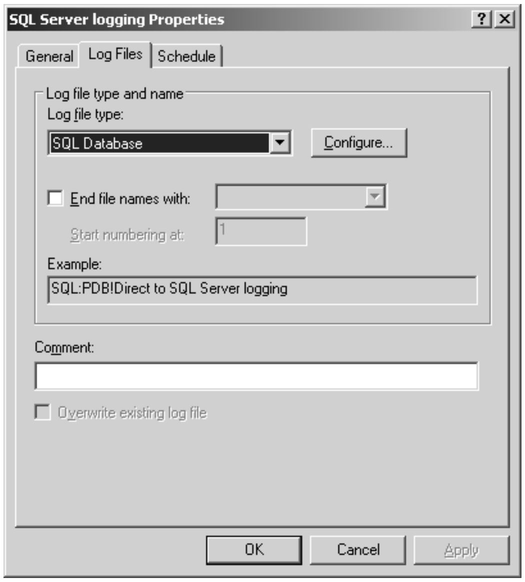
Figure 4-19 A Chart view illustrating memory capacity planning data from SQL Server

How to Configure System Monitor to Log to SQL Server

Although these procedures focus on using SQL Server as a repository of summarized counter log data for capacity planning, it is also possible to direct counter log data to SQL Server when the data is initially created.

1. To configure Performance Monitor to Log to SQL Server from Performance Logs and Alerts, right-click **Counter Logs**, and then click **New Log Settings.** Type a name for this log, and then click **OK**.
2. On the **General** tab, click **Add Objects** to add the objects you want to log, and then click **Add**. Enter the counters that you want to monitor, and then click **Close**. In the **Run As** box, be sure to supply a user name that has the proper credentials to create and store data in the SQL Server database selected. The following permissions are required:
   - ❑ The correct rights to run System Monitor
   - ❑ The correct rights to the SQL Server database (both create and read)

3. Click the Log Files tab, click **SQL Database** in the **Log File Type** list, and then click **Configure**, as illustrated in Figure 4-20. The **Configure SQL Logs** dialog box is displayed.

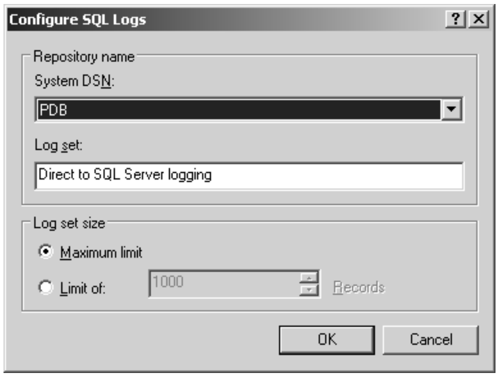


Log file type and name
Log file type: SQL Database
End file names with: [ ]
Start numbering at: 1
Example: SQL:PDB!Direct to SQL Server logging
Comment:
Overwrite existing log file [ ]

Figure 4-20 Configuring the SQL database

Note that using automatic versioning when you are logging counter log data directly to SQL Server is usually not advisable, as illustrated. Versioning will create new counter log sets continuously and increase the size of the CounterDetails table accordingly. This table growth will make using the counter logs stored in the SQL Server database more complicated when you employ query and analysis tools such as Microsoft Excel.

4. In the **System DSN** box, click the DSN that you want to connect to and provide a name for the counter log set you want to log to, as illustrated in Figure 4-21. If this log set already exists, the counter log data will be added to the existing log set. If the log set is new, it will be created when the counter log session is started.


Configure SQL Logs

Repository name

System DSN: PDB

Log set: Direct to SQL Server logging

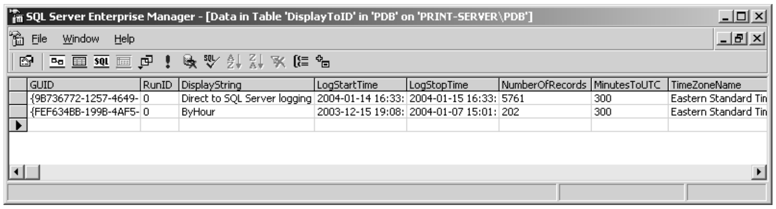
Log set size

(●) Maximum limit

( ) Limit of: 1000 Records

Figure 4-21 Setting the DSN and naming the log set

After you start the counter log session, you can verify that the counter log was created properly inside SQL Server using SQL Server Enterprise Manager. Navigate to the database to which you are logging counter data and access its tables. You can then right-click the **DisplayToID** table, select **Open Table**, **Open All Rows**. You should see a display similar to Figure 4-22.


SQL Server Enterprise Manager - [Data in Table 'DisplayToID' in 'PDB' on 'PRINT-SERVER\PDB']

GUID	RunID	DisplayString	LogStartTime	LogStopTime	NumberOfRecords	MinutesToUTC	TimeZoneName
{9B736772-1257-4649-	0	Direct to SQL Server logging	2004-01-14 16:33:	2004-01-15 16:33:	5761	300	Eastern Standard Tin
{FEF634BB-199B-4AF5-	0	ByHour	2003-12-15 19:08:	2004-01-07 15:01:	202	300	Eastern Standard Tin

Figure 4-22 Using SQL Server Enterprise Manager to verify that a counter log was created properly inside SQL Server

Counter Log Database Schema

When you first use the SQL Server support in either Logman or Relog to create a counter log data repository, three new tables are defined to store the counter log data, as illustrated in Figure 4-17. Counter log data is stored in SQL Server in three inter-linked tables:

- The CounterData table contains individual measurement observations, with one measurement sample per row.
- The CounterDetails table tells you what counter fields are stored in the database, with one set of counter identification information stored per row.
- The DisplayToID table contains information used to identify one or more sets of counter log data stored in the database.

CounterData Table

The CounterData table contains a row for each observation of a distinct counter value measured at a specific time. Each individual counter measurement is stored in a separate row of the table, so you can expect this table to grow quite large.

The columns (or fields) of the CounterData table that you are likely to use most frequently are described in Table 4-5.

Table 4-5 Frequently Used CounterData Columns (Fields)

Column Name	Explanation
CounterID	Part of the primary key to the table; foreign key to the CounterDetails table
CounterDateTime	The start time when the value of this counter was collected, based on Coordinated Universal Time (UTC)
CounterValue	The formatted value of the counter, ready to be displayed
GUID	Part of the primary key to the table; foreign key to the DisplayToID table
RecordIndex	Part of the primary key to the table; uniquely identifies the data sample

The CounterData table also contains the raw performance data values that were necessary to calculate the formatted data value. Deriving the formatted counters might require up to four raw performance data values that are used in the formatted counter value calculation. If available, these raw performance data values are stored in *FirstValueA*, *FirstValueB*, *SecondValueA*, and *SecondValueB*. When you use the Relog utility to summarize data from a SQL Server database, the raw performance counters are put to use. For most other purposes, you can ignore them.

The primary key for this table is a combination of the *GUID*, *CounterID*, and *RecordIndex* fields. That means that a SQL *SELECT* statement identifies a unique row in the CounterTable when you specify a value for the *CounterID*, GUID, and *RecordIndex* fields.

Even though it contains the measurement data for all the counters you are interested in, by itself the CounterData table is not usable. However, once you join the CounterTable with the CounterDetails table that identifies each counter by its *CounterID*, the *CounterData* yields the measurement data you are interested in.

CounterDetails Table

The CounterDetails table associates the *CounterID* that is the key to the CounterData table with fields that identify the counter by name. It stores the counter type, which is required for summarization. CounterDetails provides the fully qualified counter name, which it breaks into a series of fields so that the counter name value can be stored and retrieved readily. For example, to build a fully qualified counter name such as \\Computer\Processor(1)\% Processor Time, the CounterDetails table supplies values for the *ComputerName*, *ObjectName*, *CounterName*, and *InstanceName* columns. The *CounterID* field is the key to the CounterDetails table. The fields listed in Table 4-6 are available in the CounterDetails table.

Table 4-6 Fields Available in the CounterDetails Table

Column Name	Explanation
CounterID	A unique identifier; foreign key to the CounterData table
MachineName	The machine name portion of the fully qualified counter name
ObjectName	The object name portion of the fully qualified counter name
CounterName	The counter name portion of the fully qualified counter name
InstanceName	The counter instance name portion of the fully qualified counter name, if applicable
InstanceIndex	The counter instance index portion of the fully qualified counter name, if applicable
ParentName	The name of the parent instance, if applicable
ParentObjectID	The parent object ID, if applicable
CounterType	The counter type for use in summarization
DefaultScale	The default scaling factor to be applied when the counter value is charted using the System Monitor console

When you join the CounterDetails table with the CounterData table in a SQL query, you can associate the measurement data values stored in CounterData with the proper counter name identification. Storing the counter name details once in a separate table is an example of database *normalization*, and saves a great of deal of disk space in the database tables. It does require an extra step during reporting, namely, that you must join the two tables first. But this is very easy to do using the rich data mining tools that are provided in programs like Microsoft Excel and Microsoft Query, which is illustrated in the "Querying the SQL Performance Database" section.

Note If you intend to code SQL statements to retrieve data from counter logs stored in SQL Server, you must first join the CounterData and CounterDetails tables, as in the following generic example:

```
Select * from CounterData, CounterDetails where  CounterData.CounterID
= 50
```

DisplayToID Table

The DisplayToID table serves several purposes. Primarily, it associates the character string you specified when you originally created the SQL Server counter log database with a GUID that can serve as a unique identifier for the counter log database. This character string is for reference only. For example, in the following Relog command, the character string "*ByHour*" identifies the log set of the PDB database that is referenced:

```
relog <summaryfile-list> -o "SQL:PDB!ByHour" -f SQL -t 4 -cf
c:\perflogs\summary-counters-setting-file.txt
```

This character string is stored in the DisplayToID table in a field called *DisplayString*, which is automatically associated with a GUID to create a unique identifier that can be used as a key to both this table and the CounterData table. The fields listed in Table 4-7 are in the DisplayToID table.

Table 4-7 Fields in the DisplayToID Table

Column Name	Explanation
GUID	A unique identifier; also a foreign key to the CounterData table. Note this *GUID* is unique; it is not the same GUID stored in the registry at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SysmonLog\Log Queries\.
DisplayString	Equivalent to a log file name; the character string specified with the *-o* subparameter following the exclamation point character (!) in the SQL Server database output file reference.
LogStartTime	Log start and end times in *yyyy-mm-dd hh:mm:ss:nnn* format. Note that the datetime values in the *LogStartTime* and *LogStopTime* fields allow you establish the time range for the log without scanning the entire CounterData table.
LogStopTime	
NumberOfRecords	The number of *CounterData* table rows associated with this log file.
MinutesToUTC	Add this value to *CounterData.CounterDateTime* to convert the *CounterData* datetime field to local time.
TimeZoneName	The name of the time zone where the *CounterData* was gathered.
RunID	A reserved field used for internal use only.

The Time conversion data present in the DisplayToID table allows you to convert the *datetime* field in the CounterData table to local time.

If you now issue a different Relog command, such as the one that follows, separate rows will be created in the DisplayToID table—one set of rows that store the ByHour summary log and another set that corresponds to the ByShift summary log:

```
relog <summaryfile-list> -o "SQL:PDB!ByShift" -f SQL -t 32 -cf
c:\perflogs\summary-counters-setting-file.txt
```

Counter measurement data from both logs will be stored together in the same CounterData table. A unique GUID is generated for each log. This GUID identifies the log set uniquely and is used as the key to the DisplayToID table. The GUID also serves as foreign key to the CounterData table so that the rows associated with each log can be distinguished.

Note If, as in the example just discussed, multiple logs are stored in a single database, you must join all three tables first before attempting to utilize the counter log data in a report. For a simple example, the following *SELECT* statement will join all three tables and retrieve information from just the ByHour counter log:

```
Select * from CounterData, CounterDetails, DisplayToID where
DisplayToID.DisplayString = ‘ByHour' and CounterData.GUID =
DisplayToID.GUID and CounterData.CounterID = 50
```

Fortunately, the SQL Server Query Optimizer will ensure that this complex, nested *SELECT* statement executes as efficiently as possible against the database.

To keep your SQL statements from growing complex as a result of maintaining multiple logs in a single SQL Server database, you can define additional counter log databases in the same instance of SQL Server or in separate instances of SQL Server.

Querying the SQL Performance Database

Once Performance Monitor data is stored in one or more SQL Server databases, you can access it for reporting and analysis in a variety of ways. You can, for example, use the Relog command-line utility to access performance data stored data in a SQL Server log set and export it to a .csv file. You are also free to develop your own tools to access and analyze the counter log data stored in SQL Server databases. Finally, you might want to take advantage of the data mining capabilities built into existing tools like Microsoft Excel, which can be used to query SQL Server databases and analyze the data stored there. This section will walk you step by step through a typical SQL Server counter log database access scenario. This scenario analyzes the ByHour counter log file example using Microsoft Excel to retrieve the counter log data stored there.

More Info For more information about developing your own tools to access and analyze the counter log data stored in SQL Server databases, see the topic entitled "Using the PDH Interface" in the SDK documentation available at http://msdn.microsoft.com/library/en-us/perfmon/base/using_the_pdh_interface.asp.

To use the Excel database access component to access counter log data stored in SQL Server, you follow two simple steps. The first step is to access the *CounterDetails* table to generate a list of *CounterIDs* and counter names. If only one counter log file is stored in the specified SQL Server database, a simple SQL *SELECT* operation returns all rows:

```
USE PDB
SELECT * FROM CounterDetails
```

However, if multiple counter logs are stored in a single database, you will need to select the appropriate *DisplayString* in the *DisplayToID* table and perform a join with *CounterDetails* on the corresponding GUID:

```
USE PDB
SELECT * from CounterDetails, DisplayToID where DisplayToID.DisplayString = ‘ByHour'
and CounterData.GUID = DisplayToID.GUID
```

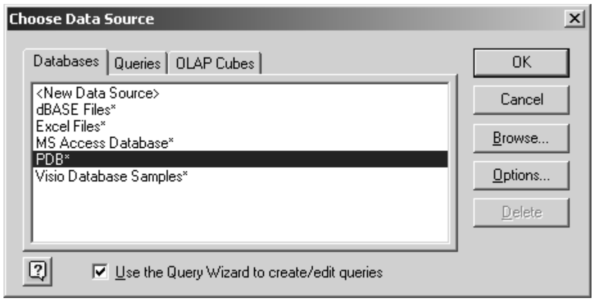
Creating a worksheet first that contains all the information from *CounterDetails* allows you to determine which counter IDs to use in subsequent selections from the *CounterData* table. Because the *CounterData* table can grow quite large, you want to avoid any operation that involves scanning the entire table. To select specific rows in the *CounterData* based on *CounterIDs*, you must first determine the correlation between the *CounterID* field, which is a foreign key to the *CounterData* table, and the *Object:Counter:Instance* identification information, which is stored in *CounterDetails*.

When you use Excel, you do not need to code any SQL statements to execute queries of this kind against the counter log database. Excel supports Microsoft Query, which generates the appropriate *SELECT* statement as you step through a Query generation wizard with a few mouse clicks, as illustrated in the next procedure.

Here is the procedure for using Excel to access a counter log file stored in a SQL Server database. It uses the PDB!ByHour counter log file example that has been referenced throughout this chapter.

To access a Counter Log file stored in SQL Server

1. Start with an empty Excel worksheet and workbook. From the **Data m**enu, select **Get External Data**, **New Database Query**. This invokes the Microsoft Query Wizard.
2. In the **Choose Data Source** dialog box, click the **Databases** tab, shown in Figure 4-23. Select your defined ODBC connection to the counter log SQL Server database.

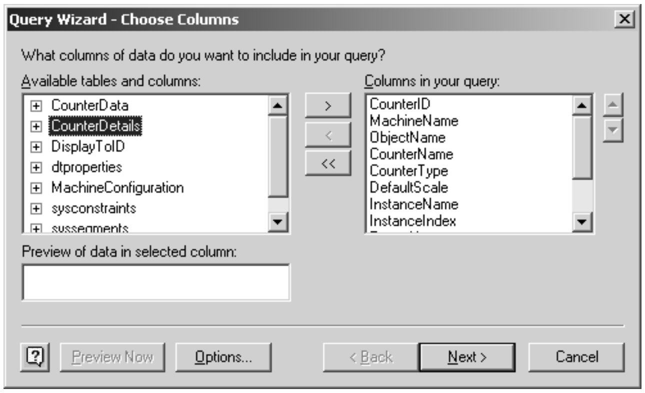


<New Data Source>
dBASE Files*
Excel Files*
MS Access Database*
PDB*
Visio Database Samples*

☑ Use the Query Wizard to create/edit queries

Figure 4-23 The Databases tab in Choose Data Source

3. Make sure that the **Use The Query Wizard To Create/Edit Queries** check box is selected and click **OK**. The **Choose Columns** dialog box is displayed.
4. In the **Available Tables And Columns** section, select the *CounterDetails* table. All the columns from *CounterDetails* appear in the **Columns In Your Query** list, as illustrated in Figure 4-24. Click **Next**.



What columns of data do you want to include in your query?

Available tables and columns:

CounterData
CounterDetails
DisplayToID
dtproperties
MachineConfiguration
sysconstraints
syssegments

Columns in your query:

CounterID
MachineName
ObjectName
CounterName
CounterType
DefaultScale
InstanceName
InstanceIndex

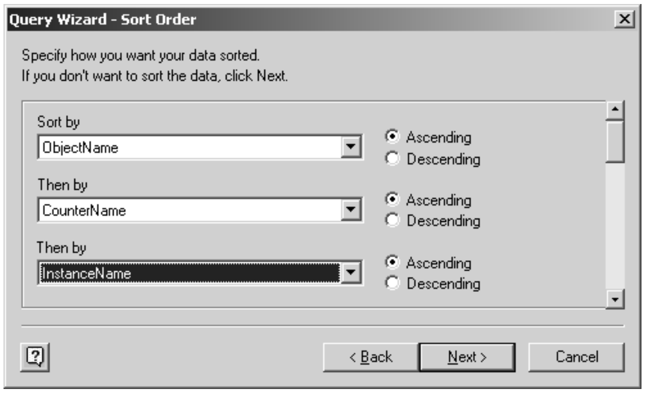
Preview of data in selected column:

Figure 4-24 Displaying the columns from the CounterDetails query

5. Skip the next step of the Query Wizard in which you would normally enter your filtering criteria to build the SQL *SELECT* statement's *WHERE* clause. Because the information you are retrieving from the *CounterDetails* table is intended to act as a data dictionary for all subsequent requests, unless the result set of rows

is too large for Excel to handle conveniently, there is no reason to filter this request.

Optionally, you can select a sort sequence for the result set by specifying the fields in any order that will help you find what you want fast. For example, if you want to gather data from multiple machines, sort the result set by *ObjectName*, *CounterName*, and *InstanceName*, and *MachineName*, as illustrated in Figure 4-25. If you are focused instead on the measurements from a single machine, sort by *MachineName*, followed by *ObjectName*, *CounterName*, and *InstanceName*.

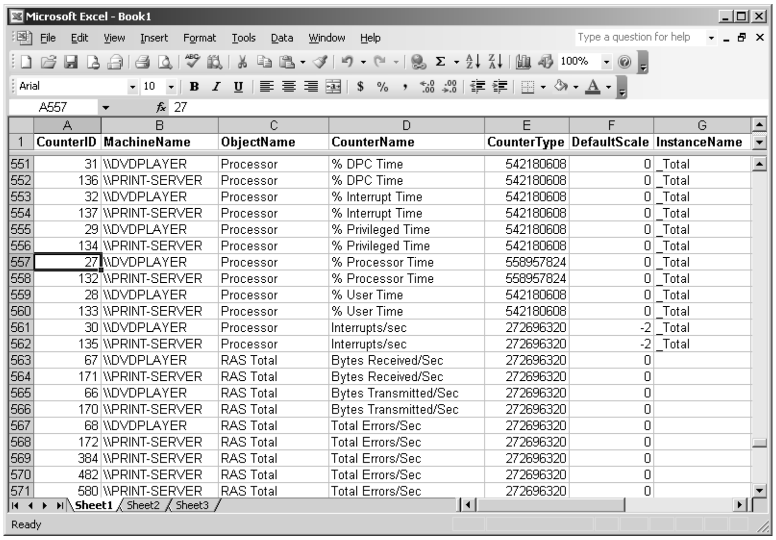


Specify how you want your data sorted.
If you don't want to sort the data, click Next.

Sort by: ObjectName — [✓ Ascending] Descending
Then by: CounterName — [✓ Ascending] Descending
Then by: InstanceName — [✓ Ascending] Descending

Figure 4-25 Gathering data from multiple machines

6. Select **Return Data To Microsoft Excel** and click **Finish**. When the query runs, you will have an Excel worksheet that looks similar to Figure 4-26.



A557 27

	A	B	C	D	E	F	G
1	CounterID	MachineName	ObjectName	CounterName	CounterType	DefaultScale	InstanceName
551	31	\\DVDPLAYER	Processor	% DPC Time	542180608	0	_Total
552	136	\\PRINT-SERVER	Processor	% DPC Time	542180608	0	_Total
553	32	\\DVDPLAYER	Processor	% Interrupt Time	542180608	0	_Total
554	137	\\PRINT-SERVER	Processor	% Interrupt Time	542180608	0	_Total
555	29	\\DVDPLAYER	Processor	% Privileged Time	542180608	0	_Total
556	134	\\PRINT-SERVER	Processor	% Privileged Time	542180608	0	_Total
557	27	\\DVDPLAYER	Processor	% Processor Time	558957824	0	_Total
558	132	\\PRINT-SERVER	Processor	% Processor Time	558957824	0	_Total
559	28	\\DVDPLAYER	Processor	% User Time	542180608	0	_Total
560	133	\\PRINT-SERVER	Processor	% User Time	542180608	0	_Total
561	30	\\DVDPLAYER	Processor	Interrupts/sec	272696320	-2	_Total
562	135	\\PRINT-SERVER	Processor	Interrupts/sec	272696320	-2	_Total
563	67	\\DVDPLAYER	RAS Total	Bytes Received/Sec	272696320	0	
564	171	\\PRINT-SERVER	RAS Total	Bytes Received/Sec	272696320	0	
565	66	\\DVDPLAYER	RAS Total	Bytes Transmitted/Sec	272696320	0	
566	170	\\PRINT-SERVER	RAS Total	Bytes Transmitted/Sec	272696320	0	
567	68	\\DVDPLAYER	RAS Total	Total Errors/Sec	272696320	0	
568	172	\\PRINT-SERVER	RAS Total	Total Errors/Sec	272696320	0	
569	384	\\PRINT-SERVER	RAS Total	Total Errors/Sec	272696320	0	
570	482	\\PRINT-SERVER	RAS Total	Total Errors/Sec	272696320	0	
571	580	\\PRINT-SERVER	RAS Total	Total Errors/Sec	272696320	0	

Figure 4-26 Data returned to an Excel worksheet

Scan this Excel worksheet and look for the *CounterIDs* that are of interest in the next procedure, which will retrieve counter log measurement data from the *CounterData* table.

To retrieve Counter Log measurement data from the *CounterData* table

1. Repeat the procedure titled "To access a counter log file stored in SQL Server." Start with a blank worksheet. Select **Get External Data**, **Edit Query** from Excel's **Data** menu. This invokes the Microsoft Query Wizard with the settings from the last SQL Query intact.
2. This time you want to execute a query that will join the *CounterData* and *CounterDetail* tables. To the previous query, add the columns from the *CounterData* table you want to see (for example, *CounterData* and *CounterDateTime*), as shown in Figure 4-27.

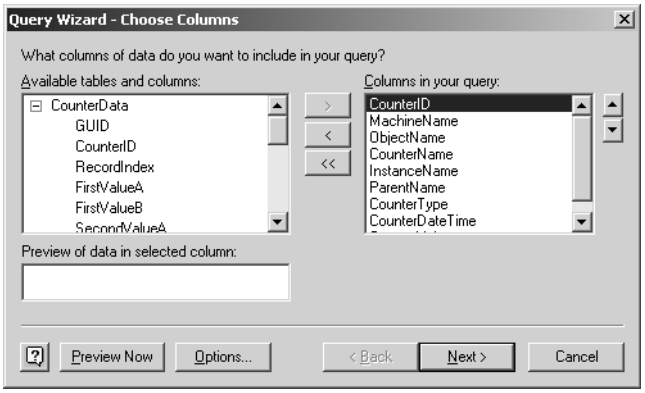
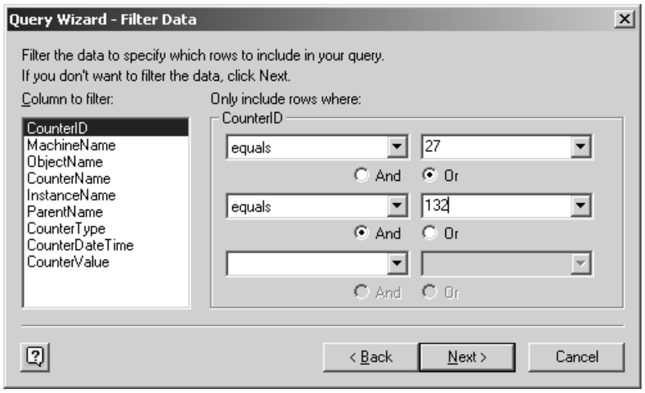


Figure 4-27 Choosing columns to include in a query

3. Filter on specific *CounterIDs*; in this example, select only *CounterData* rows in which the *CounterID* equals 27 or 132, which in this case (Figure 4-28) represents instances of Processor(_Total)\% Processor Time counter from two different machines.


Query Wizard - Filter Data

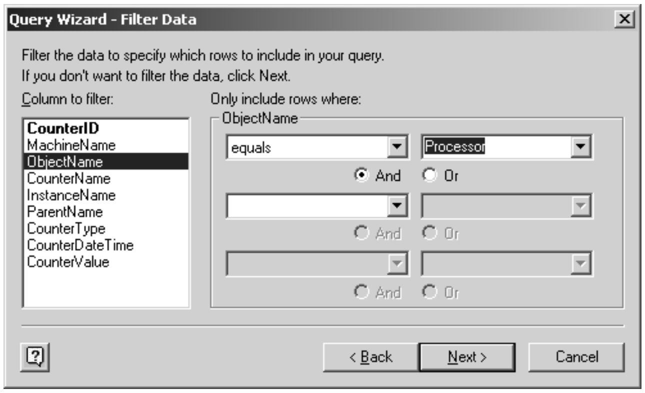
Filter the data to specify which rows to include in your query.
If you don't want to filter the data, click Next.

Column to filter: CounterID, MachineName, ObjectName, CounterName, InstanceName, ParentName, CounterType, CounterDateTime, CounterValue

Only include rows where:
CounterID
equals 27
( ) And [✓ Or]
equals 132
[✓ And] ( ) Or

Figure 4-28 Including data from two different machines

Or, for example, for a generic Processor\% Processor Time query that will retrieve counter data about all instances of the Processor object that are stored in the counter log database, filter on values of *Processor* in the *ObjectName* field, as illustrated in Figure 4-29.


Query Wizard - Filter Data

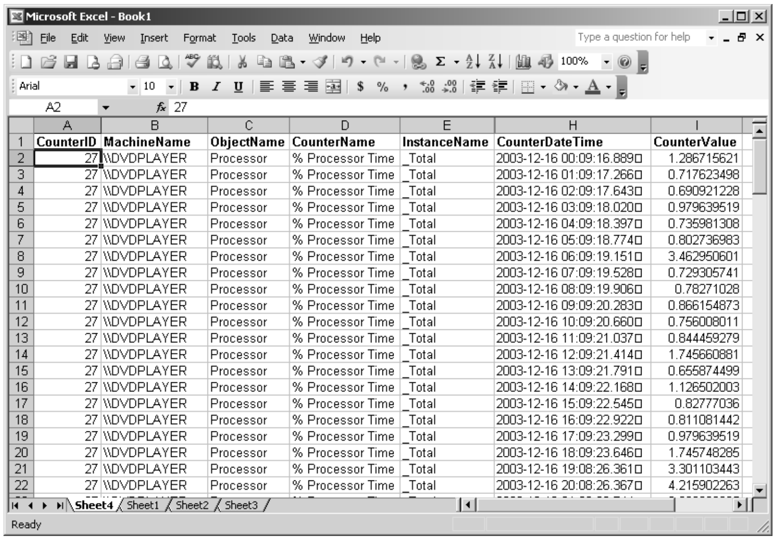
Filter the data to specify which rows to include in your query.
If you don't want to filter the data, click Next.

Column to filter: CounterID, MachineName, ObjectName, CounterName, InstanceName, ParentName, CounterType, CounterDateTime, CounterValue

Only include rows where:
ObjectName
equals Processor
[✓ And] ( ) Or

Figure 4-29 Filtering on values of Processor in the ObjectName field

4. Microsoft Query will generate a SQL *SELECT* statement according to these specifications and execute it, returning all the rows that meet the selection criteria and placing the result set of columns and rows inside your spreadsheet. The result will appear similar to Figure 4-30.



	A	B	C	D	E	H	I
1	CounterID	MachineName	ObjectName	CounterName	InstanceName	CounterDateTime	CounterValue
2	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 00:09:16.889□	1.286715621
3	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 01:09:17.266□	0.717623498
4	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 02:09:17.643□	0.690921228
5	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 03:09:18.020□	0.979639519
6	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 04:09:18.397□	0.735981308
7	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 05:09:18.774□	0.802736983
8	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 06:09:19.151□	3.462950601
9	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 07:09:19.528□	0.729305741
10	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 08:09:19.906□	0.78271028
11	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 09:09:20.283□	0.866154873
12	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 10:09:20.660□	0.756008011
13	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 11:09:21.037□	0.844459279
14	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 12:09:21.414□	1.745660881
15	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 13:09:21.791□	0.655874499
16	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 14:09:22.168□	1.126502003
17	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 15:09:22.545□	0.82777036
18	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 16:09:22.922□	0.811081442
19	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 17:09:23.299□	0.979639519
20	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 18:09:23.646□	1.745748285
21	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 19:08:26.361□	3.301103443
22	27	\\DVDPLAYER	Processor	% Processor Time	_Total	2003-12-16 20:08:26.367□	4.215902263

Figure 4-30 The Excel Workbook For SQL PDB Reporting & Alalysis.xls

5. Finally, execute another query on the *DisplayToID* table to gain access to the *MinutesToUTC* field for this counter log so that you can adjust the *CounterData CounterDateTime* to local time, if necessary.

Capacity Planning and Trending

Capacity planning refers to processes and procedures designed to anticipate future resource shortages and help you take preventative action. Because preventative actions might include acquiring more hardware to handle the workload, computer capacity planning is closely tied to the budget process and has a strong financial component associated with it. Many successful IT organizations make it a practice to coordinate major hardware acquisition with resource planners.

Capacity planners rely on historical information about computer resource usage taken from counter logs. Their planning horizon often encompasses decisions like whether to build or extend a networking infrastructure or build a new computing site, which would involve major capital expenditures that need approval at the highest levels of your organization. As you can see, capacity planners are typically not concerned with day-to-day computer performance problems (although some planners wear many hats). Rather, they need information that can support strategic planning decisions, and they need the performance databases that can supply it.

Capacity planners also receive input on growth, expansion plans, acquisitions, and other initiatives that might affect the computer resource usage that supports various business functions. Capacity planners must understand the relationship between business factors that drive computer resource usage. Historical information that shows how these business drivers operated in the past often plays a key role in predicting the future.

To cope with anticipated shortages of critical resources, capacity planners prepare a forecast of future computer resource requirements and their associated costs. This forecast normally takes into account historical data on resource usage, which is kept in the performance database. The computer capacity planners then factor in growth estimates that are supplied by business planners. Like any prognostication about the future, an element of guesswork is involved. However, by relying on extrapolations from historical trends that are derived from empirical measurement data, capacity planners strive to make their forecasts as accurate as possible.

In this section, using counter log data on resource usage to assist in long-range computer capacity planning is discussed. It also discusses basic statistical techniques for performing workload forecasting using the resource data you have gathered. It will provide guidelines for establishing a useful capacity planning function in your organization.

Organizing Data for Capacity Planning

The data of interest to capacity planners includes:

- The utilization of major resource categories, including processors, disks, network, and memory
- The throughput and transaction rates of key server applications

The measurement data best suited for capacity planning largely overlaps with the recommendations made earlier for management reporting in the section entitled "Sample Management Reports." This means that that the counter log data you gather to produce those reports will also feed a longer-term capacity planning process.

Because capacity planning focuses on longer-term usage trends, it deals exclusively with summarized data and is usually limited to resources and workloads that have a major financial impact on information technology (IT) costs and services. Planning horizons span months and years, so you must accumulate a good deal of historical data before you can reliably use it to make accurate predictions about the future.

Tip As a general rule, for every unit of time that you have gathered accurate historical data, you can forecast a future trend line that is one-half that length of time. For example, to forecast one year of future workload activity with reasonable accuracy, you should have amassed two years of historical data.

Because you can accumulate data points faster, building a capacity planning database initially that consists of measurement data organized for weekly reporting is a good place to start. (Data organized monthly is usually more attuned to financial planning cycles, which makes it more useful in the long run.) In the example discussed here, measurement data has been organization by week for an entire year, allowing you to produce a workload forecast for the next six month period. As you accumulate more and more measurement data, over time you may prefer to shift to monthly or even quarterly reporting.

Accumulating measurement data from counter logs into a performance database using the procedures outlined in this chapter is only the first step in providing a decision support capability for computer resource capacity planning. Both data mining and data analysis need to be performed outside the scope of these automated procedures. Data analysis in particular seeks to uncover any patterns that would seriously undermine the accuracy of a capacity forecast based purely on statistical techniques. Some of the key considerations include:

- **Identifying periods of peak load** Summarization tends to smooth data over time, eliminating the most severe peaks and values, as illustrated in Chapter 1, "Performance Monitoring Overview." Because performance problems are most acute during periods of peak load, an important function of capacity planning is identifying these periods of peak load. Machines and networks should be sized so that performance is acceptable during peak periods of activity that occur with predictable regularity. In the process of summarizing data for capacity planning purposes, attempt to identify periods of peak usage so that requirements for peak load processing can also be understood. In the next example examined in this chapter, workload activity was summarized over a prime shift interval of several hours involving a five-day work week. In addition to calculating average utilization over that extended period, the peak level of activity for any one- hour processing window within the summarization interval was retained. As illustrated, analysis of the peak workload trend, in comparison to the trend in the smoothed average data, is usually very instructive.

- **Cleansing the performance database of anomalies and other statistical outliers** In its original unedited form, the performance database is likely to contain many measurement data points made during periods in which the operational environment was irregular. Consider, for example, the impact on measurement data over an extended time period in which one of the application programs you tracked was permitted to execute undetected in an infinite processor loop. This creates a situation where the measured processor utilization statistics greatly overestimate the true workload. So that future projections are not erroneously based on this anomalous measurement data, these statistical outliers need to be purged from the database. Instead of deleting these observations and diluting the historical record, one option is to replace outlying observations with a more representative calculation based on a moving average.
- **Ensuring that the historical record can account for seasonal adjustments and other predictable cycles of activity that contribute to variability in load** Many load factors are sensitive to both time and date. To understand the full impact of any seasonal factors that contribute to variability in the workload, it is important to accumulate enough data about these cycles. For example, if your workload tends to rise and fall predictably during an annual cycle, it is important to accumulate several instances of this cyclic behavior.

After the data points in the capacity planning database are analyzed and statistical anomalies are removed, you can safely subject the database to statistical forecasting techniques. Figure 4-31 illustrates this approach using measurement data on overall processor utilization accumulated for 52 weeks on a 4-way system that is used as a primary mail server. The time series plot shows average weekly utilization of the processors during the prime shift, alongside the peak hour utilization for the summarization interval.

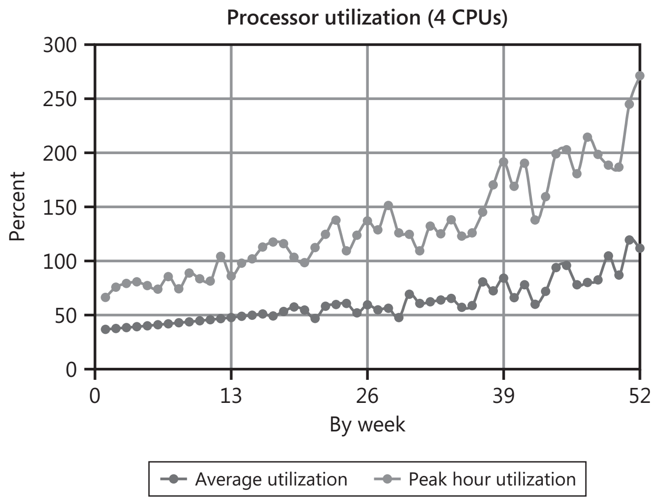


Figure 4-31 Processor utilization data accumulated for 52 weeks on a 4-processor system

Overall utilization of the four processors was less than 50 percent on average across the week at the outset of the tracking period, but rose to over 100 percent utilization (one out of the four processors was busy on average during the weekly interval) by the end of the one-year period. Looking at peak hour utilization over the interval reveals peak hour utilization growing at an even faster rate. In the next section, statistical forecasting techniques are used to predict the growth of this workload for the upcoming half-year period.

Notice in Figure 4-31 that by the end of the one-year period, peak hour utilization is approaching 300 percent (three out of four processors are continuously busy during the interval), which suggests a serious resource shortage is looming. This resource shortage will be felt initially only during peak hour workloads. But as the upward growth trend continues, this resource shortage will impact more and more hours of the weekly prime processing shift. Statistical forecasting techniques can now be applied to predict when in the future this out-of-capacity condition will occur.

Tip To justify an upgrade, you might need to supplement simple measurements of resource utilization with empirical data that suggests performance degradation is occurring when the resource saturates. In the case of a mail server, for example, evidence that Microsoft Exchange Server message delivery queue lengths are growing, or that average message delivery times are increasing, bolsters an argument for adding more processing resources *before* the resource shortage turns critical.

Forecasting Techniques

Capacity planners use forecasting techniques that are based on statistical models. The simplest techniques are often the most effective, so unless you are competent to use advanced statistical techniques, you should stick with basic methods such as *linear regression*. Linear regression derives a straight line in the form of an equation, $y=mx+b$, based on a series of points that are represented as (x,y) coordinates on a Cartesian plane. The regression formula calculates the line that is the *best fit* to the empirical data. (There is one and only one such line that can be calculated for any series of three or more points.) Because the x values of each of the (x,y) coordinates used in this regression line calculation represent time values, the series of points (x,y) is also known as a *time series*.

Note Just because a regression procedure derives a line that is the *best fit* to a series of (x,y) coordinates does not mean that the line is a *good fit* to the underlying data that is suitable for forecasting. Linear regression models also produce *goodness-of-fit* statistics that help you determine whether the regression line is a good fit to the underlying data. The most important goodness-of-fit statistic in linear regression is the *correlation coefficient*, also known as r^2.

Forecasts Based on Linear Regression

The Excel function *LINEST* uses linear regression to derive a line from a set of (x,y) coordinates, produce a trendline forecast, and also return goodness-of-fit statistics, including r^2. In the following example, Excel's *LINEST* function is used to generate a linear regression line that could be used for forecasting. Excel also provides a linear regression–based function called *TREND* that simply returns the forecast without providing any of the goodness-of-fit statistics. If you do not feel competent to interpret the linear regression goodness-of-fit statistics, the *TREND* function will usually suffice.

More Info For more information about the use of the *LINEST* function, see Help in Microsoft Excel.

Figure 4-32 illustrates the use of the Excel *TREND* function to forecast workload growth based on fitting a linear regression line to the historical data. In this chart of the actual measured values of the workload and the 26-week forecast, the historical data is shown using a heavy line, whereas the linear trend is shown as a dotted line. In an Excel chart, this is accomplished by creating an xy-scatterplot chart type based on the historical data, and then adding the forecasting time series to the plot as a new series. Making the forecast trend a separate data series from the actual history data makes it possible to use formatting that clearly differentiates each on the chart.

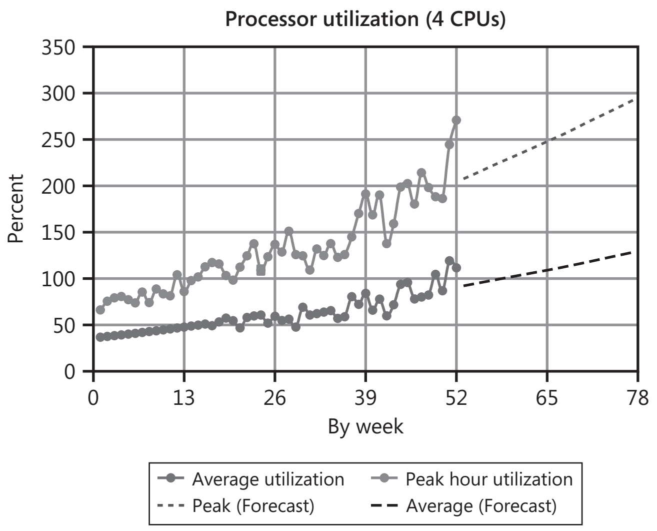


Figure 4-32 A chart showing both actual historical data and forecasted future data points

If you use the Excel *LINEST* function that returns goodness-of-fit statistics, the r^2 correlation coefficient for the linear regression line that was used as the basis of the forecast is available. In this instance, an r^2 value of 0.80 was calculated, indicating that the regression line can "explain" fully 80 percent of the variability associated with the underlying data points.

Note The r^2 correlation coefficient ranges from 0, which means there is no significant correlation among the observations, to 1, which means all the observed data points fall precisely on the regression line and the fit is perfect. For a more elaborate discussion of the linear regression goodness-of-fit statistics, see the Microsoft Excel Help for the *LINEST* function or any good college-level introductory text on statistics.

This would be regarded as a relatively weak correlation for measurement data acquired under tightly controlled circumstances—for example, in drug testing effectiveness trials, in which people's lives are at risk. However, for uncontrolled operational environments like most real-world computer workloads, it is a strong enough correlation to lend authority to any forecasts based on it. In the uncontrolled operational environments of computers, you are fortunate when you can observe correlation coefficients of 0.75 or higher, and even then only when the underlying measurement has been thoroughly scrubbed.

Nonlinear Regression Models

Workload growth trends are often nonlinear, causing forecasts based on linear models to underestimate actual growth. That is because many growth processes are cumulative, operating on both the base workload and the growth portion of the workload, just like compounded interest. Figure 4-33 is a variation of Figure 4-31 in which the Add Trendline feature of the Excel (x,y) scatterplot is employed to plot the linear regression line that the *TREND* function calculates against the underlying data. For the Peak Hour Utilization data, the trend line tends to be above the actual values in the first half of the time series and below the actual values toward the end of the chart. This is often a sign of a nonlinear upward growth trend.

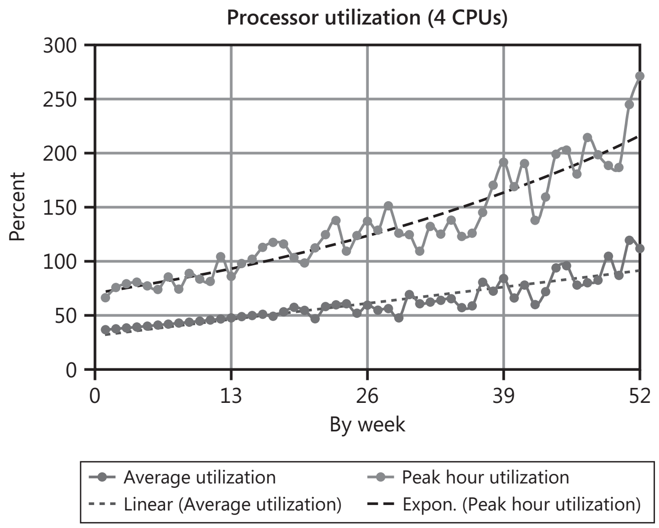


Figure 4-33 A chart variation showing a nonlinear trendline that can be used in forecasting

Because Excel also supports nonlinear regression models, it is easy to compare linear and nonlinear models to see which yield better goodness-of-fit statistics for the same set of data points. *LOGEST* is the nonlinear regression function in Excel that corresponds to *LINEST*. Excel also provides a nonlinear regression–based function called *GROWTH* that returns only the forecast, without providing any of the goodness-of-fit statistics. If you do not feel competent to interpret the nonlinear regression goodness-of-fit statistics, the *GROWTH* function might suffice.

Figure 4-34 adds an exponential growth trendline calculated based on the peak hour utilization measurements, forecasting peak hour utilization that diverges significantly from the linear estimate. Comparing goodness-of-fit statistics for the two models, *LINEST* reports an r^2 of 0.84, whereas *LOGEST* reports an even higher r^2 value of 0.90. The exponential growth model is evidently a better fit to the underlying data. The exponential growth trend predicts that the machine will be running at its absolute CPU capacity limit by week 80, whereas the linear estimate suggests that saturation point might not be reached for another six months.

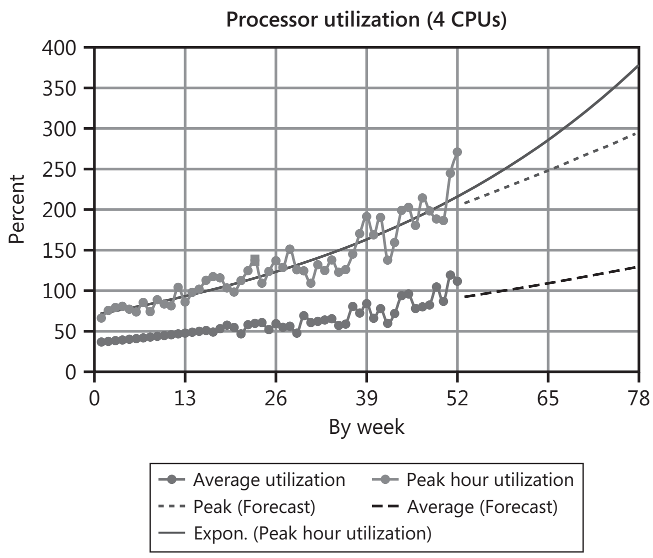


Figure 4-34 An exponential growth trend added to the chart

In this example, the nonlinear growth forecast for peak hour utilization is the safer bet. The goodness-of-fit statistics recommend the nonlinear model as the better fit to the underlying data. The nonlinear trendline also forecasts a serious out-of-capacity condition six months sooner than the linear estimate. Budget impact, technical feasibility, and other considerations might also bear on a decision to relieve this capacity constraint during the next six-month period, sometime before the saturation point is reached.

The Problem of Latent Demand

Once resources saturate, computer capacity limits will constrain the historical workload growth trends that are evident in this example. The measurements taken during periods in which resource constraints are evident will show the historical growth trend leveling off. In many instances, you will find that the historical growth trend will resume once the capacity constraint that is dampening growth is removed. In other instances, demand for new processing resources might simply have leveled off on its own. This is known in capacity planning circles as the problem of *latent demand*. When a resource constraint is evident in the measurement data, there is normally no way to determine definitively from the historical record alone whether latent demand exists or the workload growth has tailed off of its own accord.

In forecasting, what to do about the possibility of latent demand is complicated by evidence that users alter their behavior to adapt to slow systems. End users who rely on transaction processing systems to get their work done, will adapt to a condition of shortage that slows their productivity. Assuming suitable facilities are available, they tend to work smarter so that their productivity does not suffer as a consequence of the resource shortage. One example of this sort of adaptation is the behavioral change in end users who use search engines. When search engine response time is very quick, a user is more likely to scroll through pages and pages of results, one Web page at a time. However, if response time is slow, end users are more likely to use advanced search capabilities to narrow the search engine result set, thus boosting their productivity while using the application.

Counter Log Scenarios

This chapter has focused on uniform counter log data collection, summarization, and reporting procedures that support both performance management and capacity planning, and that can scale effectively to even the largest environments. Still, these model procedures might not be ideal for every situation. This section discusses alternative ways to approach counter log data collection.

In particular, the model counter log data collection procedures described here rely on gathering local counter data and writing counter log files to a local disk. Other counter log scenarios exist as well and are explained in this section. You can use performance data logging and analysis tools to gather both local and remote performance data. You can store the data you collect in a file on the local computer you are monitoring, or write to a file stored on a remote computer. Where to monitor the data from and log the data to depends on your environment and how you want to store and process the logged data.

When you encounter a performance problem and need to initiate a counter log session promptly to identify the problem in real time, you cannot always rely on logging local counters to a local disk. You might have difficulty gaining physical access to the machine experiencing the problem, and you might not be able to use the Remote Desktop facility to gain access either. In situations like this, you must rely instead on facilities to gather counter logs remotely. Remote counter log sessions and their special performance considerations are discussed in this section.

Logging Local Counters

When you specify the specific objects and counters you want to gather, you have the option to gather these counters from only the local computer or to use the Remote

Registry service to gather them from a remote machine somewhere on the network. For efficiency, gathering counter log data from the local computer only is usually the recommended approach. However, you would be forced to gather measurement data from a remote machine in real time to diagnose a problem that is currently happening. Considerations for gathering data from remote machines in real time for problem diagnosis are discussed later in "Using the Performance Console for Remote Performance Analysis."

Logging data from only the local machine is more efficient because of the architecture of the Performance Monitoring facility in Windows Server 2003. (This architecture was discussed extensively in Chapter 2, "Performance Monitoring Tools.") Figure 4-35 shows a simple view of this architecture and focuses on a single aspect of it—namely, the facilities used to identify and gather only a specific set of counters out of all the counters that are available. It shows a performance monitoring application, such as the System Monitor console or the background Counter Logs and Alerts service, specifying a list of counters to be gathered and passing that list to the Performance Data Helper library (Pdh.dll) interface. It shows the PDH library sitting between the calling application and the set of Performance Library (Perflib) DLLs that are installed on the machine.

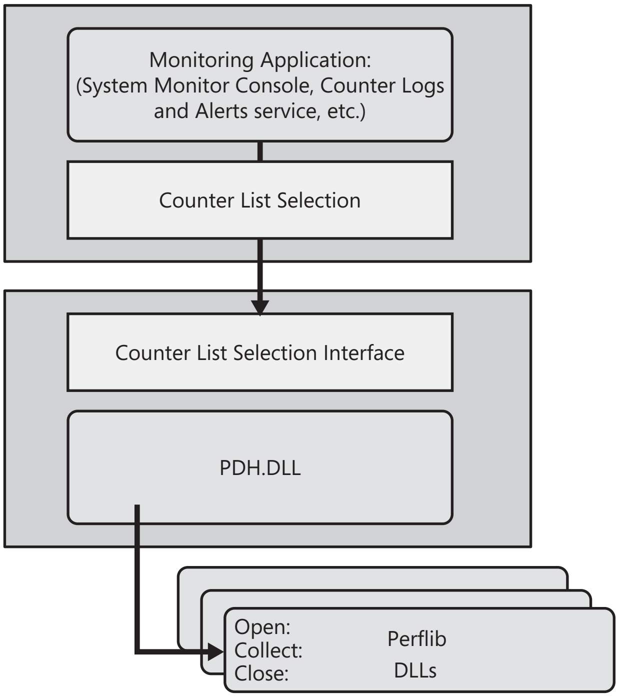

Monitoring Application:
(System Monitor Console, Counter Logs and Alerts service, etc.)
Counter List Selection
Counter List Selection Interface
PDH.DLL
Open:
Collect:
Close:
Perflib DLLs

Figure 4-35 The facilities used to identify and gather a specific set of counters

Each Perflib DLL is defined using a performance subkey in the registry at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services. This performance subkey defines the library name so that Pdh.dll can locate and load it as well as identifying three external functions that Pdh.dll can call. These are the *Open*, *Collect*, and *Close* routines. The *Open* routine is called to inventory the objects and counters that the Perflib DLL supplies. The *Collect* routine is called to gather the measurement data.

Ultimately, no matter which application programming interface is used for the counter log data, a single Performance Library DLL is responsible for gathering the data for individual counter values. These Performance Library DLLs reply to *Collect* function calls to gather the counters they maintain. Only one *Collect* function call is defined for each Performance Library DLL, and it is designed to return *all* the current counter values that the Perflib DLL maintains. Even if the calling program using the PDH interface to the counter log data requests a single counter value, the Performance Library DLL returns current data from all the counters that it is responsible for. An editing function provided by the PDH interface is responsible for filtering out all the counter values that were not requested by the original caller and returning only the values requested.

When you gather counter log data from a remote machine, the Remote Registry service that provides network access to a Performance Library DLL that is executing remotely must transfer all the data gathered from the call to the Perflib DLL Collect routine across the network, back to the PDH function. It is the responsibility of the PDH counter selection editing function to then discard the counter data not explicitly requested by the original caller. As you might expect, this architecture has major implications on performance when you need to gather counter log data from a remote machine. The biggest impact occurs when counters are selected that are part of a very voluminous set of counter data that one of the Perflibs is responsible for gathering. The most common example where extreme care should be exercised involves any counter from either the *Process* or *Thread* objects.

Process and *Thread* counter data is supplied by the Perfproc.dll Perflib, which is associated with the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\PerfProc\Performance Registry Key. The *Collect* routine of the Perfproc.dll Perflib returns all counters for all instances of the *Process* and *Thread* objects. When you request a single Process or Thread counter from a remote machine, it is necessary for all the data—returned by the Perfproc.dll Perflib *Collect* routine—to be transferred across the network back to the local machine where the performance monitoring application is running. This architecture for remote machine monitoring is discussed in more detail in the section entitled "Monitoring Remote Servers in Real Time."

Local Logging—Local Log File

In local logging with a local log file, the performance data from the local computer is logged, the log service of the local computer is configured and started locally, and the log file is stored on the local computer. This combination is the most efficient way to manage counter log data collection and has no impact on the network. Use this scenario to collect performance data for baseline use, or to track down a suspected performance problem on the local computer.

The model daily performance monitoring procedures described in this chapter all rely on logging local data to a local disk. They assume you will want to consolidate on a centrally located machine Counter Logs gathered across multiple machines on your network for the purpose of reporting, building, and maintaining a capacity planning repository of Counter Log data. This will necessitate implementing daily performance procedures, such as the ones described here, to transfer Counter Log files from each local machine to a consolidation server on a regular basis. To minimize the network impact of these file transfers, the model procedures recommended here perform this data transfer on summarized files, which are much smaller than the original counter logs. It is also possible that you can arrange to have these file transfers occur during periods of slack network activity.

Local Logging—Remote Log File

In local logging with a remote log file, you configure the local log service to collect performance data from the local computer, but the log file is written to another computer. This combination is useful on a computer with limited hard disk space. It is also useful when you need to ensure that the performance of the local hard disk is not impacted by your performance monitoring procedure. The network impact of using local logging to a remote log file is minimal because the only counter data sent across the network are those specific counter values that were selected to be logged. Although this network workload is typically small in proportion to the rest of the network traffic that occurs, it does start to add up when the data from many computers that are being monitored is written to the remote disk.

Remote Logging—Local Log File

If you are responsible for monitoring several computers in a small enterprise, it might be simplest to implement daily performance monitoring procedures in which the log service runs on one machine, gathering data from the local machine and from several remote computers (the other servers in your enterprise, for example). To limit the network impact of performing remote logging, you can log counter data to a consolidated log file on the local computer. In a small environment, this saves the trouble of having to consolidate multiple log files for analysis and reporting later. It also simplifies file management because it is not necessary to have an automated procedure, like the one documented in "Automated Counter Log Processing," to perform file management on each monitored system.

As long as you are careful about which counters you gather from the remote machines and the rate at which you gather them, this scheme is easy to implement and relatively efficient. When there is a performance problem and you need to initiate a Counter Log session promptly to identify the problem in real time, and you encounter difficulty gaining physical access to the machine experiencing the problem, you are advised to try this logging scenario.

The most frequent performance concern that arises in this scenario occurs when you need to gather data at the process or thread level from the remote machine. Even if you select only a single Process or Thread counter to monitor, the call to the *Collect* routine of the Perflib DLL loaded on the remote machine will cause all the current counter data associated with every counter and instance to be transferred across the network back to the local machine each sample interval.

Remote logging might also have additional security considerations. These are discussed later in the section entitled "Monitoring Remote Servers in Real Time."

Remote Logging—Remote Log File

Gathering counter data from one or more remote machines and writing the Counter Log to a remote disk causes the biggest network impact of any logging scenario. It might be necessary to engage in remote logging using a remote log file if the conditions are favorable for remote logging and there is not ample disk space on the local computer. If you are very careful about which counters you gather from the remote machines, the rate at which you gather them, and the amount of data that must be sent across the network to be written to the remote disk, you can implement this scheme without paying a prohibitively large performance penalty. If the computer being monitored is connected to the central monitoring server via a high-speed network, the best configuration might be to copy the detailed file to the central monitoring server and have it processed there. The optimum configuration cannot be prescribed in this text because there are too many site-specific factors to weigh.

Monitoring Remote Servers in Real Time

In any enterprise, large or small, you can't always have physical access to the computer you want to monitor. There are several ways to remotely obtain performance data from a system or group of systems. Reviewing performance parameters of several computers at the same time can also be a very effective way to study how multiple systems interact.

Monitoring performance counters remotely requires that you have network access to the remote computer and an agent on the remote computer that collects performance data and returns it to the local computer that requested it. The remote collection agent supplied with the Windows Server 2003 family is the Remote Registry service (Regsvc.dll). Regsvc.dll collects performance data about the computer it is running on and provides a remote procedure call (RPC) interface, which allows other computers to connect to the remote computer and collect that data. This service must be started and running on the remote computer for other computers to connect to it and collect performance data. Figure 4-36 illustrates the different interfaces and functional elements used when monitoring performance data remotely.

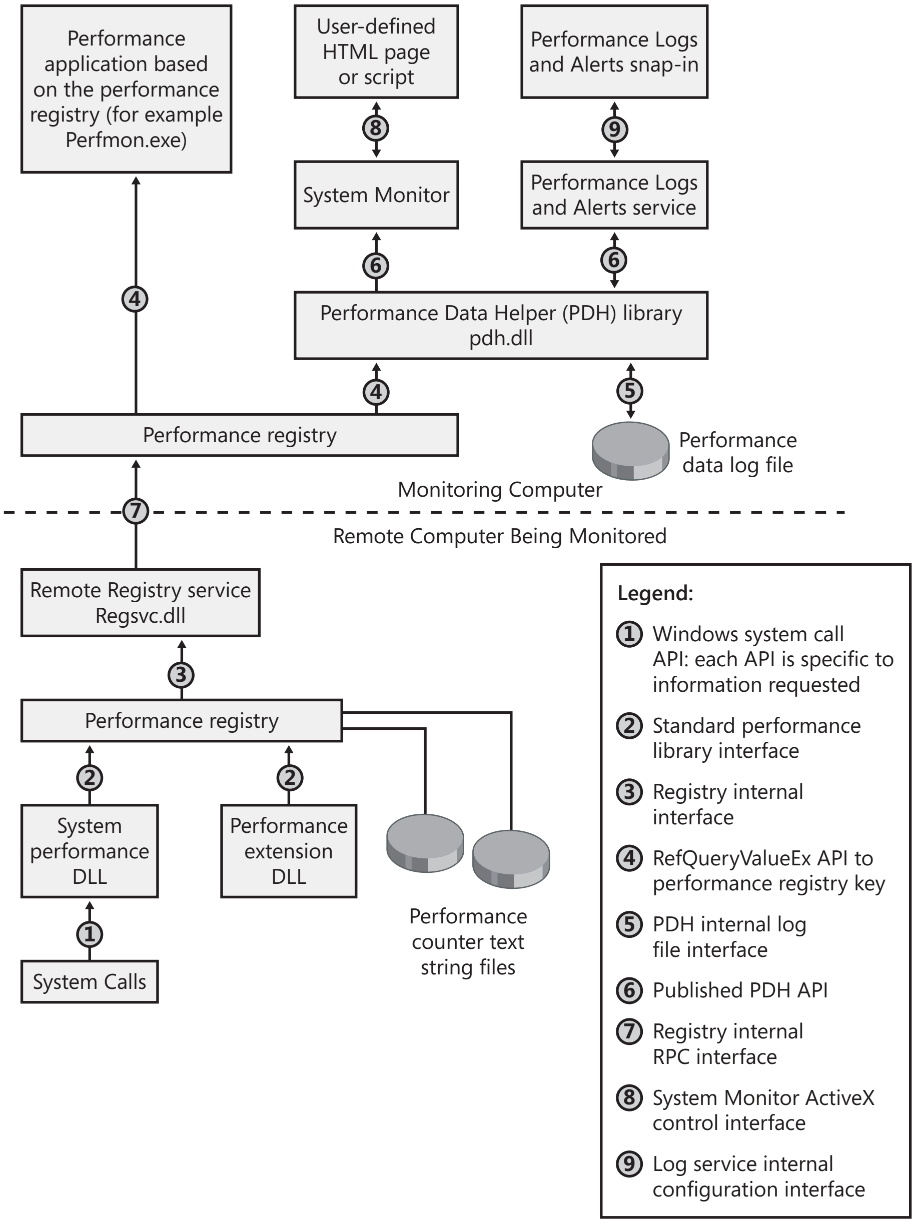

Performance application based on the performance registry (for example Perfmon.exe)
User-defined HTML page or script
Performance Logs and Alerts snap-in
8
9
System Monitor
Performance Logs and Alerts service
6
6
4
Performance Data Helper (PDH) library pdh.dll
4
5
Performance registry
Performance data log file
Monitoring Computer
7
Remote Computer Being Monitored
Remote Registry service Regsvc.dll
3
Performance registry
2
2
System performance DLL
Performance extension DLL
Performance counter text string files
1
System Calls
Legend:
1 Windows system call API: each API is specific to information requested
2 Standard performance library interface
3 Registry internal interface
4 RefQueryValueEx API to performance registry key
5 PDH internal log file interface
6 Published PDH API
7 Registry internal RPC interface
8 System Monitor ActiveX control interface
9 Log service internal configuration interface

Figure 4-36 Remote performance monitoring architecture

Access Rights and Permissions

A discretionary access control list (DACL) controls access to the performance data on the remote computer. The user or account (in the case of a service) must have permission to log on to the remote computer and read data from the registry.

Performance counter text string files To save space in the registry, the large *REG_MULTI_SZ* string variables that make up the names and explanatory text of the performance counters are saved in performance counter text string files outside the registry. These files are mapped into the registry so that they appear as normal registry keys to users and applications. Although this all takes place transparently to the calling user or application, the user or application must still have access to these files to access the performance data on the system. The performance counter text string file names are:

- %windir%\system32\perfc009.dat
- %windir%\system32\perfh009.dat

If these files are installed on an NTFS file system partition, the DACLs on both files must grant at least read access to the intended users of the performance data. By default, only the Administrators group and interactive users are given sufficient access to these files; therefore, the values of entries in the \HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\009 subkey are invisible to all other users. Note that both files must have the correct ACL; if they do not, neither will be visible.

Using the Performance Console for Remote Performance Analysis

The Performance console that is used to monitor a local computer can also be used to monitor remote computers at the same time. The ability to monitor local and remote computers by using the same tool makes it possible for you to easily observe the interaction of the different components in a client/server application.

Selecting systems to monitor To interactively monitor the performance of remote systems, select or enter the name of the system you want to monitor in the Select Counters From Computer box in the System Monitor Add Counters dialog box. There is normally a delay between the time you specify the remote computer name and the time that its performance counters and objects appear in the dialog box. During the delay, the local computer is establishing a connection with the remote computer's performance registry and retrieving the list of performance counters and objects that are available on the remote computer.

Saving console settings for remote performance analysis When you save the settings from a Performance console that is monitoring remote computers, be sure the counter paths are saved the way you want to apply them later. Determine exactly which counters you will want the user of that settings file to monitor, then use the correct setting for those counters. This determination is made in the Add Counters dialog box of the Performance console. In the Add Counters dialog box, you can choose from two options that determine from where the performance counters are read:

- **Use Local Computer Counters** If this option is selected, only counters from the local computer are displayed in the Add Counters dialog box. When the Performance console settings are saved and sent to another computer, the same list of counters from the computer loading the settings file will be displayed in System Monitor.
- **Select Counters From Computer** If this option is selected, the name of the computer you specify is saved along with the counters from that computer. No matter where that Performance console settings file is sent, the counter data from the original computer specified in each counter path will always be used.

Sending console settings to others for remote performance analysis How you specify the computer you want to monitor in the Add Counters dialog box determines how the settings will be applied on another computer. For example, if you click the Select Counters From Computer option, and the computer specified in the combo box is the current computer (\\MyMachine, for example), when Performance console settings from \\MyMachine are sent to another computer (\\YourMachine, for example) and opened in the Performance console on \\YourMachine, that Performance console will try to connect to \\MyMachine to collect performance data. This might not be what you want, so make this selection carefully.

Troubleshooting Counter Collection Problems

As you come to rely on counter logs for detecting and diagnosing common performance problems, you will develop less and less tolerance for any problems that interfere with your ability to gather counter logs reliably. Most of the counter collection problems you are likely to encounter are associated with unreliable extended counters supplied by third-party Performance Library DLLs. In this section, a variety of troubleshooting procedures to cope with common counter collection problems are discussed.

Missing Performance Counters

When Performance Monitoring API calls that are routed via Pdh.dll to Performance Library DLLs fail, counter values that you expected to gather are missing. (See the discussion earlier in this chapter about the internal architecture of the performance monitoring API in the section entitled "Counter Log Scenarios.") When these failures occur, the Performance Data Helper library routines attempt to document the error condition and isolate the failing component so that the component does not cause system-wide failure of the performance monitoring data gathering functions. This section reviews some of the common error messages that can occur and what should be done about them. It also discusses the Disable Performance Counter function, which is performed automatically to isolate the failing component and prevent it from causing a system-wide failure. If you are unable to gather some performance counters that should be available on a machine, it is usually because the Performance Library associated with those counters has been disabled automatically because of past errors.

Common Perflib Error Messages

When Performance Monitoring API calls that are routed via Pdh.dll to Performance Library DLLs fail, the PDH routines attempt to recover from those failures and issue a diagnostic error message. These error messages are written to the Application event log with a Source identified as "Perflib." These error messages are a valuable source of information about counter collection problems.

The performance monitoring API defines three external function calls that each Performance Library DLL must support. These are *Open*, *Collect*, and *Close*. Errors are most common in the *Open* routine, where the Performance Library DLL responds with a set of counter definitions that it supports, and in the *Collect* routine, where the Performance Library DLL supplies current counter values. (See Table 4-8.)

Table 4-8 Error Messages in the *Open* and *Collect* Routines

Event ID	Message/Explanation
1008	The Open Procedure for service (service name) in DLL (DLL name) failed. Performance data for this service will not be available. Status code returned is DWORD 0.
	Self-explanatory. The *DisablePerformanceCounters* flag in the associated registry key at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<service-name>\Performance is set to 1 to prevent the problem from recurring.

Table 4-8 Error Messages in the *Open* and *Collect* Routines

Event ID	Message/Explanation
1009	The Open Procedure for service (service name) in DLL (DLL name) generated an exception. Performance data for this service will not be available. Exception code returned is DWORD 0.
	Self-explanatory. The *DisablePerformanceCounters* flag in the associated registry key at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<service-name>\Performance is set to 1 to prevent the problem from recurring.
1010	The Collect Procedure for the (service name) service in DLL (DLL name) generated an exception or returned an invalid status. Performance data returned by counter DLL will not be returned in Perf Data Block. Exception or status code returned is DWORD 0.
	Self-explanatory. The *DisablePerformanceCounters* flag in the associated registry key at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<service-name>\Performance is set to 1 to prevent the problem from recurring.
1011	The library file (DLL name) specified for the (service name) service could not be opened. Performance data for this service will not be available. Status code is data DWORD 0.
	Perflib failed to load the performance extensions library. The status code from *GetLastError* is posted in the data field of the event. For example, 7e means the DLL could not be found or the library name in the registry is not correct.
1015	The timeout waiting for the performance data collection function (function name) to finish has expired. There may be a problem with that extensible counter or the service from which it is collecting data.
	The *Open* or *Collect* routine failed to return in the time specified by the *Open Timeout*, *Collect Timeout,* or *OpenProcedureWaitTime* registry fields. If no registry values are set for the Perflib, the default timeout value is 10 seconds.

When serious Perflib errors occur that generate Event ID 1008, 1009, and 1010 messages, steps are taken to isolate the failing component and safeguard the integrity of the remainder of the performance monitoring facility. Performance data collection for the associated Perflib is disabled until you are able to fix the problem. Fixing the problem might require a new or an upgraded version of the Perflib. After the problem is resolved, you can re-enable the Performance Library and begin gathering performance statistics from it again.

Additional error conditions in which the Perflib DLL returns invalid length performance data buffers also cause the performance monitoring API to disable the Perflib generating the error.

More Info For more information about these other Perflib error messages, see KB article 226494, available at http://support.microsoft.com/default.aspx?scid=kb;zh-tw;226494.

A *Collect Timeout* value can be specified (in milliseconds) in the HKLM\System\CurrentControlSet\Services\<Service-name>\Performance key for the Performance Library DLL. If this value is present, the performance monitoring API sets up a timer value prior to calling the Perflib's *Collect* routine. If the *Collect* function of the Perflib does not return within the time specified in this registry value, the call to *Collect* fails and an Event ID 1015 error message is posted to the event log.

Similarly, there is an *Open Timeout* value in the registry under the Performance subkey that functions in the same fashion for calls to the Perflib's *Open* routine.

A registry field called *OpenProcedureWaitTime* at HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib establishes a global default timeout value for all Performance Library DLLs if *Open Timeout* or *Collect Time* are not set explicitly. The *OpenProcedureWaitTime* registry value defaults to a timeout value of 10,000 milliseconds, or 10 seconds.

Disable Performance Counters

To maintain the integrity of the performance data and to improve reliability, the performance monitoring API disables any performance DLL that returns data in the incorrect format, causes an unhandled program fault, or takes too long to return the performance data. As a result of an error condition of this magnitude, a field is added to the registry in the HKLM\System\CurrentControlSet\Services\<Service-name>\Performance key named Disable Performance Counters. When Disable Performance Counters is set to 1, no performance monitoring application will be able to gather the counters that the disabled Perflib is responsible for gathering. Perflib Event ID 1017 and 1018 messages are written to the Application event log at the time the *Disable Performance Counters* flag is set.

Note When a Performance Library DLL is disabled, the performance counters gathered by that DLL are not available through the System Monitor console, the background Counter Logs and Alerts service, or any other performance application that calls Performance Data Helper API functions. Disabled Perflib DLLs remain disabled until the *Disable Performance Counters* flag in the registry is reset manually. Disabled DLLs are not reloaded when the system is restarted.

Sometimes, the problem that leads to the Performance Library is transient and will clear up on its own. You can try re-enabling the extension DLL using the ExCtrLst utility (part of the Windows Server 2003 Support Tools) and restarting the counter log session. Alternatively, you can use the registry Editor to change the value of the *Disable Performance Counters* flag to zero.

Sometimes, by the time you notice that some performance counters are disabled, the event log messages that informed you of the original Perflib failure are no longer available. The only way to reconstruct what might have happened is to reset the *Disable Performance Counters* flag and try to perform the counter logging again.

If the problem persists and the *Disable Performance Counters* flag is set again, contact the vendor that developed the Performance Library for a solution. If the object is a Windows Server 2003 system object (such as the *Process* object), please contact Microsoft Product Support Services (PSS). See the procedures discussed in "Troubleshooting Counter Collection Problems" for more information.

Troubleshooting Counter Collection Problems

If you are having problems gathering performance counters provided by a Microsoft or a third party–supplied Performance Library DLL, you might be expected to gather the following information to assist your vendor in determining the problem.

1. Obtain a copy of the HKLM\SYSTEM\CurrentControlSet\Services registry hive.
2. Obtain a copy of the Perfc009.dat and Perfh009.dat files in %SystemRoot%\system32.
3. Copy the HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib\009 key. To do this, double-click both the Counter and Help fields and copy their contents to a .txt file because they cannot be exported to a .reg file.
4. Obtain a copy of this registry hive: HKEY_CURRENT_USER\Software\Microsoft\PerfMon.
5. Start a counter log session that attempts to gather every counter registered on your system for a brief interval. The counter log can be compared to the Performance subkeys in the registry at HKLM\SYSTEM\CurrentControlSet\Services to see what objects, counters, and instances are missing.

Restoring Corrupt Performance Counters

Problems associated with a bad Performance Library DLL might become so serious you find it necessary to reconstruct the base set of performance counter libraries that come with Windows Server 2003, along with any extensible counters associated with other Perflib DLLs. The procedure for rebuilding the performance counters is to issue the following command:

```
lodctr /r
```

This command restores the performance counters to their original baseline level, plus applies any extended counter definitions that have been added later through subsequent software installations.

Chapter 5

Performance Troubleshooting

In this chapter:

Bottleneck Analysis 402
Analysis Procedures. 404
Processor Troubleshooting. 406
Memory Troubleshooting. 430
Disk Troubleshooting 464
Network Troubleshooting 509

A framework for proactive performance monitoring of your Microsoft Windows Server 2003 infrastructure was provided in Chapter 4, "Performance Monitoring Procedures." Sample procedures that illustrated ways to implement systematic performance monitoring were discussed, as were the measurements and the procedures for gathering those measurements. These measurements are used to establish a *baseline*, documenting historical levels of service that your Windows Server 2003 machines are providing to the client customers of your server applications. By extrapolating from historical trends in resource usage—that is, processor usage, disk space consumption, network bandwidth usage, and so on—you can use the data collection, summarization, and archival procedures described in Chapter 4 to anticipate and detect periods during which resource shortages might cause severe availability and performance problems, and also take action to keep these issues from disrupting the orderly delivery of services to server application clients.

However, despite your best efforts at proactive performance monitoring, some problems will inevitably occur. The range and complexity of at least some of your Windows Server 2003 machines may mean that the relatively simple performance data gathering procedures discussed in Chapter 4 are not going to catch everything occurring on your machines. There is simply too much performance data on too many operating system and application functions to gather all the time.

In this chapter, procedures for troubleshooting performance problems in specific areas are discussed and illustrated. The focus is primarily on troubleshooting performance problems and identifying the precise cause of a shortage in any of the four classes of machine resources common to all application environments: processor, physical memory and paging, disk, and network. This analysis cannot be performed in a vacuum. The load on these resources is generated by specific application requests. The troubleshooting procedures you use might have to be tailored for each application-specific environment that you are responsible for managing.

The troubleshooting procedures documented here all begin with the baseline of performance data you have been accumulating (using procedures similar to the ones described in Chapter 4, "Performance Monitoring Procedures") to identify resource *bottlenecks*. Resources that are bottlenecks are not sized large enough to process all the requests for service they receive, thus causing serious delays. Bottleneck analysis is the technique used to identify computer resource bottlenecks and systematically eliminate them. Identifying the root cause of a resource shortage often involves the use of very fine-grained troubleshooting tools. This chapter will illustrate how to use performance baseline measurements to identify resource shortages and how to use troubleshooting tools to hone in on specific problems.

Bottleneck Analysis

Analyzing a performance problem is easier if you already have established data collection and analysis procedures, and have collected performance data at a time when the system or enterprise is performing optimally.

Baseline Data

The procedures for capturing, reporting, and archiving baseline performance data are described in Chapter 4, "Performance Monitoring Procedures." Collecting and analyzing baseline data before you have to conduct any problem analysis or troubleshooting makes the troubleshooting process much easier. Not only will you have a set of reference values with which you can compare the current values, but you will also gain some familiarity with the tools.

The baseline data to use during a troubleshooting or analysis session can exist in several forms:

- Tables of data collected from baseline analysis can be used in hardcopy form as a reference for comparison to current values. Graphs can also be used for baseline data that varies over time.
- Performance data logs saved during baseline analysis can be loaded into one System Monitor window and compared with current performance data displayed in another System Monitor window.

Tip You might find it useful to save baseline workload data from a System Monitor Report view in .tsv format. Then you can open open several .tsv files that reflect different points in time in Microsoft Excel and compare them side by side in a spreadsheet.

Although having this baseline data on hand is the ideal case, it is not always possible. There might be times during the installation or setup of a system or an application when performance is not as good as it needs to be. In those cases, you will need to troubleshoot the system without the benefit of a baseline to help you isolate or identify the problem. Both approaches to troubleshooting—with baseline data and without—are covered in this chapter.

Current Performance Levels

The first step in a performance analysis is to ensure performance data is being logged. Even though System Monitor can monitor and display performance data interactively, many problems show up more clearly in a performance data log.

When collecting performance data for problem analysis, you typically collect data at a higher rate (that is, using a shorter sample interval) and over a shorter period of time than you would for routine data logging of the type recommended in Chapter 4, "Performance Monitoring Procedures." The goal is to generate a performance data log file with sufficient resolution to perform a detailed investigation of the problem. Otherwise, important information from instantaneous counters can be lost or obscured in the normal performance logs that use a longer sample interval. Creating a separate performance log query for each task is one way to accomplish this. Of the possible performance data log formats, the binary format gives the best results. The binary log retains the most information and allows for the most accurate relogging to other formats. Whenever raw data is processed into formatted data (which occurs with all other data formats), some information is lost; therefore, subsequent computations on formatted data are less accurate than computations performed on the raw, original logged data that is stored in the binary log file format. If you have many servers to monitor, creating a SQL database for your performance data might be useful as well, making consolidation and summarization of the data easier.

Resource Utilization and Queue Length

To identify saturated resources that might be causing performance bottlenecks, it is important to focus on the following activities:

- Gathering measurement data on resource utilization at the component level
- Gathering measurement data on queuing delays that are occurring at the resource that might be overloaded
- Determining the *relationship* between resource utilization and request queuing that exists at a particular resource

Theoretically, a nonlinear relationship exists between utilization and queuing, which becomes evident when a resource approaches saturation. When you detect one of these characteristically nonlinear relationships between utilization and queuing at a resource, there is a good chance that this overloaded resource is causing a performance constraint.

Decomposition

Once you identify a bottlenecked resource, you often break down utilization at the bottlenecked resource according to its application source, allowing you to hone in on the problem in more detail. Several examples that illustrate bottleneck detection and decomposition are provided later in this chapter.

Analysis Procedures

Listing all possible problems and their subsequent resolutions in this chapter is impossible. It is possible, however, to describe common problems, the steps to take when analyzing those problems, and the data you should gather so that you can perform your own analysis of similar problems.

Understanding the Problem

The obvious but often overlooked first step in problem analysis is to understand the problem being observed or reported. The initial indication of a potential problem can come from a user, a customer, or even a superior. Each person might characterize the same problem differently or use the same characterization for different problems. As the person doing the analysis, you need to clearly understand what the person reporting the problem is describing.

Analyzing the Logged Performance Data

The performance counters you log to provide data for troubleshooting are listed later in this chapter. These lists are different from the lists of counters that you evaluate and analyze daily, because logging the data to a binary log file is best done by performance object, whereas the actual analysis is performed on the individual performance counters. Also, having more available data (to a point) is better than having less during analysis. Then, if you find something interesting when examining one of the key counters, you will have additional data at your disposal for further investigation. On the other hand, if your system is resource-constrained and you are trying to generate the smallest possible log file, you can collect only the key counters to analyze, with the understanding that you might not have all the data you need to do a comprehensive analysis. Of course, some analysis is usually better than no analysis.

Analyzing Performance Data Interactively

For the performance problems that result from a lengthy and subtle change in system behavior, the best way to start your analysis is by using a log file; however, some problems are immediate and must be analyzed interactively. In such cases, the counters you need to monitor interactively are the same ones listed later in this chapter. During an interactive analysis session, the counters listed are the first ones that should be loaded into the interactive tool.

Fine-Grained Analysis Tools

In many problem-solving scenarios, gathering counter log data is often only the first step in your analysis. You will often need to rely on more fine-grained performance analysis tools, some of which are described in the "Help and Support" section of the Windows Server 2003 documentation. In this chapter, how to use the following fine-grained analysis tools is discussed:

- Kernrate for the analysis of processor bottlenecks
- Trace logs, Kernel Debugger commands, and the Poolmon utility for the analysis of physical and virtual memory shortages
- Trace logs for the analysis of disk bottlenecks
- Server Performance Advisor reports for troubleshooting networking problems
- Network Monitor packet traces and trace logs for the analysis of network bottlenecks

What to Check Next in the Enterprise

In most cases, the next step in evaluating a problem at the enterprise level is to go to the system exhibiting the problem and investigate further. In the case of a network problem, look at the system components and the role they play in the enterprise before examining the system itself. For example, heavier-than-normal network traffic on one client might be the result of an errant application on the client, or it might be a problem on a server that is not responding correctly to requests from that client. For more information, see "Network Troubleshooting" later in this chapter.

Tip A quick way to localize the source of an overloaded server is to temporarily remove it from the network (if you can do that safely).

Monitor the Processor(_Total)\% Processor Time counter by using the Performance console on the server in question. If it is not possible to observe the System Monitor directly, make sure this performance counter is logged to a performance data log file while performing this test.

Briefly, remove the server from the network for a period at least as long as several sample intervals. If you are monitoring interactively, observe the value of the Processor(_Total)\% Processor Time performance counter. If the value drops when the server is disconnected from the network, the load on the server is the result of client-induced requests or operations. If the value remains more or less the same, the load is the result of a process or application that resides on that server.

Be sure to reconnect the server to the network when this test is complete.

Processor Troubleshooting

Processor bottlenecks occur when the processor is so busy that it cannot respond to requests for a noticeable period of time. Extended periods of near 100 percent processor utilization, accompanied by an increase in the number of ready threads delayed in the processor dispatcher queue, are the primary indicators of a processor bottleneck. These indicators reflect direct measurement of resource utilization and the queue of requests delayed at a busy resource.

Resource Utilization and Queue Length

The major indicators of a processor bottleneck are the system-level measurements of processor utilization and processor queuing, as shown in Table 5-1.

Table 5-1 Primary Indicators of a Processor Bottleneck

Counter	Primary Indicator	Threshold Values
Processor(_Total)\ % Processor Time	Processor utilization	Sustained values > 90% busy on a uniprocessor or sustained values > 80% on a multiprocessor should be investigated.
System\Processor Queue Length	Current depth of the thread Scheduler Ready Queue	Numerous observations > 2 Ready threads per processor should be investigated. Observations > 5 Ready threads per processor are cause for alarm.*

* This is an instantaneous counter and, therefore, often can report data that is not representative of overall system behavior.

In addition to the recommended threshold levels in Table 5-1, take note of any measurements that differ sharply from the historical baseline.

The combination of high processor utilization and a lengthy processor queue signals an overloaded processor. Often the situation is unambiguous, for example, a program thread stuck in an infinite loop of instructions that drives processor utilization to 100 percent. This condition is normally a programming bug that can be alleviated only by identifying the offending program thread and ending its parent process. A sudden spike in demand or even a gradual, sustained increase in the customer workload might also create an evident out-of-capacity situation. The proactive performance monitoring procedures discussed in Chapter 4, "Performance Monitoring Procedures," of course, are designed to allow you to detect and anticipate an out-of-capacity situation and intervene before it impacts daily operations.

Note This section ignores complex measurement and interpretation issues introduced by hyperthreaded processors, multiprocessors, and Non-Uniform Memory Access (NUMA) architectures. These are all subjects discussed in Chapter 6, "Advanced Performance Topics."

On the other hand, in many situations, you need to exercise considerable judgment. The interpretation of the two performance counters described in Table 5-1, which are the primary indicators of a resource shortage, is subject to the following considerations:

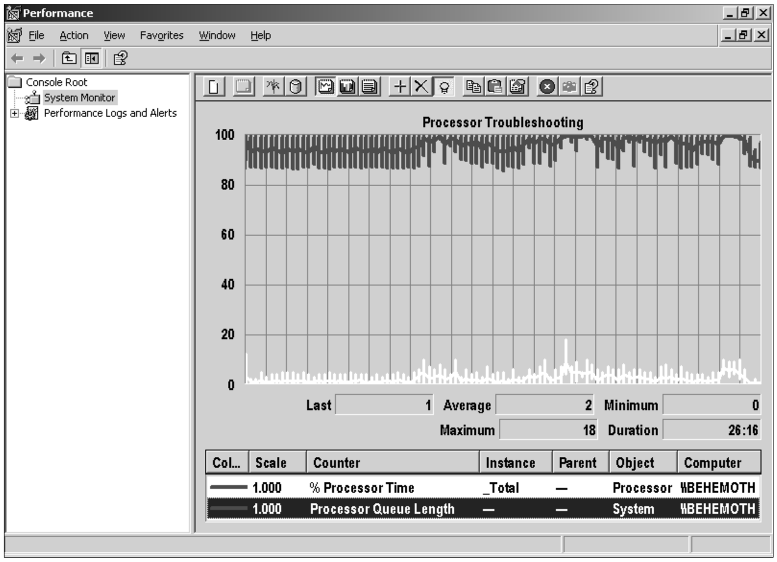
- Processor state is sampled several hundred times per second and the results are accumulated over the measurement interval. Over small intervals, the sampling data can be skewed.
- Beware of program threads that are *soakers*, that is, capable of absorbing any excess processor cycles that are available. Screen savers, Web page animation controls, and other similar applications need to be excluded from your analysis.
- Processor Queue Length is an instantaneous counter reflecting the depth of the Ready Queue at the last processor state sample.
- Polling threads of various kinds that are activated by a timer interrupt that can pop in tandem with the measurement timer can create a false impression about the size of the Ready Queue
- Individual Processor Queue Length counter samples can occur at intervals that are at odds with the processor-busy measurements accumulated continuously during the interval.
- Most threads are in a voluntary Wait state much of the time. The remaining threads that are actively attempting to run—ordinarily a small subset of the total number of threads that exist—form the practical upper limit on the number of threads that you can observe in the processor queue.

Any normal workload that drives processor utilization to near 100 percent for an extended period of time is subject to a processor capacity constraint that warrants some form of relief. This is self-evident even if the processor queue of Ready threads remains low. Such a *CPU-bound* workload will undoubtedly benefit from a faster processor. If the workload is multithreaded and can proceed in parallel, a multiprocessor upgrade should also relieve the situation.

If an afflicted machine is truly short of processor capacity, a range of possible solutions to increase processor capacity are available, including these:

- Moving the workload to a faster machine
- Adding processors to the current machine
- Directing the workload to a cluster of machines

Figure 5-1 illustrates a processor out-of-capacity condition in which processor utilization remains at or near 100 percent for an extended period of time. The observed maximum Processor Queue Length during this measurement interval is 18 threads delayed in the Ready Queue. The Chart view also reveals several observations in which the Processor Queue Length exceeds 10 on this machine, which happens to be a uniprocessor.


Performance
File Action View Favorites Window Help
Console Root
System Monitor
Performance Logs and Alerts
Processor Troubleshooting
100
80
60
40
20
0

Last 1 Average 2 Minimum 0
Maximum 18 Duration 26:16

Col...	Scale	Counter	Instance	Parent	Object	Computer
	1.000	% Processor Time	_Total	—	Processor	\\BEHEMOTH
	1.000	Processor Queue Length	—	—	System	\\BEHEMOTH

Figure 5-1 Out-of-capacity condition

Note that the scale for the System\Processor Queue Length scale in the Chart view has been changed from its default value, which multiples the observed values by 10, to a scaling value of 1.

Decomposition

After you identify a processor out-of-capacity condition, you need to investigate further. Initially, you have three lines of inquiry:

- **Determine processor utilization by processor state** The processor state measurements allow you to determine whether the component responsible for the processor load is a User mode or Privileged mode application, which includes device interrupt processing routines.
- **Determine processor utilization by processor** This is usually necessary only when the machine is deliberately configured for asymmetric multiprocessing. Asymmetric multiprocessing is an important subject that is addressed at length in Chapter 6, "Advanced Performance Topics."

- **Determine processor utilization by process and thread** When the processor overload originates in a User-mode application, you should be able to identify the process or processes responsible.

In each of these cases, it might also be necessary to determine processor utilization by application module and function.

Processor Use by State

When the processor executes instructions, it is in one of two *states*: Privileged mode or User mode.

Privileged mode Authorized operating system threads and interrupts, including all device driver functions, execute in Privileged mode. Kernel-mode threads also execute in Privileged mode.

Processor(*n*)\% Privileged Time records the percentage the system was found busy while executing in Privileged mode. Within % Privileged Time, you can also distinguish two additional modes: Interrupt mode and deferred procedure call (DPC) mode, which typically account for only a small portion of the time spent in Privileged mode.

Interrupt mode is a high-priority mode reserved for interrupt service routines (ISRs), which are device driver functions that perform hardware-specific tasks to service an interrupting device. Interrupt processing is performed at an *elevated* dispatching level, with interrupts at the same or lower priority disabled.

Note Although interrupt priority is a hardware function, the Windows Server 2003 Hardware Abstraction Level (HAL) maintains an interrupt priority scheme known as interrupt request levels (IRQLs), which represent the current dispatching mode of a processor. Processing at a higher IRQL will preempt a thread or interrupt running at a lower IRQL. An IRQL of 0 means the processor is running a normal User- or Kernel-mode thread. There is also an IRQL of 1 for asynchronous procedure calls, or APCs. An IRQL value of 2 indicates a deferred procedure call (DPC) or dispatch-level work, discussed in more detail later. An IRQL greater than 2 indicates a device interrupt or other high priority work is being serviced. When an interrupt occurs, the IRQL is raised to the level associated with that specific device and calls the device's interrupt service routine. After the ISR completes, the IRQL is restored to the previous state when the interrupt occurred. Once the IRQL on the processor is raised to an interrupt-level IRQL, interrupts of equal or lower IRQL are masked on that processor. Interrupts at a higher IRQL can still occur, however, and will preempt the current lower priority activity that was running.

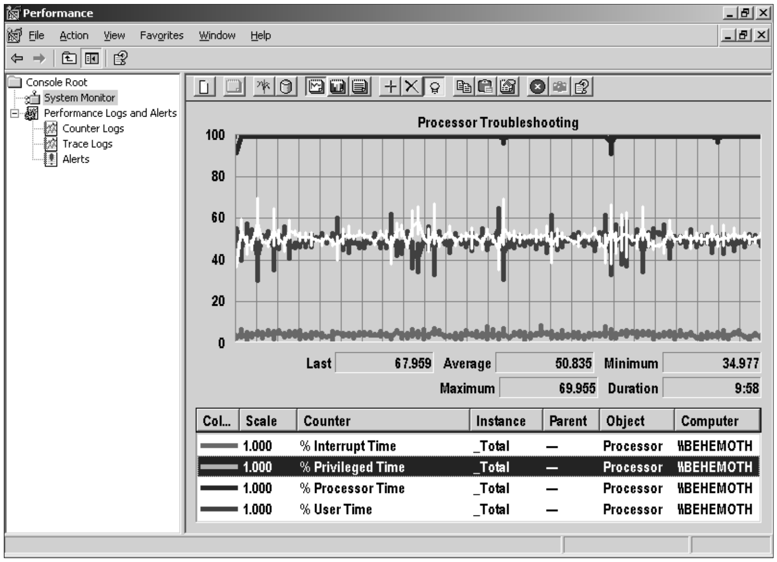
Processor(*n*)\% Interrupt Time records the percentage of time the system was found busy while it was executing in Interrupt mode, or other high IRQL activity. This includes the execution time of ISRs running at a higher priority than any other Kernel or User-mode thread. The amount of % Interrupt Time measured is included in the % Privileged Time measurement. % Interrupt Time is broken out separately to make identifying a malfunctioning interrupt service routine easier.

DPC mode is time spent in routines known as deferred procedures calls, which are device driver–scheduled routines called from ISRs to complete device interrupt processing once interrupts are re-enabled. DPCs are often referred to as *soft interrupts*. DPCs run at *dispatch level* IRQL, just below the priority level of interrupt service routines. DPCs at dispatch level IRQL will be interrupted by any interrupt requests that occur because hardware interrupts have IRQL greater than the dispatch level. In addition to DPCs, there are other sources of dispatch level activity on the system.

Processor(*n*)\% DPC Time records the percentage of time the system was found busy while at dispatch level. This represents the execution time of DPCs running at a higher priority than any other Kernel- or User-mode thread, excluding ISRs and other dispatch level activity. The amount of % DPC Time that is measured is also included in the % Privileged Time measurement. % DPC Time is broken out separately to make identifying a malfunctioning interrupt service routine's DPC easier.

User mode Program threads from services and desktop applications execute in User mode. User-mode threads cannot access system memory locations and perform operating system functions directly. Processor(*n*)\% User Time records the percentage the system was found busy while it was executing in User mode. Figure 5-2 shows the % Processor Time on the same machine broken down by Processor State.

The relative proportion of % User Time and % Privileged Time is workload-dependent. Do not expect a constant relationship between these two counters on machines running different workloads. In this example, time spent in User mode is only slightly more common than Privileged-mode execution time. On the same machine running a similar workload, the relative proportion of User-mode and Privilege-mode execution time should be relatively constant. Compare measurements from the current system to the historical baseline measurements you have put aside. Unless the workload changes dramatically, the proportion spent in User mode and Privileged mode should remain relatively constant for the same machine and workload over time. Are you able to observe a major change in the ratio of User- to Privileged-mode processing? This could be an important clue to understanding *what* has changed in the interim.


Performance
File Action View Favorites Window Help
Console Root
System Monitor
Performance Logs and Alerts
Counter Logs
Trace Logs
Alerts
Processor Troubleshooting
100
80
60
40
20
0

Last 67.959 Average 50.835 Minimum 34.977
Maximum 69.955 Duration 9:58

Col...	Scale	Counter	Instance	Parent	Object	Computer
	1.000	% Interrupt Time	_Total	—	Processor	\\BEHEMOTH
	1.000	% Privileged Time	_Total	—	Processor	\\BEHEMOTH
	1.000	% Processor Time	_Total	—	Processor	\\BEHEMOTH
	1.000	% User Time	_Total	—	Processor	\\BEHEMOTH

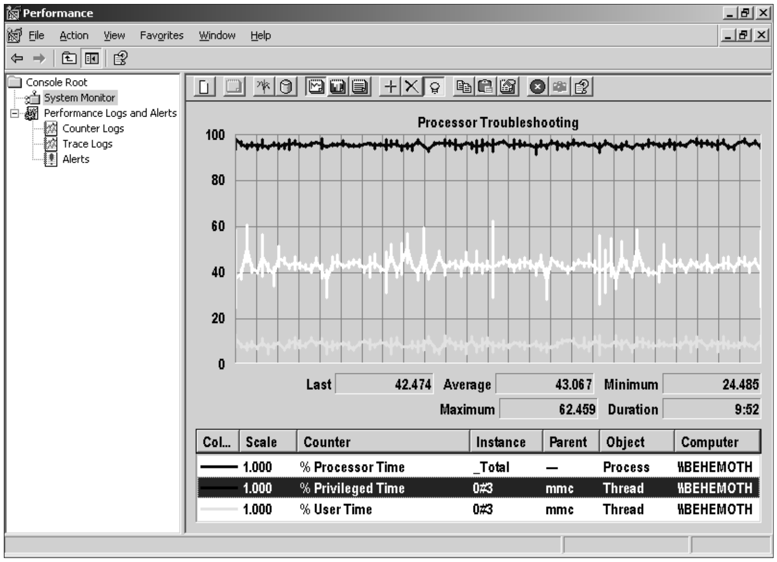
Figure 5-2 Percentage of processor time broken down by processor state

The relative proportion of time spent in Interrupt mode—generally, a function of network and disk interrupt rates—is normally quite small. In the example shown in Figure 5-2, the amount of time was insignificant, consistently less than 2 or 3% busy. If the current amount of % Interrupt Time is much higher than historical levels, you might have a device driver problem or a malfunctioning piece of hardware. Compare the current Interrupts/sec rate to historical levels. If the current Interrupts/sec rate is proportional to the level measured in the baseline, device driver code is responsible for the increase in % Interrupt Time. If the Interrupt rate is sharply higher, you have a hardware problem. In either case, you will want to use the Kernrate utility to hone in on the problem device. (Kernrate is discussed later.) If you observe a spike in the amount of % DPC Time being consumed, identical considerations apply.

If the bulk of the time is being spent in User-mode processing, proceed directly to analyzing processor usage at the process level. For Privileged-mode processing, you might or might not find a clear relationship to a specific process or processes. Nevertheless, because both User-mode and Privileged-mode processing are broken out at the process level, checking out the process level measurements, even when excessive Privileged-mode execution time is the concern, is worthwhile.

Processor Use by Process

The sample measurements that determine the state of the machine when it is busy executing instructions also track the process and thread context currently executing. Once you determine which process instance or instances you need to observe closely, select their Process(*n*)\% User Time and Process(*n*)\% Privileged Time counters. You can even drill down to the thread level, as illustrated in Figure 5-3.



Last 42.474 Average 43.067 Minimum 24.485
Maximum 62.459 Duration 9:52

Col...	Scale	Counter	Instance	Parent	Object	Computer
	1.000	% Processor Time	_Total	—	Process	\\BEHEMOTH
	1.000	% Privileged Time	0#3	mmc	Thread	\\BEHEMOTH
	1.000	% User Time	0#3	mmc	Thread	\\BEHEMOTH

Figure 5-3 A single thread impacting the percentage of processor time consumed

In this example, a single thread from an Mmc.exe parent process is responsible for about 40 percent of the total % Processor Time recorded in Figure 5-2. The complication introduced when you need to drill down to the process or thread level is that there are many individual processes to examine. It is often easier to use Task Manager first to identify a CPU-bound process, as discussed later.

Identifying a Runaway Process by Using Task Manager

If you are diagnosing a runaway process in real time, use Task Manager to help you zero in on a problem process quickly.

1. Press CTRL+SHIFT+ESC to launch Windows Task Manager, or click the Task Manager icon in the system tray if Task Manager is already active.

2. Click the Processes tab. Select View, choose Select Columns, and make sure that the CPU Usage field is displayed. This column is labeled CPU in the Processes view.

3. Click the CPU column label to sort the displayed items in sequence by processor usage. You should see a display similar to the one in Figure 5-4. To sort the display in reverse order, click the column heading again.

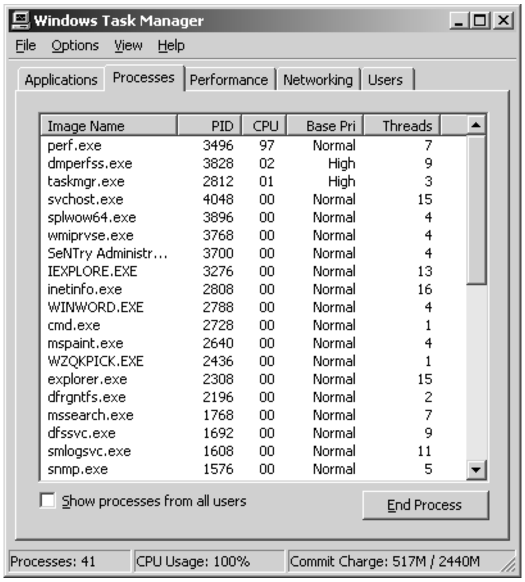

Windows Task Manager
File Options View Help
Applications Processes Performance Networking Users
Show processes from all users
End Process

Image Name	PID	CPU	Base Pri	Threads
perf.exe	3496	97	Normal	7
dmperfss.exe	3828	02	High	9
taskmgr.exe	2812	01	High	3
svchost.exe	4048	00	Normal	15
splwow64.exe	3896	00	Normal	4
wmiprvse.exe	3768	00	Normal	4
SeNTry Administr...	3700	00	Normal	4
IEXPLORE.EXE	3276	00	Normal	13
inetinfo.exe	2808	00	Normal	16
WINWORD.EXE	2788	00	Normal	4
cmd.exe	2728	00	Normal	1
mspaint.exe	2640	00	Normal	4
WZQKPICK.EXE	2436	00	Normal	1
explorer.exe	2308	00	Normal	15
dfrgntfs.exe	2196	00	Normal	2
mssearch.exe	1768	00	Normal	7
dfssvc.exe	1692	00	Normal	9
smlogsvc.exe	1608	00	Normal	11
snmp.exe	1576	00	Normal	5

Processes: 41 | CPU Usage: 100% | Commit Charge: 517M / 2440M

Figure 5-4 Processes in sequence by processor usage (CPU column)

When the display is sorted, a runaway process will appear at the top of the display and remain there for an extended period of time.

4. Select the runaway process, right-click, and then click Set Priority to reset the dispatching priority to Low or Below Normal, as illustrated in Figure 5-5.

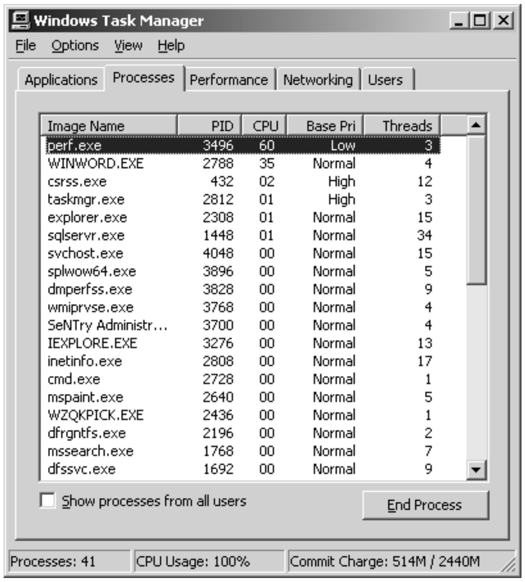

Windows Task Manager
File Options View Help
Applications Processes Performance Networking Users
Show processes from all users
End Process

Image Name	PID	CPU	Base Pri	Threads
perf.exe	3496	60	Low	3
WINWORD.EXE	2788	35	Normal	4
csrss.exe	432	02	High	12
taskmgr.exe	2812	01	High	3
explorer.exe	2308	01	Normal	15
sqlservr.exe	1448	01	Normal	34
svchost.exe	4048	00	Normal	15
splwow64.exe	3896	00	Normal	5
dmperfss.exe	3828	00	Normal	9
wmiprvse.exe	3768	00	Normal	4
SeNTry Administr...	3700	00	Normal	4
IEXPLORE.EXE	3276	00	Normal	13
inetinfo.exe	2808	00	Normal	17
cmd.exe	2728	00	Normal	1
mspaint.exe	2640	00	Normal	5
WZQKPICK.EXE	2436	00	Normal	1
dfrgntfs.exe	2196	00	Normal	2
mssearch.exe	1768	00	Normal	7
dfssvc.exe	1692	00	Normal	9

Processes: 41 | CPU Usage: 100% | Commit Charge: 514M / 2440M

Figure 5-5 Resetting the dispatching priority

Once this action is successful, you will find that desktop applications are much more responsive to keyboard input and mouse actions. You can allow the runaway process to continue to run at a lower dispatching priority while you attempt to figure out what is wrong with it. You can now work at a more leisurely pace, because the runaway process can consume only as much excess processor time as is available after all other processes have been serviced. So it can continue to run at Low or Below Normal priority without causing any harm to overall system performance.

As you proceed to determine the root cause of the problem, your next step is to perform one of the following actions:

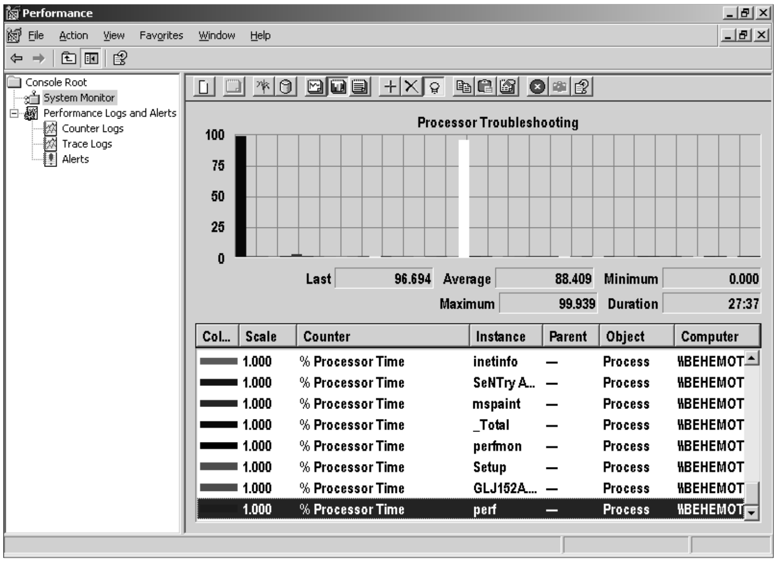
- If you are familiar with the application, use the Task Manager context menu to attach the debugger that was used to develop the application.
- If you are not familiar with the application, you can run the Kernrate utility to determine which parts of the program are consuming the most processor time.

When you are finished, you might want to end the runaway process. One approach is to use the Task Manager context menu and click End Process to end the process.

Identifying a Runaway Process by Using a Counter Log

You can also work with counter logs to diagnose a problem with a runway process by using the performance data recorded there, assuming you are gathering process level statistics.

1. Using either the Histogram view or the Report view, select all process instances of the % Processor Time counter. The Histogram view is illustrated in Figure 5-6.



Last 96.694 Average 88.409 Minimum 0.000
Maximum 99.939 Duration 27:37

Col...	Scale	Counter	Instance	Parent	Object	Computer
▬	1.000	% Processor Time	inetinfo	—	Process	\\BEHEMOT
▬	1.000	% Processor Time	SeNTry A...	—	Process	\\BEHEMOT
▬	1.000	% Processor Time	mspaint	—	Process	\\BEHEMOT
▬	1.000	% Processor Time	_Total	—	Process	\\BEHEMOT
▬	1.000	% Processor Time	perfmon	—	Process	\\BEHEMOT
▬	1.000	% Processor Time	Setup	—	Process	\\BEHEMOT
▬	1.000	% Processor Time	GLJ152A...	—	Process	\\BEHEMOT
▬	1.000	% Processor Time	perf	—	Process	\\BEHEMOT

Figure 5-6 Histogram view of process instances of the % Processor Time counter

2. Click the Freeze Frame button to freeze the display once you can see that there is a runaway process.
3. Click the Highlight button and then, using the keyboard, scroll through the legend until you can identify the instance of the process that is consuming an excessive amount of processor time.
4. Delete all the extraneous process instances, unfreeze the display, and revert to Chart view to continue to observe the behavior of the runaway process over time.

Processor Use by Module and Function

When you have a runaway process disrupting service levels on a production machine, the first step is to identify the process and remove it before it causes any further damage. You might be asked to investigate the problem further. If the application that is causing the problem was developed in-house, the programmers involved might need help in further pinpointing the problem. If the excessive processor utilization is associated with a Kernel-mode thread or device driver function, you need more information to determine *which* kernel module is involved.

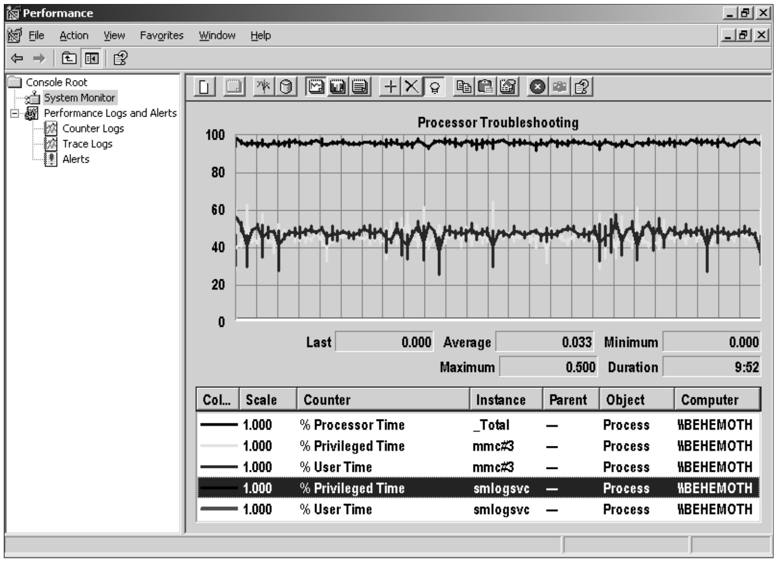
The Kernrate utility included in the *Windows Server 2003 Resource Kit* is an efficient code-profile sampling tool that you can use to resolve processor usage at the application and kernel module level, and at the function level. Using Kernrate, you can drill deep into your application and into the operating system functions that are executing.

Caution Kernrate is a potentially high-overhead diagnostic tool that can be expected to impact normal operations. The sample rate used in Kernrate can be adjusted to trade off sampling accuracy against the tool's impact on the running system. Use Kernrate carefully in a production environment and be sure to limit its use to very small measurement intervals.

The Overhead of the System Monitor

In skilled hands, the System Monitor is a very powerful diagnostic and reporting tool. It can also be misused. The following examples illustrate some of the dos and don'ts of using the System Monitor effectively.

Figure 5-7 shows a very busy system running only two applications. The first is the System Monitor console (Sysmon.ocx embedded in the Microsoft Management Console, Mmc.exe). The System Monitor console is being used very inappropriately. In a real-time monitoring session, the System Monitor is gathering every instance of every Process and Thread object and counter. This is occurring on a 1.2-GHz machine with approximately 50 active processes and 500 active threads running Windows Server 2003. Obviously, this is not something anyone would do in real life—it is an example illustrating how the System Monitor application works.


Performance
File Action View Favorites Window Help
Console Root
System Monitor
Performance Logs and Alerts
Counter Logs
Trace Logs
Alerts
Processor Troubleshooting
100
80
60
40
20
0

Last 0.000 Average 0.033 Minimum 0.000
Maximum 0.500 Duration 9:52

Col...	Scale	Counter	Instance	Parent	Object	Computer
—	1.000	% Processor Time	_Total	—	Process	\\BEHEMOTH
—	1.000	% Privileged Time	mmc#3	—	Process	\\BEHEMOTH
—	1.000	% User Time	mmc#3	—	Process	\\BEHEMOTH
—	1.000	% Privileged Time	smlogsvc	—	Process	\\BEHEMOTH
—	1.000	% User Time	smlogsvc	—	Process	\\BEHEMOTH

Figure 5-7 A very busy system running only two applications

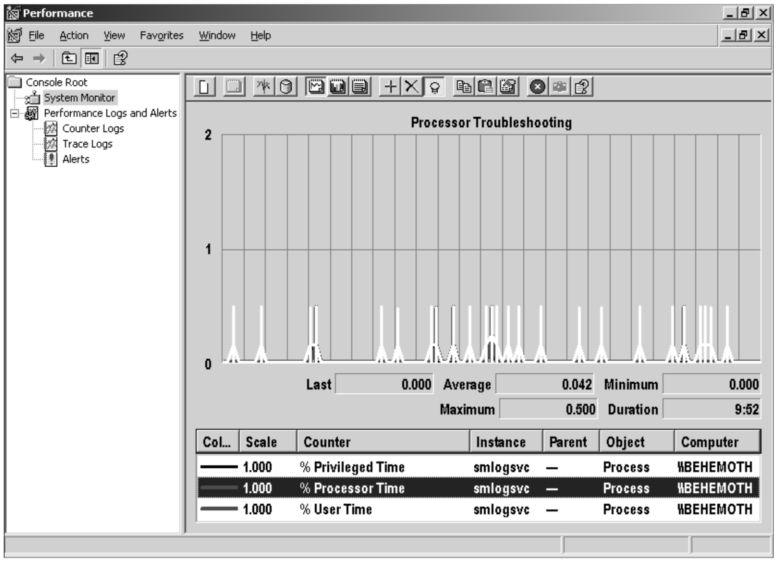
The overall system averages 83 percent busy during the approximately 10-minute interval illustrated. The Mmc.exe process that runs the System Monitor accounts for over 90 percent busy over the reporting interval. At the bottom of the chart, the % Processor Time for the Smlogsvc process is shown, broken out, like Sysmon, into % User Time and % Privileged Time. Over the same 10-minute interval, Smlogsvc accounts for less than 1 percent processor busy.

Smlogsvc is the service process that gathers performance logs and alerts to create both counter logs and trace logs. Smlogsvc also provides performance data collection services for the Logman utility. Smlogsvc runs as a background service. In this example, it is writing performance data on processor utilization to a binary log file. Smlogsvc gathered the performance data on processor usage per process, which is being reported in Figure 5-7, using the System Monitor console. Smlogsvc gathered performance data on itself and on the other main application running on the system, which happened to be the interactive System Monitor session. Smlogsvc is writing performance data once a second to a binary log file. Obviously, there is a big difference in the amount of processor overhead associated with the two performance data collectors that are running concurrently and performing similar tasks.

This example helps to answer the question, “What is the processor overhead of the System Monitor?” The answer, of course, is, “It depends.” It depends on what functions you are performing with the System Monitor. As you can see, the background data collection functions are very efficient when the binary format file is used. In foreground mode, the System Monitor is expensive, by comparison. However, no one would use the System Monitor in the foreground in the manner illustrated, unless he or she were interested in generating an excessive processor load.

The point of this exercise is to illustrate that you do not have to guess at the overhead of a system performance monitoring session. You can use the performance monitoring tools that Windows Server 2003 supplies to find out for yourself how much processor time a monitoring session uses.

Figure 5-8 provides a fairer comparison of a background performance data collection session by using Smlogsvc to gather the same process and thread objects and counters at 1-second intervals as the interactive System Monitor session gathered, as illustrated in Figure 5-7. The processor utilization is so small—approximately 1 percent busy—that it was necessary to reduce the y-axis scale of the System Monitor Chart view to render the display meaningful.



Last 0.000 Average 0.042 Minimum 0.000
Maximum 0.500 Duration 9:52

Col...	Scale	Counter	Instance	Parent	Object	Computer
	1.000	% Privileged Time	smlogsvc	—	Process	\\BEHEMOTH
	1.000	% Processor Time	smlogsvc	—	Process	\\BEHEMOTH
	1.000	% User Time	smlogsvc	—	Process	\\BEHEMOTH

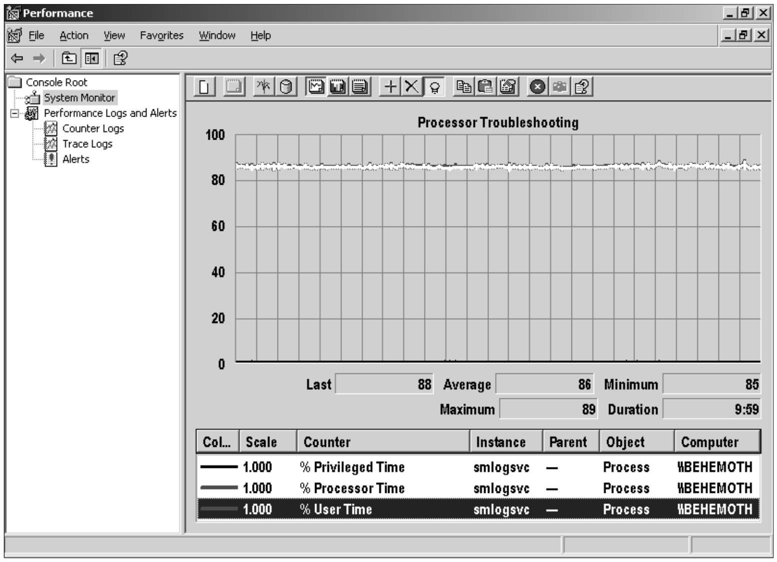
Figure 5-8 A background performance data collection session using Smlogsvc

Counter Selection Logic

Smlogsvc runs even more efficiently in this example in Figure 5-8 than it did in the previous one. The only difference in the two counter log sessions was that the first collected process level data for a few selected processes and threads. In the second example, illustrated in Figure 5-8, the Smlogsvc gathered performance data for *every* process and thread. In Figure 5-8, Smlogsvc is gathering much more performance data, but evidently executing more efficiently. As the measurement indicates, the counter log service performs more efficiently when it can gather data from a Performance Library and write that data directly to a binary log file without subjecting it to any intervening object instance or counter selection logic. This is the factor that accounts for the greater efficiency of the counter log session illustrated in Figure 5-8. Counters from all process and thread instances are being logged, compared to Figure 5-7, which shows only select instances and counters being logged.

Even though this approach is the most efficient way to gather performance data, there can be problems. When you gather all the instances and all the counters from high-volume objects like Process and Thread, the data collection files grow extremely fast. In approximately the same amount of time, the binary log file holding selected Process and Thread data grew to about 15 MB in the logging session documented in Figure 5-7. By comparison, the binary log file holding every Process and Thread counter from the logging session illustrated in Figure 5-8 grew to about 50 MB in just over 10 minutes. Although gathering all instances and counters might be the most efficient way to gather performance data from the standpoint of processor utilization, the size of the binary format files is prohibitively large, making this approach generally undesirable to take.

Log file formats Figure 5-9 illustrates another significant aspect of System Monitor overhead—the difference between using a binary file format and a text file format file to log performance data. Figure 5-9 shows another Smlogsvc data gathering session, identical to the one in Figure 5-8, except that a text format output log file is being created instead of a binary one. Processor utilization increases dramatically, as illustrated. Note also that the bulk of the processing to create a text format file takes place in User mode. Compare the proportion of Privileged-mode processing here to that shown in Figure 5-7, in which System Monitor processing in real-time mode is gathering the same set of objects and counters. In Figure 5-7, the proportion of time spent in Privileged mode was substantial. This is because of the number of Graphics Device Interface (GDI) calls required to update the Chart view every second. In Figure 5-9, the amount of Privileged-mode processing is negligible.


Performance
File Action View Favorites Window Help
Console Root
System Monitor
Performance Logs and Alerts
Counter Logs
Trace Logs
Alerts
Processor Troubleshooting
100
80
60
40
20
0

Last	88	Average	86	Minimum	85
		Maximum	89	Duration	9:59

Col...	Scale	Counter	Instance	Parent	Object	Computer
—	1.000	% Privileged Time	smlogsvc	—	Process	\\BEHEMOTH
—	1.000	% Processor Time	smlogsvc	—	Process	\\BEHEMOTH
—	1.000	% User Time	smlogsvc	—	Process	\\BEHEMOTH

Figure 5-9 Increase in processor utilization because of using a text file format file

Again, the explanation is that the counter log service operates most efficiently when it can gather data from a Performance Library and write it directly to a binary log file without much intervening processing logic. In fact, the data stored in a binary format counter log file is in a raw form, identical to the buffers returned by calls to Performance Library DLL *Collect* routines. There is no intervening processing of these raw data buffers—they are simply written directly and efficiently to the output log file.

Comparing Figure 5-7 to Figure 5-8 illustrates one form of intervening processing logic that needs to occur when only specific object and counter instances are selected for output. A call to the Perfib DLL *Collect* routine responsible for process and thread performance data returns data for all object instances and counters in a series of raw buffers. Counter selection requires parsing the raw data buffers to identify individual object instances and counters, and discarding data about objects and counters that were not selected. This selection logic is one of the functions that the Performance Data Helper (PDH) Library of routines provides. The System Monitor console application and the Smlogsvc Counter Logs and Alerts service both rely on these PDH functions to parse the raw data buffers returned by Perflib DLL data *Collect* routines. Moreover, by relying on the same set of common PDH routines to parse and interpret Perflib DLL raw data, the System Monitor console can process and display data representing current activity—either directly from the Performance Library DLLs or from binary log files in which the raw collection data is stored in its original format.

When creating a text format counter log, significant processing needs to be performed. This processing is similar to the logic that the System Monitor console applies to raw binary format data when transforming that data into displayable counters. This essential processing logic is also provided by PDH routines. These helper routines are associated with the supported performance *counter types*. Consider, for example, a PERF_COUNTER_RAWCOUNT, which is an instantaneous observation of a single measurement. An instantaneous counter requires very little processing needs—only the transformation of the original numeric value in binary format into a text representation. But consider a different counter of the popular PERF_COUNTER_COUNTER type. Here the Perflib DLL raw data buffers contain only the current value of this continuously accumulated metric. PDH routines must locate the raw value of this counter in the data buffers retained from the *previous* data collection interval to calculate an interval activity rate that is generated as the current counter value. This is not a trivial effort. First, a large number of instances of process and thread data are contained in both sets of Perflib DLL raw data collection buffers—which correspond to the current and previous measurement intervals—that must be parsed. Second, the dynamic nature of process and thread creation and destruction ensures that the sequence of data in the raw buffer can change from one interval to the next.

PDH routines exist that parse these raw data Perflib buffers and derive the current counter values, which are then generated to create text format counter logs. To be sure, these are the same PDH functions that the System Monitor console relies on to format data to generate its Chart and Report views. In Chart view, showing many counter values in the same display is impractical, so using selection logic first to trim the amount of raw data needing to be processed speeds up the processing.

The two factors affecting the overhead incurred by creating a text format counter log are:

- The number of instances and counters contained in the raw data buffers
- The need to apply selection logic to weed out instances and counters that are not written to the data file

If the text format counter log file is restricted to a few object instances and counters, much less parsing of raw data buffers is required. Counter log files limited to a few object instances and counters can be created relatively efficiently. But, as Figure 5-9 indicates, creating text format file counter logs that track large numbers of object instances and counters can be prohibitively expensive. This is especially true when voluminous raw data buffers associated with process and thread objects and counters require parsing. This relative inefficiency of text format file counter log creation underlies the recommendations in Chapter 4, “Performance Monitoring Procedures,” to use binary format log files for bulk data collection.

One mystery that still needs to be addressed is why the background logging session illustrated in Figure 5-9 uses as many processor resources as the interactive foreground session, illustrated in Figure 5-7. In both cases, because system-wide % Processor Time is pinned at 100 percent busy, processing is probably constrained by processor capacity. The implication of the System Monitor data collection functions being processor-constrained is that there is little or no processor idle time between data collection intervals. In both the foreground System Monitor console session and the background text file format counter log session, as soon as the program is finished gathering and processing one interval worth of data, it is time to gather the next. In the case of Figure 5-7, it seems reasonable to assume that GDI calls to update the Chart view are responsible for at least some of the heavy processing load. In Figure 5-9, performance logging by Smlogvc has no associated GUI overhead but uses every bit as much processing time. Using Kernrate, it will be possible to get to the bottom of this mystery.

Using Kernrate to Identify Processor Use by Module and Function

Complete documentation for the Kernrate tool is included in the *Windows S*erver 2003 Resource Kit Help. Using Kernrate, you can see exactly where processor time is spent inside processes, their modules, and module functions. Kernrate is an efficient code-profiling tool that samples processor utilization by process virtual address. This includes instructions executing inside the operating system kernel, the HAL, device drivers, and other operating system functions. It is similar to some of the CPU execution profilers available from third-party vendors.

This section illustrates using Kernrate to resolve processor usage at the process, module, and function level. This example answers the questions raised earlier in the chapter about the performance impact of the Performance Monitor application. A Kernrate CPU execution profile is useful whenever you need to understand processor usage at a finer level of detail than the Performance Monitor objects and counters can provide.

Listing 5-1 shows the output from a Kernrate monitoring session, initiated by the following command:

```
kernrate -n smlogsvc -s 120 -k 100 -v 2 -a -e -z pdh
```

Table 5-2 briefly explains the Kernrate command-line parameters. Note that if the *_NT_SYMBOL_PATH* environment variable is not set, you can specify it using the *-j* switch.

Table 5-2 Key Kernrate Run-Time Parameters

Switch	Parameter	Function
-n	*process-name*	Multiple processes can be monitored in a single run.
-s	*Duration in seconds*	Gathers instruction execution observations for the specified number of seconds. Because of the overhead of Kernrate sampling, it is wise to limit the duration of a Kernrate profile.
-k	*Hit count*	Restricts output to only those modules and buckets that equal or exceed the *Hit count* threshold. Defaults to 10.
-v	*Verbosity-level*	Controls the output that Kernrate supplies. Verbosity level 2 displays instruction information per bucket, including symbols and source-code line information for every bucket.
-i		Sets the sampling interval. More frequent sampling provides more accurate data but is more intrusive. If the CPU impact of Kernrate sampling is a concern, use a less frequent sampling interval.
-a		Performs a consolidated Kernel- and User-mode instruction execution profile.
-e		Prevents gathering of system-wide and process-specific performance metrics (context switches, memory usage, and so on) to reduce processing overhead.
-z	*module-name*	Multiple modules can be monitored in a single run.

In this example, Kernrate is monitoring the Smlogsvc process in the midst of a text file format counter log session identical to the one illustrated in Figure 5-9. This is the session that used a surprising amount of User-mode processor time. Kernrate allows you to drill down into the Smlogsvc process to see what modules and instructions are being executed. In this example, Kernrate is also used to drill down into the Pdh.dll Performance Data Helper library that it relies on to parse raw performance data buffers and calculate counter type values.

Note The Kernrate output has been edited for the sake of conciseness.

Listing 5-1 A Kernrate Report on Processor Usage

```
Starting to collect profile data

Will collect profile data for 120 seconds
===> Finished Collecting Data, Starting to Process Results

------------Overall Summary:--------------

P0     K 0:00:03.094 ( 2.6%)  U 0:01:19.324 (66.1%)  I 0:00:37.584 (31.3%)  DPC 0:00
:00.200 ( 0.2%)  Interrupt 0:00:00.991 ( 0.8%)      Interrupts= 190397, Interrupt Ra
te= 1587/sec.
```

The Kernrate Overall Summary statistics are similar to the Processor(_Total)\% Processor Time counters. They break down the execution profile samples that Kernrate gathers by processor state, including the Idle state. Note that DPC and interrupt time are included in Kernrate's calculation of the kernel time. Kernrate identifies 31.3 percent of the sample duration as time spent in the Idle thread. Kernrate examples will further illuminate the Idle thread implementation details in a moment—a discussion that is also relevant to some of the multiprocessing discussion in Chapter 6, "Advanced Performance Topics."

Continuing to review the Kernrate output in Listing 5-1, you can see that Kernrate found the processor busy executing a User-mode instruction 66.1 percent of the time and a Privileged (Kernel) mode instruction 2.6 percent of the time. The sum of Idle-, User-, and Kernel-mode processing is, of course, equal to 100 percent, after allowing for possible rounding errors.

Following the Overall Summary statistics, Kernrate lists all the processes that were active during the monitoring session (not shown), which, in this case, includes the Smlogsvc process and the Kernrate process. Kernrate then provides a hit count of the Kernel-mode modules that executed instructions during the interval, as shown in Listing 5-2.

Listing 5-2 A Kernrate Report on Processor Usage by Kernel Routines (continued)

```
Results for Kernel Mode:
-----------------------------

OutputResults: KernelModuleCount = 619
Percentage in the following table is based on the Total Hits for the Kernel

Time   20585 hits, 19531 events per hit --------
Kernel CPU Usage (including idle process) based on the profile interrupt
total possible hits is 33.50%

Module                        Hits        Shared    msec  %Total %Certain Events/Sec
ntoskrnl                     15446           0     120002    75 % 75 %   2513923
processr                      4381           0     120002    21 % 21 %    713032
hal                            524           0     120002     2 %  2 %     85283
win32k                         114           0     120002     0 %  0 %     18554
Ntfs                            42           0     120002     0 %  0 %      6835
nv4_disp                        22           0     120002     0 %  0 %      3580
nv4_mini                        12           0     120002     0 %  0 %      1953
USBPORT                         11           0     120002     0 %  0 %      1790
```

The Kernrate output explains that its resolution of instructions being executed for modules and routines in the Kernel state includes samples taken during the Idle state. Including Idle state samples, Kernel-mode instructions have been detected 33.5 percent of the time for a total of 20,585 module hits. Seventy-five percent of the module hits are in Ntoskrnl, 21 percent are in Processr.sys, and 2 percent are in Hal.dll. Other than being in the Idle state (31.3 percent), very little Kernel-mode processing is taking place.

Listing 5-3 continues the example with a Kernrate report on processor usage at the module level. The process-level statistics for the Smlogsvc process are reported next.

Listing 5-3 A Kernrate Report on Processor Usage by Module Level (continued)

```
Results for User Mode Process SMLOGSVC.EXE (PID = 2120)

OutputResults: ProcessModuleCount (Including Managed-Code JITs) = 41
Percentage in the following table is based on the Total Hits for this Process

Time   40440 hits, 19531 events per hit --------
User-
Mode CPU Usage for this Process based on the profile interrupt total possible hits i
s 65.82%

 Module               Hits       Shared    msec  %Total %Certain Events/Sec
pdh                  31531          0     120002    77 % 77 %   5131847
kernel32              6836          0     120002    16 % 16 %   1112597
msvcrt                1700          0     120002     4 %  4 %    276684
ntdll                  362          0     120002     0 %  0 %     58917
```

Smlogsvc has 40,440 hits, which means the machine is executing Counter Logs and Alerts service User-mode instructions 65.8 percent of the time. Inside Smlogsvc, only four modules have more than the threshold number of hits. Calls to the Pdh.dll Performance Data Helper function account for 77 percent of the User-mode processing within the Smlogsvc. The Kernel32.dll and Msvcrt.dll run-time libraries account for 16 percent and 4 percent of the Smlogsvc hits, respectively. The fourth runtime library, Ntdll.dll, yields less than 1 percent of the instruction execution hits.

Listing 5-4 is the Kernrate report on processor usage by module function. The module-level statistics for Pdh.dll provide additional insight.

Listing 5-4 A Kernrate Report on Processor Usage by Module Function (continued)

```
===> Processing Zoomed Module pdh.dll...


----- Zoomed module pdh.dll (Bucket size = 16 bytes, Rounding Down) --------
Percentage in the following table is based on the Total Hits for this Zoom Module

Time   31671 hits, 19531 events per hit -------- (33371 total hits from summing-
up the module components)
(51.55% of Total Possible Hits based on the profile interrupt)

 Module                      Hits      Shared    msec  %Total %Certain Events/Sec
StringLengthWorkerA         23334         1     119992    69 % 69 %   3798056
IsMatchingInstance           1513         0     119992     4 %  4 %    246269
GetInstanceByName            1374         0     119992     4 %  4 %    223644
IsMatchingInstance           1332         0     119992     3 %  3 %    216808
NextInstance                 1305        12     119992     3 %  3 %    212413
GetInstanceName               829         0     119992     2 %  2 %    134935
PdhiHeapFree                  495       428     119992     1 %  0 %     80570
GetInstance                   469        11     119992     1 %  1 %     76338
PdhiHeapReAlloc               428         0     119992     1 %  1 %     69665
GetCounterDataPtr             338       338     119992     1 %  0 %     55015
NextCounter                   246         1     119992     0 %  0 %     40041
GetInstanceByName             233       229     119992     0 %  0 %     37925
FirstInstance                 121       121     119992     0 %  0 %     19695
PdhiHeapFree                  121         0     119992     0 %  0 %     19695
PdhiMakePerfPrimaryLangId     108       108     119992     0 %  0 %     17579
PdhiWriteTextLogRecord        101         0     119992     0 %  0 %     16439
GetObjectDefByTitleIndex       93         3     119992     0 %  0 %     15137
FirstObject                    84        30     119992     0 %  0 %     13672
GetPerfCounterDataPtr          84         0     119992     0 %  0 %     13672
GetInstanceByUniqueId          77        77     119992     0 %  0 %     12533
GetQueryPerfData               75         0     119992     0 %  0 %     12207
_SEH_prolog                    53         0     119992     0 %  0 %      8626
NextObject                     40        39     119992     0 %  0 %      6510
FirstInstance                  36        13     119992     0 %  0 %      5859
UpdateRealTimeCounterValue     31         0     119992     0 %  0 %      5045
_SEH_epilog                    31         5     119992     0 %  0 %      5045
PdhiPlaInitMutex               30        30     119992     0 %  0 %      4883
GetStringResource              30         0     119992     0 %  0 %      4883
```

Kernrate reports that the PDH module has 31,671 hits, accounting for 51.6 percent of the total instruction samples collected. Fully 69 percent of these hits are associated with an internal helper function called *StringLengthWorkerA*, which is called repeatedly to parse current and previous raw data buffers. Additional raw data buffer parsing helper functions such as *IsMatchingInstance*, *GetInstanceByName*, *IsMatchingInstance*, *NextInstance*, *GetInstanceName*, *GetInstance*, *GetCounterDataPtr*, *NextCounter*, and *GetInstanceByName* account for another 20 percent of the instructions inside Pdh.dll. The column that indicates the number of shared hits refers to address range buckets that span function boundaries and make identification less than certain. The default address bucket range is 16 bytes, an alignment that is consistent with the optimized output from most compilers. If too many uncertain hits occur, the bucket size can be adjusted downward using the *-b* command-line switch.

The details Kernrate provides at the Module level are mainly of interest to the programmers responsible for building the software module and maintaining it, so exploring the inner workings of PDH any further isn't necessary here.

Idle thread processing Kernrate can also be used to drill down into Kernel-mode modules, which allows you to identify device drivers or other Kernel-mode functions that are consuming excessive CPU time. This function of Kernrate is illustrated in the following code—it drills down into Ntoskrnl and Processr.sys to illuminate the mechanism used to implement the Idle thread that plays a distinctive role in the processor utilization measurements.

Under circumstances identical to the previous example that illustrated the use of Kernrate and produced the output in Listings 5-1 through 5-4, the following Kernrate command was issued to drill down into the Ntoskrnl.exe and Processr.sys modules that, along with the HAL, lie at the heart of the Windows Server 2003 operating system:

```
kernrate -s 120 -k 10 -v 2 -x -a -e -z processr -z ntoskrnl
```

This Kernrate session gathered Module level hits against these two modules. Listing 5-5 shows a Kernrate report on processor usage for the Processr.sys Kernal module by function.

Listing 5-5 A Kernrate Report on Processor Usage by Function

```
===> Processing Zoomed Module processr.sys...


----- Zoomed module processr.sys (Bucket size = 16 bytes, Rounding Down) -------
-
Percentage in the following table is based on the Total Hits for this Zoom Module

Time   4474 hits, 19531 events per hit -------- (4474 total hits from summing-
up the module components)
(7.28% of Total Possible Hits based on the profile interrupt)

 Module                  Hits        Shared    msec  %Total %Certain Events/Sec
AcpiC1Idle               4474          0     120002    100 % 100 %      728168
```

Kernrate found instructions being executed inside the Processr.sys module 7.28 percent of the time. The Processr.sys device driver module is a new feature of the Windows Server 2003 operating system. It provides hardware-specific processor support, including implementation of the processor power management routines. Depending on the processor model and its feature set, the operating system will provide an optimal Idle thread implementation. In a multiprocessor configuration, for example, the operating system will issue no-operation (NOP) instructions inside the Idle loop to ensure no memory bus traffic can clog up the shared memory bus. Here, a call is made to the Advanced Configuration and Power Interface (ACPI) C1 Idle routine because the target processor is a machine that supports this power management hardware interface.

Listing 5-6 A Kernrate Report on Processor Usage for Ntoskrnl.exe by Function

```
===> Processing Zoomed Module ntoskrnl.exe...


----- Zoomed module ntoskrnl.exe (Bucket size = 16 bytes, Rounding Down) -------
-
Percentage in the following table is based on the Total Hits for this Zoom Module

Time   15337 hits, 19531 events per hit -------- (15639 total hits from summing-
up the module components)
(24.96% of Total Possible Hits based on the profile interrupt)

 Module                      Hits   Shared    msec  %Total %Certain Events/Sec
KiIdleLoop                  14622      0     120002    93 % 93 %   2379812
KiDispatchInterrupt           104    104     120002     0 %  0 %     16926
KiSwapContext                 104      0     120002     0 %  0 %     16926
READ_REGISTER_BUFFER_UCHAR     46     46     120002     0 %  0 %      7486
READ_REGISTER_ULONG            46      0     120002     0 %  0 %      7486
```

```
FsRtlIsNameInExpressionPrivate  44      0      120002      0 %   0 %       7161
FsRtlIsNameInExpressionPrivate  39      0      120002      0 %   0 %       6347
READ_REGISTER_USHORT            30     30      120002      0 %   0 %       4882
READ_REGISTER_UCHAR             30      0      120002      0 %   0 %       4882
KiTrap0E                        24      0      120002      0 %   0 %       3906
KiSystemService                 24      0      120002      0 %   0 %       3906
WRITE_REGISTER_USHORT           15     15      120002      0 %   0 %       2441
WRITE_REGISTER_UCHAR            15      0      120002      0 %   0 %       2441
FsRtlIsNameInExpressionPrivate  14      0      120002      0 %   0 %       2278
ExpCopyThreadInfo               13      0      120002      0 %   0 %       2115
KiXMMIZeroPagesNoSave           13     13      120002      0 %   0 %       2115
KiTimerExpiration               13      0      120002      0 %   0 %       2115
ObReferenceObjectByHandle       10      0      120002      0 %   0 %       1627
SwapContext                     10      0      120002      0 %   0 %       1627
```

Ninety-three percent of the module hits inside Ntoskrnl are for the KiIdleLoop routine. This is the Idle thread routine that eventually calls the Processr.sys AcpiC1Idle function.

In summary, Kernrate is a processor instruction sampling tool that allows you to profile processor usage at a considerably more detailed level than the Performance Monitor counters allow. Kernrate can be used to understand how the processor is being used at the module and instruction level. It is capable of quantifying processor usage inside system functions, device drivers, and even operating system kernel routines and the HAL.

Memory Troubleshooting

Memory problems—either a shortage of memory or poorly configured memory—are a common cause of performance problems. This section looks at two types of memory bottlenecks. The first is a shortage of RAM, or physical memory. When there is a shortage of RAM, the virtual memory manager (VMM) component, which attempts to keep the most recently accessed virtual memory pages of processes in RAM, must work harder and harder. Performance might suffer as paging operations to disk increase, and these paging operations can interfere with applications that need to access the same disk on which the paging file (or files) is located. Even though excessive paging to disk is a secondary effect of a RAM shortage, it is the symptom that is easiest to detect. Examples are discussed that illustrate how to identify a system with a shortage of RAM that is encountering this type of memory bottleneck.

A second type of memory bottleneck occurs when a process exhausts the amount of virtual memory available for allocation. Virtual memory can become depleted by a process with a *memory leak*, the results of which, if undetected, can be catastrophic. The program with the leak might fail, or it might cause other processes to fail because of a shortage of resources. Memory leaks are usually program defects.

Normal server workload growth can also lead to a similar shortage of virtual memory. Instead of a virtual memory leak, think of this situation as virtual memory *creep*. Virtual memory creep is very easy to detect and avoid. There is an example later in this chapter that illustrates what to look for to diagnose a memory leak or virtual memory creep. A useful technique to use in memory capacity planning is also discussed in Chapter 6, “Advanced Performance Topics.”

The memory on a computer is not utilized in quite the same fashion as other hardware resources. You cannot associate memory utilization with specific requests for service, for example, or compute a service time and response time for memory requests. Program instructions and data area *occupy* physical memory to execute. They often occupy physical memory locations long after they are actively addressed. A program’s idle virtual memory code and data areas are removed from RAM only when new requests for physical memory addresses cannot be satisfied from current supplies of unallocated (or *available*) RAM. Another factor that complicates the memory utilization measures is that RAM tends to look fully utilized all the time because of the way a process’s virtual memory address space is mapped to physical memory on demand.

The statistics that are available to measure memory utilization reflect this dynamic policy of allocating virtual memory on demand. The performance measurements include instantaneous virtual memory allocation counters, instantaneous physical memory allocation counters, and continuous interval counters that measure paging activity. These measurements are available at both the system and process levels.

Counters to Evaluate When Troubleshooting Memory Performance

The first step in analyzing memory problems is determining whether the problem is a result of insufficient available physical memory leading to excessive paging. Even though insufficient available memory can cause excessive paging, excessive paging can occur even when there is plenty of available memory, when, for example, an application is functioning improperly and leaking memory.

Ideally, compare the values of the counters listed in Table 5-3 to the value of these same counters that you archived in your baseline analysis. If you do not have a baseline analysis to go by, or the system has changed considerably since you last made baseline measurements, the suggested thresholds listed in Table 5-3 can be used as very rough usage guidelines.

Table 5-3 Memory Performance Counters to Evaluate

Counter	Description	Suggested Threshold
Memory\% Committed Bytes in Use	This value should be relatively stable during a long-term view.	Investigate if greater than 80%.
Memory\Available Bytes	If this value is low, check the Memory\Pages/sec counter value. Low available memory and high paging indicate a memory shortage resulting from an excessive application load or a defective process.	Investigate if less than 5% of the size of RAM. Alarm if less than 0.5% of the size of RAM.
Memory\Commit Limit	This value should stay constant, indicating an adequately sized paging file. If this value increases, the system enlarged the paging file for you, indicating a prolonged virtual memory shortage.	Investigate if the trend of this value is increasing over time.
Memory\Committed Bytes	This represents the total virtual memory allocated by the processes on the system. If it increases over an extended period of time, a process might be leaking memory	Investigate if the trend of this value is increasing over time.
Memory\Pages/sec	Tracks page fault pages generated by read (input) and write (output) operations. If this value is high, check the Pages Input/sec to see whether application(s) are waiting for pages that could slow response time.	Depends on page file disk speed. Additional investigation might be required when there are more than 40 per second on slow disks or more than 300 per second on faster disks.
Memory\Pages Input/sec	Tracks page faults requiring data to be read from the disk. Unlike output pages, the application must wait for this data to be read, so application response time can be slowed if this number is high. Check the disk % Idle Time to see whether the page file drive is so busy that paging performance might be adversely affected.	Varies with disk hardware and system performance. More than 20 might be a problem on slow disk drives, whereas faster drives can handle much more.

Table 5-3 **Memory Performance Counters to Evaluate**

Counter	Description	Suggested Threshold
Memory\Pool Nonpaged Bytes	Tracks memory that is always resident in physical memory. Primarily device drivers use this memory. The value of this counter should be relatively stable. An increasing value over time might indicate a pool memory leak.	Investigate if Pool Nonpaged Bytes is running at > 80% of its maximum configured pool size.
Memory\Pool Paged Bytes	Tracks memory that can be paged out of physical memory. Any service or application can use this memory. The value of this counter should be relatively stable. An increasing value over time might indicate a pool memory leak.	Investigate if Pool Paged Bytes is running at > 70% of its maximum configured pool size.
Process(_Total)\Private Bytes	Monitors the sum of all private virtual memory allocated by all the processes running on that system. If this value increases over a long period of time, an application might be leaking memory.	Investigate if the trend of this value is increasing over time.
LogicalDisk(pagefile drive)\% Idle Time	Monitors the idle time of the drive (or drives) on which the paging file resides. If this disk is too busy (that is, has a very low idle time), virtual memory operations to that disk will slow down.	Investigate paging drives with less than 50% Idle Time.
LogicalDisk(pagefile drive)\Split I/O/sec	Monitors the rate that Split I/Os are occurring on the drive (or drives) with the paging file(s). A higher than normal rate of Split I/Os on a drive with a paging file can cause virtual memory operations to that disk to take longer.	The threshold value for this counter depends on the disk drive type and configuration.

If the paging file shows a high degree of usage, the paging file might be too small for the applications you are running on the system. Likewise, a disk that holds the paging file or files that is too busy can also impact overall performance.

Memory leaks in applications are indicated in several places. First, you might get an error message indicating the system is low on virtual memory. If you have logged performance data on the computer over a period of time, a memory leak in an application process will show up as a gradual increase in the value of the Memory\Committed Bytes counter, as well as an increase in the value of the Process(_Total)\Private Bytes counter. A memory leak in one process might also cause excessive paging by squeezing other process working sets out of RAM. An example of this condition is discussed later in "Virtual Memory Leaks."

What to Check Next When Troubleshooting Memory Performance

If you determine that the system needs more physical memory, you can either install more physical memory or move applications to another computer to relieve the excessive load. If you decide that you do not have a physical memory shortage or problem, the next step is to evaluate another component—for example, the processor or disk—depending on the value of other performance counters.

If memory seems to be a problem, the next step is to determine the specific cause. Sometimes in a large Terminal Services environment, there is simply too much demand for memory from multiple application processes. Other times, you can isolate a memory problem to a specific process, and most likely that problem will be the result of one of two situations: either an application needs more memory than is installed on the system, or an application has a problem that needs to be fixed. If the memory usage of a specific process rises to a certain level and stabilizes, you can increase available virtual memory by expanding the paging file. Eventually, if the application is not defective, its memory consumption should stabilize. If the amount of physical memory installed is inadequate, the application might perform poorly, and it might cause a significant amount of paging to disk. However, its consumption of virtual memory normally will not increase forever.

If the application is defective and continually consumes more and more memory resources, its memory usage, as indicated by the Process(*ProcessName*)\Private Bytes performance counter, will constantly increase over time. This situation is known as a *memory leak*—where memory resources are reserved and used but not released when they are no longer required. Depending on the severity of the leak, this condition of using memory but not releasing it can consume all available memory resources in a matter of days, hours, or even minutes. Generally, the serious leaks are caught before an application is released to customers, so only the slow leaks are left to be noticed by the end users of the production application. Consequently, a counter log file that tracks memory usage over a long period of time is the best way to detect slow memory leaks before they cause problems in the rest of the system. The example performance monitoring procedures recommended in Chapter 4, "Performance Monitoring Procedures," incorporate the measurements you need to identify a system with a memory leak.

If the paging file is fragmented or is located on a disk that is heavily used by other applications, memory performance can be degraded even though there is no shortage of either physical or virtual memory. A consistently low value of the LogicalDisk(*PageFileDrive*)\% Idle Time performance counter indicates that the disk is very busy, which might contribute to degraded memory performance. Moving the paging file to another disk drive might improve performance in this situation. If the value of the LogicalDisk(*PageFileDrive*)\Split I/O/sec counter is high on the drive that contains the paging file, the disk or the paging file might be fragmented. If either is the case, accessing that disk is going to take longer than it would without the fragmentation. Defragmenting the drive or moving the paging file to a less crowded disk should improve memory performance.

Tip The built-in Disk Defragmenter tool does not defragment paging files. To defragment the paging file, either use a third-party defragmenter that supports this feature or follow the procedure outlined in article 229850 "Analyze Operation Suggests Defragmenting Disk Multiple Times" in the Microsoft Knowledge Base at http://support.microsoft.com.

Excessive Paging

You want to install enough RAM to prevent *excessive* paging from impacting performance, but you should not attempt to install enough RAM to eliminate paging activity completely. In a virtual memory computer system, some page fault behavior—for instance, when a program first begins to execute—is inevitable. Modified virtual pages in memory have to be updated on disk eventually, so some amount of Page Writes/sec is also inevitable.

Two types of serious performance problems can occur if too little RAM is available:

- **Too many page faults** Too many page faults leads to excessive program execution delays.
- **Disk contention** Virtual memory machines that sustain high page-fault rates might also encounter disk performance problems.

Too many page faults is the more straightforward performance problem associated with virtual memory and paging. Unfortunately, it is also the one that requires the most intense data gathering to diagnose. A commonly encountered problem occurs when disk performance suffers because of excessive paging operations to disk. Even though it is a secondary effect, it is often the easier condition to recognize. The extra disk I/O activity resulting from paging can easily interfere with applications attempting to access their data files stored on the same disk as the paging file.

Table 5-4 Primary Indicators of a Memory Bottleneck

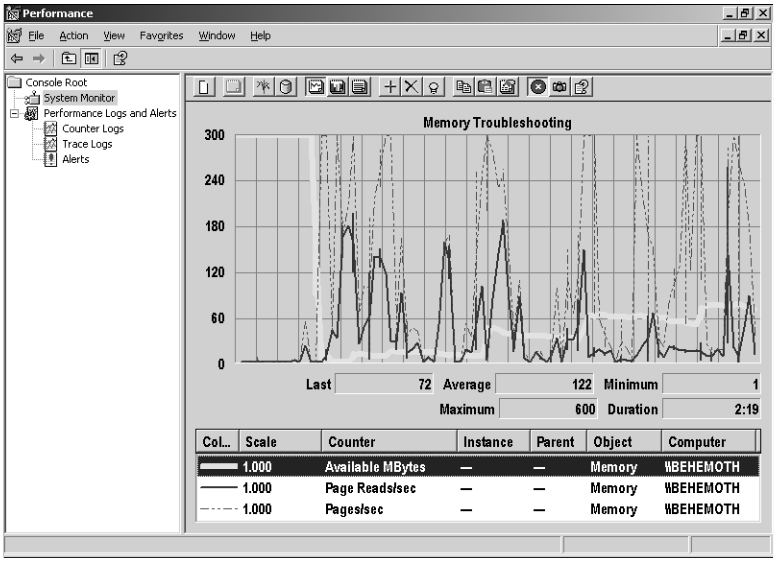
Counter	Primary Indicator	Threshold Values
Memory\Pages/sec	Paging operations to disk (Pages input + Pages output)	Pages/sec × 4K page size > 70% of the total number of Logical Disk Bytes/sec to the disk(s) where the paging file is located.
Memory\Page Reads/sec	Page faults that were resolved by reading the disk	Sustained values > 50% of the total number of Logical Disk operations to the disk(s) where the paging file is located.
Memory\Available Bytes	Free (unallocated) RAM	Available Bytes < 5% of the size of RAM is likely to mean there is a shortage of physical memory.

Table 5-4 shows three primary indicators of a shortage of RAM, and these indicators are all interrelated. The overall paging rate to disk includes both Page Reads/sec and Page Writes/sec. Because the operating system must ultimately write changed pages to disk, it is not possible to avoid most page write operations. Page Reads/sec—the hard page fault rate—is the measurement most sensitive to a shortage of RAM. As Available Bytes—the pool of unallocated RAM—becomes depleted, the number of hard page faults that occur normally increases. The total number of Pages/sec that the system can sustain is a function of disk bandwidth. When the disk or disks where the paging files are located become saturated, the system reaches an upper limit for sustainable paging activity to disk. However, because paging operations consist of a mixture of sequential and random disk I/Os, you will discover that this limit on paging activity is quite elastic. The performance of disks on sequential and random access workloads is discussed further in the section entitled "Establishing a Disk Drive Performance Baseline" later in this chapter.

Because the limit on the number of Pages/sec that the system can read and write is elastic, no simple rule-of-thumb approach is adequate for detecting *thrashing*, the classic symptom of a machine that is memory constrained. A better approach is to compare the amount of disk traffic resulting from paging to overall disk operations. If paging accounts for only 20 percent or less of total disk operations, the impact of virtual memory management is tolerable. If paging accounts for 70 percent or more of all disk operations, the situation is probably not tolerable.

Figure 5-10 illustrates these points, showing a system that is paging heavily during a 2-minute interval. The number of available bytes plummets about 30 seconds into this monitoring session, when the Resource Kit resource consumer tool, Consume.exe, is launched to create a shortage of RAM on this machine:

```
C:\>consume -physical-memory -time 600
Consume: Message: Time out after 600 seconds.
Consume: Message: Successfully assigned process to a job object ...
Consume: Message: Total physical memory: 1FF6F000
Consume: Message: Available physical memory: 7036000
Consume: Message: Will attempt to create 1 baby consumers ...
Consume: Message: Sleeping ...
```



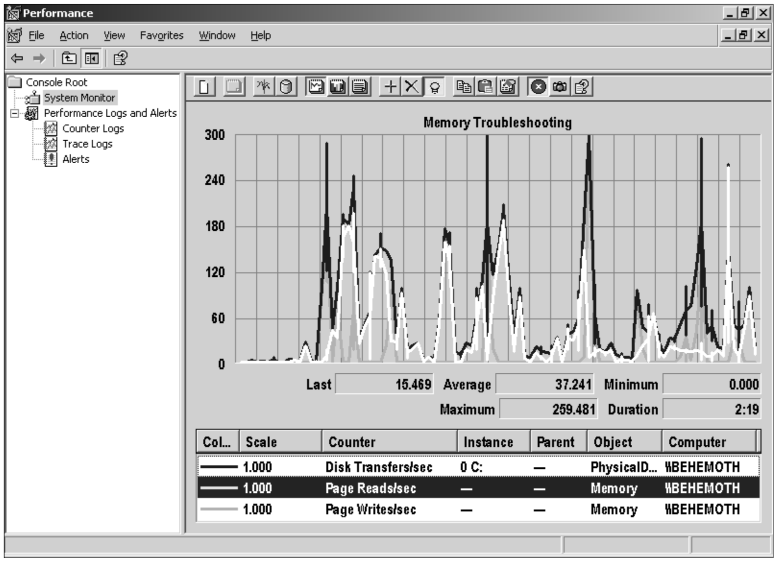
Last 72 Average 122 Minimum 1
Maximum 600 Duration 2:19

Col...	Scale	Counter	Instance	Parent	Object	Computer
━━	1.000	Available MBytes	—	—	Memory	\\BEHEMOTH
——	1.000	Page Reads/sec	—	—	Memory	\\BEHEMOTH
-·-·-	1.000	Pages/sec	—	—	Memory	\\BEHEMOTH

Figure 5-10 A system that is paging heavily during a 10-minute interval

When the consume process acquired 600 MB of virtual memory to create a memory shortage, available bytes dropped to near zero. As server applications continued to execute, they encountered serious paging delays. The operating system was forced to perform 37 Page Reads/sec on average during this period of shortage. A spike of over 300 Page Reads/sec occurred during one interval, along with several peak periods in which the number of Page Reads/sec exceeded 200.

At this rate, paging activity will consume almost all the available disk bandwidth. You can see this consumption better in Figure 5-11, which compares Page Reads/sec and Page Writes/sec to Disk Transfers/sec. It is apparent that almost all disk activity at this point results from page fault resolution. This is a classic illustration of a virtual memory system that is paging heavily, possibly to the detriment of its designated I/O workload execution.



Last	15.469	Average	37.241	Minimum	0.000
		Maximum	259.481	Duration	2:19

Col...	Scale	Counter	Instance	Parent	Object	Computer
——	1.000	Disk Transfers/sec	0 C:	—	PhysicalD...	\\BEHEMOTH
——	1.000	Page Reads/sec	—	—	Memory	\\BEHEMOTH
——	1.000	Page Writes/sec	—	—	Memory	\\BEHEMOTH

Figure 5-11 Comparing Page Reads/sec and Page Writes/sec to Disk Transfers/sec

Because the operating system often attempts bulk paging operations, especially on Page Writes, comparing bytes moved for paging operations to total Disk Bytes/sec is a better way to determine the amount of disk capacity devoted to paging. Multiply the Memory\Pages/sec counter by 4096, the size of an IA-32 page, to calculate bytes moved by disk paging. Compare that value to Logical Disk\Disk Bytes/sec. If the percentage of the available disk bandwidth devoted to paging operations exceeds 50 percent, paging potentially will impact application performance. If the percentage of the available disk bandwidth devoted to memory management–initiated paging operations exceeds 70 percent, the system is probably experiencing excessively high paging rates, that is, performing too much work in virtual memory management and not enough of its application-oriented physical disk work.

The standard remedy for a system that is paging too much is to add RAM and increase the pool of Available Bytes. This will fix most performance problems resulting from excessive paging. If you cannot add RAM to the machine, other remedies include configuring faster paging disks, more paging disks, or a combination of both. Any of the disk tuning strategies discussed later that improve disk service time can also help, including defragmentation of the paging file disk.

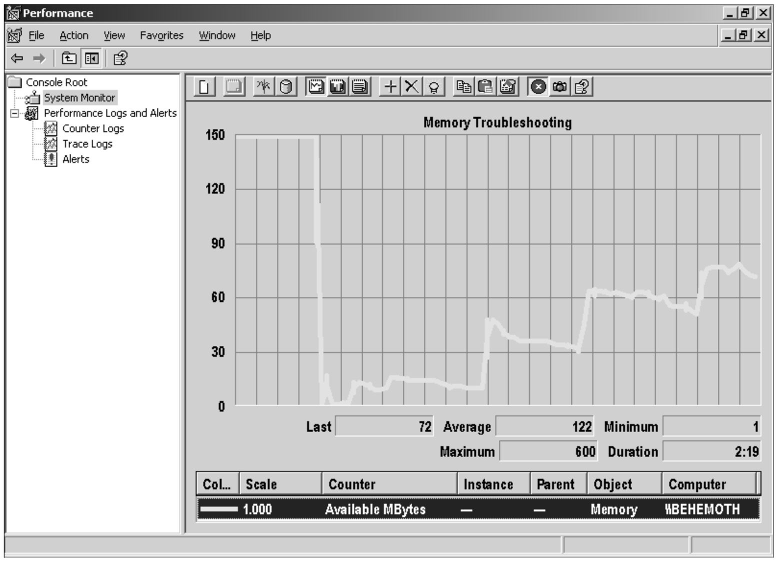
Caution Some paging activity cannot be eliminated entirely by adding more RAM. Page Reads that occur at process initialization cannot be avoided by adding more RAM. Because the operating system must ensure that modified pages are current on the paging file, many Page Writes cannot be avoided either. Demand zero paging is the allocation of virtual memory for new data areas. Demand zero paging occurs at application startup and, in some applications, as new data structures are allocated during run time. Adding memory will often reduce paging and improve memory efficiency, but generally does not reduce the demand zero fault rate.

Available Memory

The other primary indicator of a memory bottleneck is that the pool of Available Bytes has become depleted. Understanding how the size of the Available Bytes pool affects paging is very important. This relationship is so important that three Available Bytes counters are available: one that counts bytes, one that counts kilobytes, and a third that counts megabytes. Page trimming by the virtual memory manager is triggered by a shortage of available bytes. Page trimming attempts to replenish the pool of Available Bytes by identifying virtual memory pages that have not been referenced for a relatively long time. When page trimming is effective, older pages that are trimmed from process working sets are not needed again soon. These older pages are replaced by more active, recent pages. Trimmed pages are marked in transition and remain in RAM for an extra period of time to reduce the amount of paging to disk that occurs. Dirty pages must be written to disk if more pages are on the Modified List than the list's threshold value allows, so there is usually some cost associated with page trimming even though the process is very effective.

If there is a chronic shortage of Available Bytes, page trimming loses effectiveness and leads to more paging operations to disk. There is little room in RAM for pages marked in transition; therefore, when recently trimmed pages are referenced again, they must be accessed from disk instead of RAM. When dirty pages are trimmed frequently, more frequent updates of the paging file are scheduled. This paging to disk interferes with application-directed I/O operations to the same disk.

Following a round of page trimming, if the memory shortage persists, the system is probably in store for more page trimming. Figure 5-12 zooms in on the value of Available MBytes during the same period shown in Figure 5-10, when Consume.exe was active. Notice that following the initial round of page trimming to replenish the Available Bytes pool, the value of Available MBytes increases, but in an oscillating manner depending on how effective the previous round of page trimming was. It is apparent that additional rounds of page trimming are initiated because the physical memory shortage persists. When a persistent memory shortage occurs, page trimming can combat the problem, but do little to relieve the shortage for good. The only effective way to relieve the shortage is to add RAM to the machine.



Last 72 Average 122 Minimum 1
Maximum 600 Duration 2:19

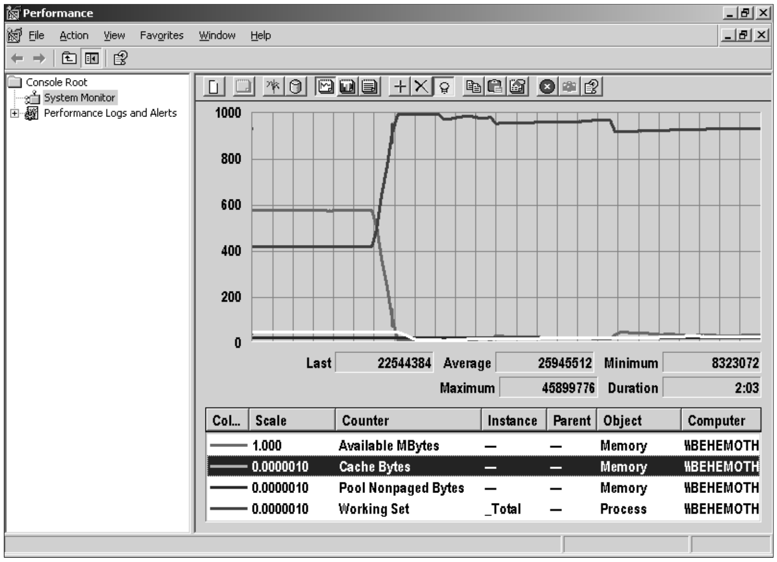
Col...	Scale	Counter	Instance	Parent	Object	Computer
▬	1.000	Available MBytes	—	—	Memory	\\BEHEMOTH

Figure 5-12 The value of Available MBytes oscillates between rounds of page trimming

As a general rule, you can avoid a memory shortage by ensuring that Available Bytes does not drop below 5 percent of RAM for an extended period of time. However, this rule of thumb can be fallible if you are running server applications that manage their own working sets. Applications that can manage their own working sets include Microsoft Internet Information Services (IIS) 6.0, Microsoft Exchange Server, and Microsoft SQL Server. As described in Chapter 1, "Performance Monitoring Overview," these applications interact with the virtual memory manager to expand their working sets when free RAM is ample and contract them when RAM is depleted. These applications rely on RAM-resident cache buffers to reduce the amount of I/O they must direct to disk. When you are running server applications that manage their own working sets, RAM will always look full. Available Bytes will remain within a narrow range, and the only reliable indicator of a RAM shortage will be a combination of more paging to disk and less effective application caching.

Memory Allocation

After you determine that physical memory is saturated and paging is excessive, exploring the way physical memory is allocated is often useful. Figure 5-13 shows the four major counters that tell you how RAM is allocated.



Last	22544384	Average	25945512	Minimum	8323072
		Maximum	45899776	Duration	2:03

Col...	Scale	Counter	Instance	Parent	Object	Computer
	1.000	Available MBytes	—	—	Memory	\\BEHEMOTH
	0.0000010	Cache Bytes	—	—	Memory	\\BEHEMOTH
	0.0000010	Pool Nonpaged Bytes	—	—	Memory	\\BEHEMOTH
	0.0000010	Working Set	_Total	—	Process	\\BEHEMOTH

Figure 5-13 The four major counters indicate how RAM is allocated

The following memory allocation counters are shown against a scale that reflects the size of RAM on the machine in Figure 5-13, namely 1 GB:

- **Memory\Available Bytes** RAM that is currently available for immediate allocation. Available Bytes is the sum of the Zero, Free, and Standby lists.
- **Memory\Cache Bytes** The pageable working set associated with allocated system memory. Cache Bytes is the sum of four counters: System Cache Resident Bytes, System Driver Resident Bytes, System Code Resident Bytes, and Pool Paged Resident Bytes.
- **Memory\Pool Nonpaged Bytes** The current nonpageable pool allocation.
- **Process(_Total)\Working Set** The sum of each process's current working set. Resident pages from shared DLLs are counted as part of every process address space that loaded the DLL.

You do not have to use the default scaling factor associated with these counters; they have all been adjusted to use a scale factor of .000001. Presentation in this straightforward format, with all the measurements on the chart reported in MB, prevents any misinterpretation or distortion of the measurements reported.

As Available Bytes is consumed, this type of report allows you to determine what kind of memory allocations are responsible for the increase in memory usage. The system working set can be further broken down into four classes of allocation requests: the System Cache Resident Bytes, System Driver Resident Bytes, System Code Resident Bytes, and Pool Paged Resident Bytes. The working set sizes of individual process address spaces are also available, but only at the cost of data gathering at the process level. The processes responsible for pageable and nonpageable pool allocations can also be identified.

Caution The four physical memory allocation counters depicted in Figure 5-13 cannot be added together reliably to determine the exact size of RAM. Dirty pages currently on the Modified list that still need to be written to the paging file are not included in any of these counter values. In addition, resident pages from shared DLLs are counted in the working set of each process address space where they are loaded, so they are subject to being counted multiple times.

Note that the only way to determine the number of dirty pages currently resident on the Modified List is to use the *!vm* command extension with the Kernel Debugger. Using the *!vm* command extension is illustrated later.

Another way to view the same physical memory allocation data is to relog the binary counter log data to text format and use Excel to create a stacked area or bar chart, as illustrated in Figure 5-14.

The advantage of the stacked format is that the charted values are cumulative. It is easier to see how the changes in the value of one of the memory allocation counters affect the overall size of the pool of Available Bytes. The page trimming operations performed by the virtual memory manager, which remove older pages from process working sets and add them to the pool of Available Bytes, are also clearly visible when you look at the measurement data in this format.

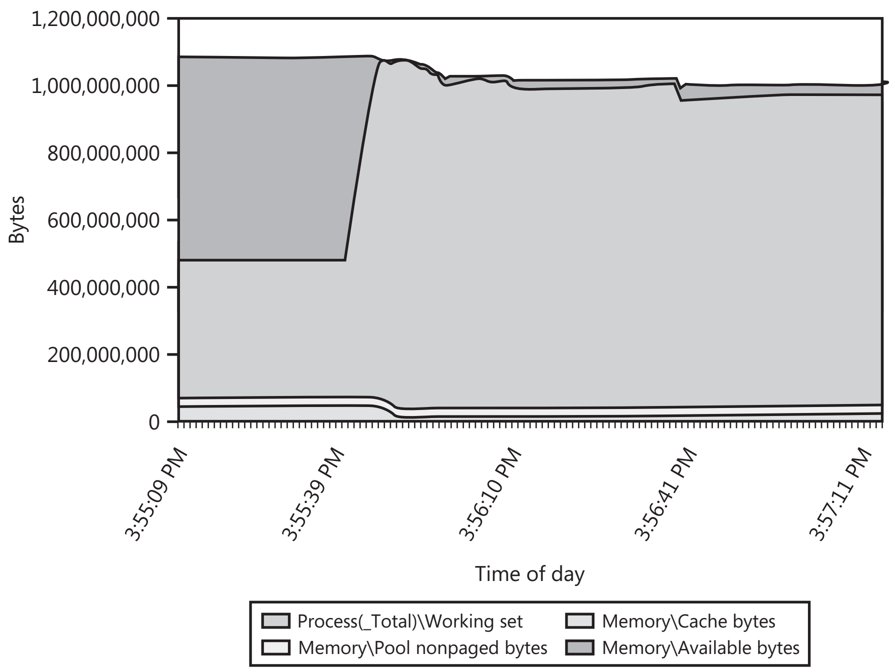


Figure 5-14 Using an Excel chart to view physical memory allocation data

Other Memory Reporting Tools

The System Monitor counters are not the only means of viewing the memory allocation statistics. The Processes tab in Task Manager allows you to view the current process working set (Memory Usage), total virtual memory allocations, and virtual memory allocations to the system pools. Task Manager also offers you the convenience of sorting the Processes display based on the values in any one of the measurement fields currently selected for display. If you need to identify a process address space that is consuming too much RAM, using Task Manager is quick and easy. Meanwhile, the Performance tab in Task Manager reports available memory, the current size of the System Cache, and current physical memory allocations for the Paged and Nonpaged pools.

When you use a Kernel Debugger session, you can view additional information by issuing the *!vm* command extension. The *!vm* command returns a number of interesting statistics that are mainly concerned with virtual memory allocations. The *!vm* command also reports the number of dirty pages currently resident on the Modified Page List, as illustrated in Listing 5-7. This example shows a snapshot of the virtual memory usage information from the *!vm* command for the machine in Figure 5-10 before and after the Consume.exe tool was run.

Before Consume.exe executes, 489 dirty pages are awaiting output to disk on the Modified Page List, as shown in Listing 5-7.

Listing 5-7 Output from the Kernel Debugger *!vm* Command Extension

```
lkd> !vm

*** Virtual Memory Usage ***
   Physical Memory:    130927   (  523708 Kb)
   Page File: \??\C:\pagefile.sys
      Current:    786432Kb Free Space:    603720Kb
      Minimum:    786432Kb Maximum:      1572864Kb
   Available Pages:     66935   (  267740 Kb)
   ResAvail Pages:      93140   (  372560 Kb)
   Locked IO Pages:       247   (     988 Kb)
   Free System PTEs:   204707   (  818828 Kb)
   Free NP PTEs:        28645   (  114580 Kb)
   Free Special NP:         0   (       0 Kb)
   Modified Pages:        489   (    1956 Kb)
   Modified PF Pages:     489   (    1956 Kb)
   NonPagedPool Usage:   2694   (   10776 Kb)
   NonPagedPool Max:    33768   (  135072 Kb)
   PagedPool 0 Usage:    3622   (   14488 Kb)
   PagedPool 1 Usage:    1277   (    5108 Kb)
   PagedPool 2 Usage:    1247   (    4988 Kb)
   PagedPool Usage:      6146   (   24584 Kb)
   PagedPool Maximum:   138240    (  552960 Kb)
   Shared Commit:        5513   (   22052 Kb)
   Special Pool:            0   (       0 Kb)
   Shared Process:       3398   (   13592 Kb)
   PagedPool Commit:     6153   (   24612 Kb)
   Driver Commit:        1630   (    6520 Kb)
   Committed pages:     98489   (  393956 Kb)
   Commit limit:       320257   ( 1281028 Kb)
```

Immediately after Consume.exe triggers a round page trimming, the number of Modified Pages in RAM increases sharply:

```
*** Virtual Memory Usage ***
   Physical Memory:    130927   (  523708 Kb)
   Page File: \??\C:\pagefile.sys
      Current:    786432Kb Free Space:    472328Kb
      Minimum:    786432Kb Maximum:      1572864Kb
   Available Pages:      4920   (   19680 Kb)
   ResAvail Pages:        358   (    1432 Kb)
   Locked IO Pages:        251   (    1004 Kb)
   Free System PTEs:    204387   (  817548 Kb)
   Free NP PTEs:         28645   (  114580 Kb)
   Free Special NP:          0   (       0 Kb)
   Modified Pages:         596   (    2384 Kb)
   Modified PF Pages:      660   (    2640 Kb)
   NonPagedPool Usage:    2750   (   11000 Kb)
   NonPagedPool Max:     33768   (  135072 Kb)
   PagedPool 0 Usage:     3544   (   14176 Kb)
   PagedPool 1 Usage:     1359   (    5436 Kb)
   PagedPool 2 Usage:     1340   (    5360 Kb)
   PagedPool Usage:       6243   (   24972 Kb)
   PagedPool Maximum:   138240   (  552960 Kb)
   Shared Commit:         6842   (   27368 Kb)
   Special Pool:             0   (       0 Kb)
   Shared Process:        3688   (   14752 Kb)
   PagedPool Commit:      6398   (   25592 Kb)
   Driver Commit:         1630   (    6520 Kb)
   Committed pages:     211846   (  847384 Kb)
   Commit limit:        320257   ( 1281028 Kb)
```

A large quantity of dirty pages recently trimmed from process address spaces and added to the Modified List triggers page write operations to disk.

The Kernel Debugger can also report on virtual memory and pool usage, as discussed in greater detail in a later section titled "System Pools."

Page faults per process One side effect of a global LRU page replacement policy like the one Windows Server 2003 uses is that the memory allocation pattern of one process can influence what happens to every other process address space running on the system. You'll find it extremely useful to drill down to the process level to see which processes are consuming the most RAM and which processes are suffering the most page faults.

Process level statistics on paging activity are available in both the System Monitor and Task Manager. Both tools can display similar process level statistics that record the number of page faults each executing process incurs during each interval. However, neither tool can differentiate between hard and soft page faults per process, which limits their usefulness in this specific context. Hard page faults have the most serious performance implications because, in the time it takes to read in the page from disk, the fault stops the execution of the thread that encounters the addressing exception. Neither System Monitor nor Task Manager allows you to break out hard page fault activity per process.

Two other tools can be used to gather information on hard page faults at the process level: the Event Tracing for Windows (ETW) facility and the Page Fault Monitor tool (Pfmon.exe) available in the *Windows Server 2003 Resource Kit*. ETW distinguishes page fault events according to whether they are a *TransitionFault*, *DemandZeroFault*, *CopyOnWrite*, *GuardPageFault*, or a *HardPageFault*. To gather only hard page faults, you must use the Logman command-line interface, as shown in this example:

```
logman create trace pagefault_trace -p “Windows Kernel Trace” 0x00002003 -
o C:\Perflogs\pagefault_trace -v mmddhhmm -f BIN -rf 600 -u admin “adminpassword"

logman start trace pagefault_trace

tracerpt pagefault_trace_2003090314.etl -o pagefault_trace_2003090314.csv
```

The first Logman command creates the trace, specifying that only Hard Page Fault, Process, and Thread events are to be captured. Each Page Fault event identifies the Thread ID that encountered the page fault, the virtual address reference that caused the fault, and the value of the Program Counter showing the address of the instruction that was being executed when the page fault occurred. The second Logman command starts the trace. Finally, the Tracerpt command formats the trace file binary output into a .csv format file that you can review using Excel. There are no built-in reports that summarize the page fault trace data. Table 5-5 illustrates the trace data fields you will see from a page fault event tracing session.

Table 5-5 An ETW Hard Page Fault Trace

Event Name	**Type**	**Thread ID**	**Clock-Time**	**Kernel (ms)**	**User (ms)**	**User Data**	
Process	DCStart	0x00000924	127071168106866000	104270	1261200	0x00008BF7	0x00000AE0
Thread	DCStart	0x00000924	127071168106866000	104270	1261200	0x00000AE0	0x00000924
Event Name	Type	TID	Clock-Time	Kernel (ms)	User (ms)	Virtual Address	Program Counter
Page-Fault	HardPageFault	0x00000924	127071168364036000	104490	1261500	0x302393DE	0x302393DE
Page-Fault	HardPageFault	0x00000924	127071168364336000	104490	1261500	0x037DA074	0x30C84CEF
Page-Fault	HardPageFault	0x00000924	127071168364336000	104490	1261500	0x037D9FF4	0x30C84CEF

Using Excel, you can read each PageFault trace event record, as illustrated in the listing here that was assembled from related rows of an Excel worksheet. The Event Trace Header record is augmented here with labels for the two PageFault User Data fields. The PageFault trace events here are associated with Thread Id 0x00000924. Manually, you have to match the Thread Id from the PageFault event to an earlier Thread Event and its corresponding Process Event trace record to identify the process that encountered the page fault. Notice in this example that the second and third hard page faults occur at the same Program Counter address. In this example, it appears that a two-operand instruction encountered a page fault in translating both address operands.

If you want to focus on the page fault activity inside a single process, consider using the Page Fault Monitor utility, Pfmon.exe, that is available in the *Windows Server 2003 Resource Kit*. Pfmon captures information about every page fault that occurs for a given process. You can limit the recording to capturing only hard page faults, the results of which are illustrated in Table 5-6. Output from the following Pfmon execution is shown in Table 5-6.

```
C:\PerfLogs>pfmon /h /l /p 976
```

Table 5-6 Capturing Only Hard Page Faults

Page Fault Number	Module at Program Counter	Program Counter	Symbol	Virtual Address
0	MsoFGetTbShowKbd-Shortcuts+0x123	820481951	Ordinal961+0x00013554	825688908
1	Ordinal999+0x3b7	820479383	Ordinal961+0x00012670	825685096
2	MsoSzCopy+0x26b	816932780	00136c2c	1272876
3	MsoSzCopy+0x26b	816932780	00135a4c	1268300
4	MsoSzCopy+0x26b	816932780	00134a4c	1264204
5	MsoSzCopy+0x26b	816932780	00133a34	1260084
6	MsoSzCopy+0x26b	816932780	132854	1255508
7	MsoSzCopy+0x26b	816932780	131854	1251412
8	CoFileTimeNow+0xd68	1998196073	31f90000	838402048
9	CoFileTimeNow+0xd68	1998196073	31f91000	838406144
10	CoFileTimeNow+0xd68	1998196073	31f99000	838438912
11	CoFileTimeNow+0xd68	1998196073	31fa1000	838471680
12	CoFileTimeNow+0xd68	1998196073	31fa9000	838504448
13	CoFileTimeNow+0xd68	1998196073	31fb1000	838537216
14	CoFileTimeNow+0xd68	1998196073	32060000	839254016
15	MLPGetPrivateFile-Name+0xae21	1610935641	MLPGet-PrivateFile-Name+0x0000AE20	1610935640
16	CoFileTimeNow+0xd68	1998196073	32643000	845426688
17	CoFileTimeNow+0xd68	1998196073	32644000	845430784

The */l* option of Pfmon writes the output to Pfmon.log, a tab-delimited .txt file that you can readily view using Excel, as illustrated above in Table 5-6.

Like an ETW trace log, the information Pfmon gathers about every page fault includes the address of the instruction that was executing when the page fault occurred and the virtual address of the referenced page that caused the fault. In addition, Pfmon attempts to map these addresses into the virtual address space of the monitored process. This allows you to see for yourself which instructions and data accesses caused page faults. Similar information is available for soft faults, which reveals the virtual memory access patterns of the target program. This information can be especially useful to the programmer who developed an application program that is causing excessive paging.

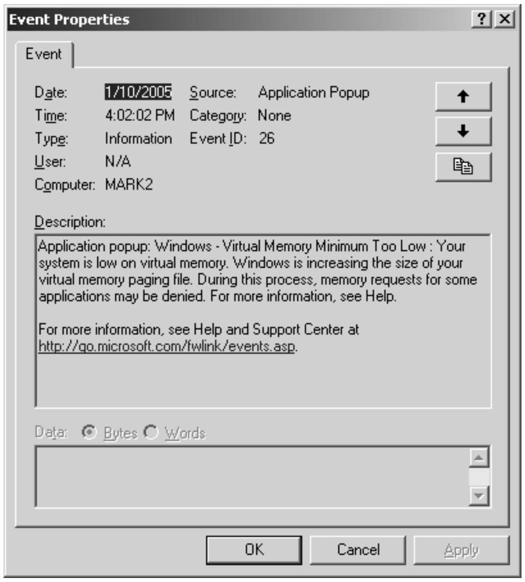
Virtual Memory Shortages

The system's Commit Limit is an upper limit on how much virtual memory can be allocated and used across all executing processes.

Note Virtual memory pages that are merely *reserved* do not count against the Commit Limit. But requests to reserve virtual memory are uncommon. Because the great majority of virtual memory allocations request committed bytes, not reserved memory, the discussion here uses the term virtual memory *allocations* to refer to committed bytes.

Virtual memory pages that are committed must be backed by either RAM or space in the paging file. Therefore, the size of RAM plus the size of the paging file represents an upper limit on the amount of virtual memory that can be allocated. When committed virtual memory begins to approach the Commit Limit, routine function calls to allocate virtual memory can fail.

If paging files are configured with flexible extents, as the virtual memory committed bytes approaches the Commit Limit, the operating system will attempt to extend a paging file and increase the Commit Limit. This provides more head room for allocating additional virtual memory. You receive notification the first time that the paging file is extended, as illustrated in Figure 5-15.



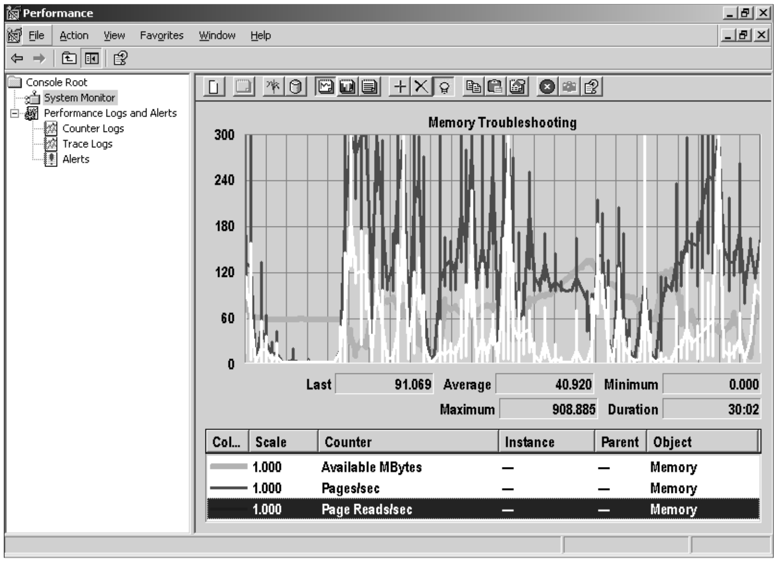
Date: 1/10/2005 Source: Application Popup
Time: 4:02:02 PM Category: None
Type: Information Event ID: 26
User: N/A
Computer: MARK2

Description:
Application popup: Windows - Virtual Memory Minimum Too Low : Your system is low on virtual memory. Windows is increasing the size of your virtual memory paging file. During this process, memory requests for some applications may be denied. For more information, see Help.

For more information, see Help and Support Center at http://go.microsoft.com/fwlink/events.asp.

Figure 5-15 Notification that the paging file is extended

Virtual memory shortages can be accompanied by performance problems because of excess paging activity. Figure 5-16 shows excessive paging activity during a period of virtual memory shortage.



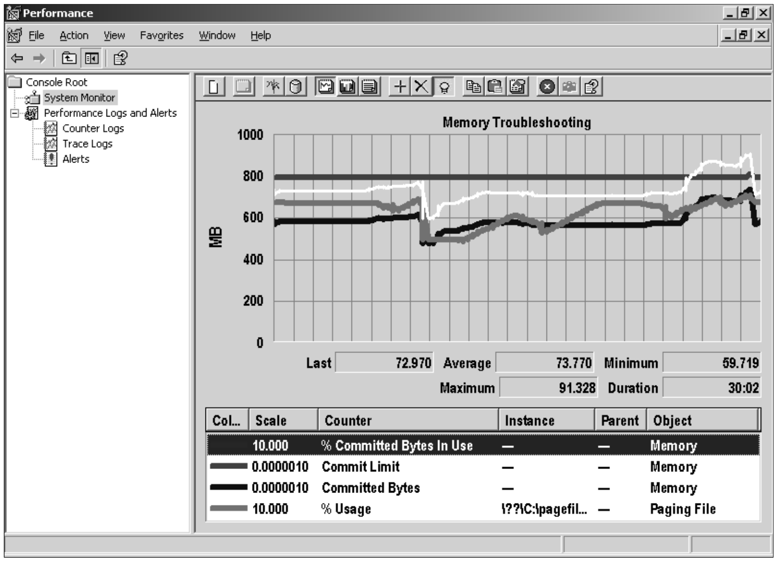
Last 91.069 Average 40.920 Minimum 0.000
Maximum 908.885 Duration 30:02

Col...	Scale	Counter	Instance	Parent	Object
	1.000	Available MBytes	—	—	Memory
	1.000	Pages/sec	—	—	Memory
	1.000	Page Reads/sec	—	—	Memory

Figure 5-16 Excessive paging activity during a period of virtual memory shortage

Figure 5-16 shows Available MBytes plunging to values near zero, triggering rounds of page trimming that are reminiscent of the virtual memory manager's behavior in Figure 5-10. The drop in the size of the Available Bytes pool is accompanied by an increase in hard page faults. Similar to the scenario shown in Figure 5-11, the hard page fault rate closely shadows the total disk paging rate.

A shortage of virtual memory is not always accompanied by the symptoms of RAM shortage leading to excessive paging, as illustrated here. If the growth in virtual memory is slow enough, the performance impact can be minimal. Virtual memory is a finite resource, however. The system cannot sustain continuous expansion of the size of virtual memory forever. When total virtual memory allocation reaches the system's Commit Limit, no more requests for virtual memory allocation can be satisfied. The most straightforward way to monitor virtual memory growth is to track the Memory\% Committed Bytes In Use counter, as shown in Figure 5-17.



Last	72.970	Average	73.770	Minimum	59.719
		Maximum	91.328	Duration	30:02

Col...	Scale	Counter	Instance	Parent	Object
■	10.000	% Committed Bytes In Use	—	—	Memory
■	0.0000010	Commit Limit	—	—	Memory
■	0.0000010	Committed Bytes	—	—	Memory
■	10.000	% Usage	\??\C:\pagefil...	—	Paging File

Figure 5-17 Monitoring virtual memory growth by tracking the Memory\% Commited Bytes In Use counter

The % Committed Bytes In Use counter is calculated from Committed Bytes and the Commit Limit, as shown here:

```
% Committed Bytes in Use = (Committed Bytes / Commit Limit) × 100
```

The counter % Committed Bytes In Use approaching 100 percent signals a virtual memory shortage.

The Chart view in Figure 5-17 also shows the two fields from which the % Committed Bytes In Use counter is calculated. Also displayed is a related measure, the Paging File(*n*)\% Usage counter, which shows how full the paging file is. At the top of the chart is the Commit Limit. To fit all four counters in the same chart, scaling factors and the y-axis have been adjusted. It should be apparent that all four measures are interrelated.

The operating system will adjust the Commit Limit upward when there is a shortage of virtual memory, as discussed in Chapter 1. This paging file extension occurs when the % Usage of a paging file reaches 90 percent. Raising the Commit Limit causes the value of the % Committed Bytes In Use counter to drop, temporarily relieving the virtual memory shortage.

Because the Commit Limit can be adjusted upward by extending the paging file (or files), also monitor the Memory\Committed Bytes and Memory\Commit Limit counters in tandem to create an accurate picture of the situation.

Situations like those shown in Figures 5-16 and 5-17, in which the demand for virtual memory suddenly surges, require determining the underlying cause. Is the increase in virtual memory allocations that are causing a shortage of RAM a consequence of normal workload growth, a one-time workload aberration, or a condition known as a *memory leak*?

Virtual Memory Leaks

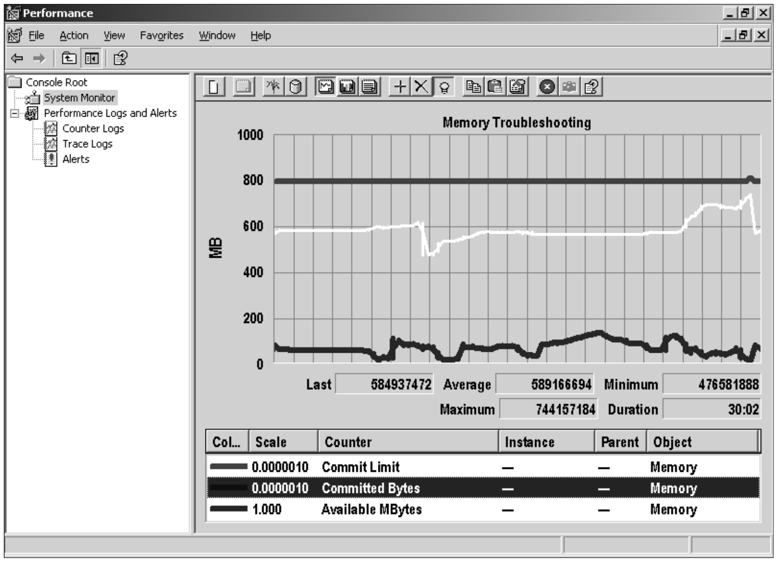
Running out of virtual addresses often happens suddenly and is the result of a program with a *memory leak*. Memory leaks are program bugs that cause a process to allocate virtual memory repeatedly but then neglect to free it when done using it. The performance impact of a memory leak varies. If the program leaking memory allocates virtual memory and forgets to free it afterwards, page trimming can frequently identify the older, idle pages of virtual memory as good candidates to be removed from RAM. As long as the page trimming process can keep up with the leaking program's demand for new virtual memory, there will be little performance impact. However, when a program leaks memory rapidly in large chunks, periods of excessive paging are likely to also accompany these acquisitions of large blocks of virtual memory.

Program bugs in which a process leaks virtual memory in large quantities over a short period of time are usually relatively easy to spot. The more sinister problems arise when a program leaks memory slowly, or only under certain circumstances. In these cases, the problem manifests itself only at erratic intervals or only after a long period of continuous execution time. This section offers some tips for diagnosing memory leaks.

Unfortunately, many programs contain memory leaks. Not all programs with memory leaks execute for a long enough time or leak memory at fast enough rates to do much damage. But those that do can cause great havoc. If a process leaking virtual memory exhausts the system's supply of virtual memory, routine calls by program functions to allocate virtual memory can fail. When virtual memory allocations fail, the results are usually catastrophic–service processes fail, the system can lock up, and so on.

Commit Limit

A virtual memory shortage occurs when the system approaches its virtual memory allocation Commit Limit. Figure 5-18 reveals rapid growth in virtual memory, triggering a paging file extension near the end of the period shown.



Last 584937472 Average 589166694 Minimum 476581888
Maximum 744157184 Duration 30:02

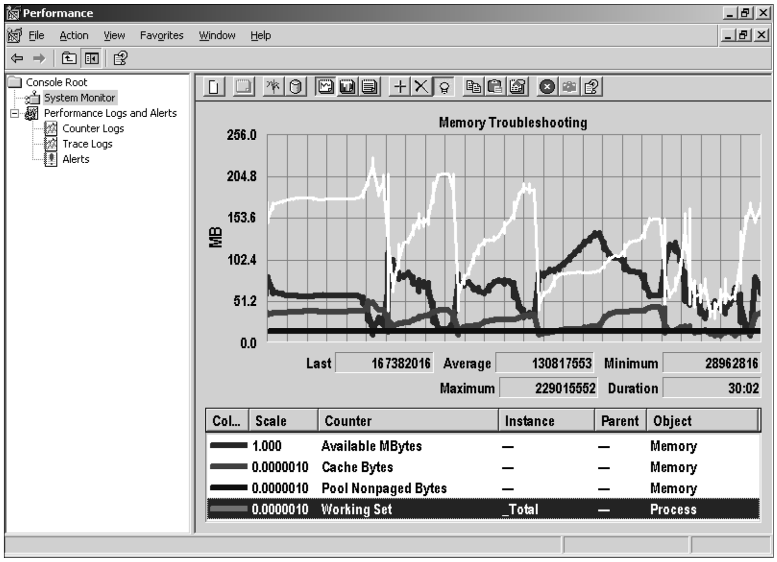
Col...	Scale	Counter	Instance	Parent	Object
▬	0.0000010	Commit Limit	—	—	Memory
▬	0.0000010	Committed Bytes	—	—	Memory
▬	1.000	Available MBytes	—	—	Memory

Figure 5-18 Rapid growth in virtual memory as the system approaches its virtual memory allocation Commit Limit

The y-axis in Figure 5-18 represents MBs, with each of the counters scaled by a factor of 0.000001 to be displayed correctly. At the outset of this period, the Commit Limit is approximately 800 MB, rising in one expansion to just above 800 MB near the end of this 30-minute period. Each incremental expansion of the Commit Limit is associated with the operating system extending the paging file. The number of Committed Bytes increases from 600 MB at the outset to a maximum value near 800 MB, at which point the paging expansion occurs. By the end of the period, Committed Bytes has retreated back to 600 MB. This expansion and contraction of Committed Bytes contradicts the classic pattern of a memory leak in which the number of Committed Bytes *never* decreases.

During this virtual memory shortage, the server experienced operational problems as new processes were unable to be launched because of a lack of resources. Some existing processes also failed and had to be restarted. Once the paging file was expanded, which raised the Commit Limit, these operational problems abated. But the performance problems associated with an over-commitment of physical memory leading to excessive paging remained.

As noted earlier, this virtual memory shortage is accompanied by excess paging. Figure 5-19 explores the dimensions of physical memory allocation for the system pools and by processes.



Last	167382016	Average	130817553
Minimum	28962816	Maximum	229015552
Duration	30:02		

Col...	Scale	Counter	Instance	Parent	Object
▬	1.000	Available MBytes	—	—	Memory
▬	0.0000010	Cache Bytes	—	—	Memory
▬	0.0000010	Pool Nonpaged Bytes	—	—	Memory
▬	0.0000010	Working Set	_Total	—	Process

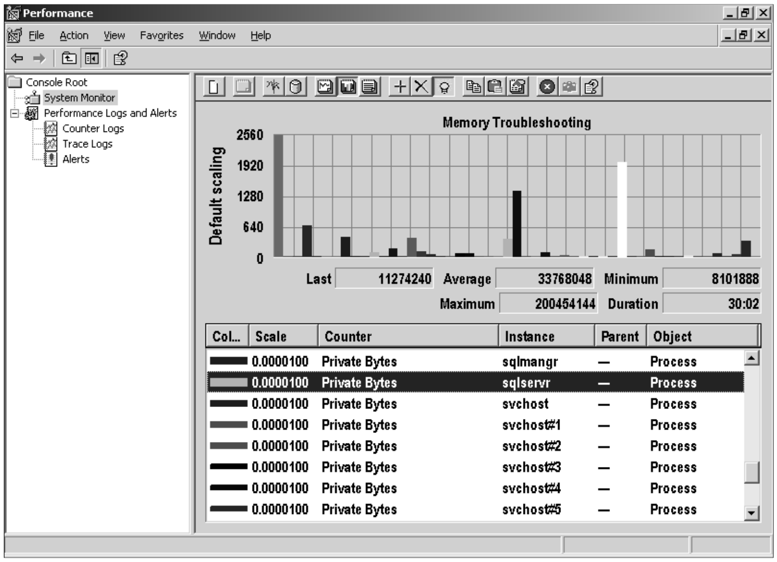
Figure 5-19 Physical memory allocated for the system pools and by processes

Figure 5-19 clearly indicates that the expansion of process working sets is responsible for the depletion of the pool of Available Bytes. At this point, it is appropriate to shift your attention to process level virtual memory allocation statistics to see what process is consuming excessive amounts of virtual memory.

Process Virtual Memory Allocations

The sheer number of processes that are active makes locating the source of a virtual memory leak tedious. If you are able to investigate the problem in real time while the condition is occurring, the Task Manager Processes display, which can be sorted by the values per column, is an effective problem-solving tool. But Task Manager can show you only what is happening in real time. To properly diagnose a memory leak, you need an historical view of the process's virtual memory usage.

The System Monitor Histogram view is one good way to search among the many processes that are running to find one that might be leaking virtual memory. Figure 5-20 shows a Histogram view of maximum values for the Process(*)\Private Bytes counter for every active process.



Last 11274240 Average 33768048 Minimum 8101888
Maximum 200454144 Duration 30:02

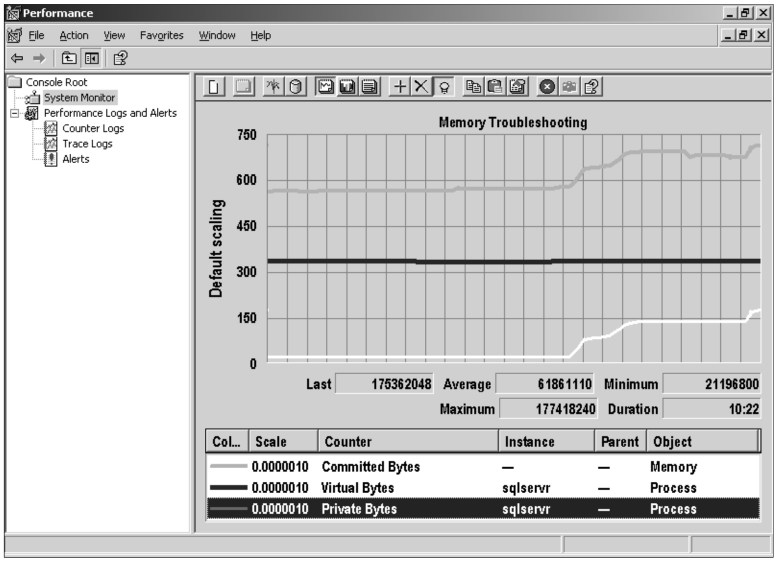
Col...	Scale	Counter	Instance	Parent	Object
▬	0.0000100	Private Bytes	sqlmangr	—	Process
▬	0.0000100	Private Bytes	sqlservr	—	Process
▬	0.0000100	Private Bytes	svchost	—	Process
▬	0.0000100	Private Bytes	svchost#1	—	Process
▬	0.0000100	Private Bytes	svchost#2	—	Process
▬	0.0000100	Private Bytes	svchost#3	—	Process
▬	0.0000100	Private Bytes	svchost#4	—	Process
▬	0.0000100	Private Bytes	svchost#5	—	Process

Figure 5-20 Histogram view of maximum values for the Process(*)\Private Bytes counter for every active process

> **Tip** A process's Private Bytes counter tracks virtual memory allocations that can be accessed by that process only. Virtual Bytes includes allocations to the system pools that can be addressed by threads running in different processes. Virtual Bytes also includes private memory that is Reserved but not Committed.

Scrolling through the process instances using the keyboard while highlighting is enabled allows you to identify processes that are consuming large amounts of virtual memory. In Figure 5-20, a process named sqlservr is identified as the largest consumer of private virtual memory, with peak usage for MS SQL Server of 200 MB on a server that contains only 256 MB of RAM.

After you identify the offending process or processes, you want to analyze the process's pattern of virtual memory usage over time. Figure 5-21 zooms in on the virtual memory usage of the sqlserver process over time.



Last	175362048	Average	61861110	Minimum	21196800
		Maximum	177418240	Duration	10:22

Col...	Scale	Counter	Instance	Parent	Object
	0.0000010	Committed Bytes	—	—	Memory
	0.0000010	Virtual Bytes	sqlservr	—	Process
	0.0000010	Private Bytes	sqlservr	—	Process

Figure 5-21 Virtual memory usage of the perf process over time

Values for both Process(sqlservr)\Private Bytes and Process(sqlservr)\Virtual Bytes are shown in Figure 5-21, alongside the value of Committed Bytes. Process(sqlservr)\Private Bytes and Committed Bytes produce virtually identically shaped lines. Notice that Process(sqlservr)\Virtual Bytes remains flat. Where Process(sqlservr)\Private Bytes increases, Committed Bytes also increases, in tandem. In intervals where Process(sqlservr)\Private Bytes drops, Committed Bytes also falls by a similar amount. It is possible to conclude that the sharp increase in Committed Bytes during the interval results from virtual memory allocation requests that come from the sqlservr process.

Still, the pattern of Process(sqlservr)\Private Bytes expanding and contracting likely does not correspond to a memory leak. A leaky application is most clearly identified as one that allocates virtual memory and never frees it. In a leaky application, Virtual Bytes allocations only increase, never decrease.

This specific problem will require more study to resolve definitively. Certain operations initiated against the SQL Server database may require more RAM than this machine has installed. The virtual memory shortage that occurred created a serious performance problem. It also caused operational problems prior to the expansion of the paging file that increased the Commit Limit. Is this a persistent problem that will recur repeatedly over the next few days and weeks? Or was it a one-time aberration that is unlikely to recur? To answer questions like these, it will be necessary to continue to review the virtual memory allocation behavior of the suspect process carefully in the days ahead.

32-Bit Virtual Memory Addressing Limits

The 32-bit addresses that can be used on IA-32-compatible Intel servers are an architectural constraint. This architectural constraint can manifest itself as a performance problem in several ways. The first performance problem results from running up against the Commit Limit, an upper limit on the total number of virtual memory pages the operating system will allocate. As discussed earlier in "Virtual Memory Shortages," this is a straightforward problem that is easy to monitor and easy to address—up to a point. Fortunately, for systems that are limited by a 32-bit virtual address space, 64-bit Windows-based systems from Intel and AMD are now available.

The second problem occurs when a User process exhausts the 2-GB range of private addresses that are available for its exclusive use. The processes that are most susceptible to running out of addressable private area are database applications that rely on memory-resident caches to reduce the amount of I/O operations they perform. Windows Server 2003 supports a boot option that allows you to specify how a 4-GB virtual address space is divided between User private area virtual addresses and shared system virtual addresses. This boot option extends the User private area to 3 GB and reduces the system range of virtual memory addresses to 1 GB. An extended discussion of this virtual memory configuration and tuning option is provided in Chapter 6, "Advanced Performance Topics."

If you specify a User private area address range that is larger than 2 GB, you must also shrink the range of system virtual addresses that are available by a corresponding amount. This can easily lead to the third type of virtual memory limitation, which is when the range of system virtual addresses that are available is exhausted. Because the system range of addresses is subdivided into several different pools, it is also possible to run out virtual addresses in one of these pools long before the full range of system virtual addresses is exhausted.

Virtual memory creep caused by slow but inexorable workload growth can also exhaust the virtual memory address range that is available. You can detect virtual memory creep by following continuous performance monitoring procedures. Detection allows you to intervene in advance to avoid otherwise catastrophic application and system failures. This section discusses these performance monitoring procedures, as well as other helpful diagnostic tools for Windows Server 2003–based machines that you should employ to detect virtual memory constraints.

System Virtual Memory

Operating system functions also consume virtual memory. The system has a working set that needs to be controlled and managed like any other process. The upper half of the 32-bit 4-GB virtual address range is earmarked for system virtual memory addresses. By setting a memory protection bit in the PTE associated with pages in the system range, the operating system assures that only privileged mode threads can allocate and reference virtual memory in the system range.

Both system code and device driver code occupy areas of the system memory region. On a large 32-bit system, running out of virtual memory in the system address range is not uncommon. The culprit could be a program that is leaking virtual memory from the Paged pool. Alternatively, it could be caused by active usage of the system address range by a multitude of important system functions—Kernel threads, TCP session data, the file cache, or many other normal functions. When the number of free System PTEs reaches zero, no function can allocate virtual memory in the system range. Unfortunately, you can even run out of virtual addressing space in the Paged or Nonpaged pools before all the System PTEs are used up.

Whenever you run out of system virtual memory addresses, either because of a memory leak or a virtual memory creep, the results are often catastrophic. When not outright catastrophic, the consequences can certainly be confusing. An application might fail during some system operation because a system allocation fails. Assuming the application can recover from this failure, the application might then issue a misleading error message.

System Pools

The system virtual memory range, 2-GB wide, is divided into three major pools: the Nonpaged pool, the Paged pool, and the file cache. When the Paged pool or the Nonpaged pool is exhausted, system functions that need to allocate virtual memory from the system pools fail. These pools can be exhausted before the system Commit Limit is reached. If the system runs out of virtual memory for the file cache, file cache performance can suffer, but the situation is not as dire.

Data structures accessed by operating system and driver functions when interrupts are disabled *must* be resident in RAM at the time they are referenced. These data structures and file I/O buffers are allocated from the nonpageable pool so that they remain in RAM. The Pool Nonpaged Bytes counter in the Memory object shows the amount of RAM currently allocated in this pool that is permanently resident in RAM.

Most system data structures are pageable—they are created in a pageable pool of storage and subject to page replacement like the virtual memory pages of any other process. The operating system maintains a *working set* of active pages in RAM for the system address space that is subject to the same LRU page replacement policy as ordinary process address spaces. The Memory\Pool Paged Bytes counter reports the amount of Paged pool virtual memory that is allocated. The Memory\Pool Paged Resident Bytes counter reports on the number of Paged pool pages that are currently resident in RAM.

The size of the three main system area virtual memory pools is based initially on the amount of RAM. Predetermined maximum sizes exist for the Nonpaged and Paged pools, but they are not guaranteed to reach their predetermined limits before the system runs out of virtual addresses. This is because a substantial chunk of system virtual memory remains in reserve to be allocated on demand—depending on which memory allocation functions requisition it first.

The operating system's initial pool sizing decisions can also be influenced by a series of settings in the HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management key, listed in Table 5-7. Both *NonpagedPoolSize* and *PagedPoolSize* can be specified explicitly in the registry. Rather than force you to partition the system area exactly, the system allows you to set either the *NonpagedPoolSize* or *PagedPoolSize* to 0xffffffff for 32-bit and 0xffffffffffffffff for 64-bit Windows-based systems. (This is equivalent to setting a -1 value.) This -1 setting instructs the operating system to allow the designated pool to grow as large as possible. The Registry Editor does not allow you to assign a negative number, so you must instead set 0xffffffff on 32-bit systems and 0xffffffffffffffff on 64-bit systems.

Note There is also a registry value named *LargeSystemCache* at HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management that influences the size of the system working set, which includes the file cache. By default, *LargeSystemCache* is set to 1, which favors the system working set over process address space working sets. If none of your server applications requires the use of the file cache, *LargeSystemCache* can be set to 0.

Table 5-7 **HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management Settings**

NonpagedPoolSize	Defaults based on the size of RAM	Can be set explicitly. A setting of -1 extends *NonpagedPoolSize* to its maximum.
PagedPoolSize	Defaults based on the size of RAM	Can be set explicitly. A setting of -1 extends PagedPoolSize to its maximum.
LargeSystemCache	Defaults to 1, which favors the system working set over other process address space working sets	Can be set to 0 when server applications do not require the system file cache. Setting *LargeSystemCache* to 1 will tune Windows memory management for file server functions. Setting it to 0 will tune for an application server.

Caution Using the /userva or /3 GB boot options that shrink the system virtual address range in favor of a larger process private address range substantially increases the risk of running out of system virtual memory. For example, the /3 GB boot option reduces the system virtual memory range to 1 GB and cuts the default size of the Nonpaged and Paged pools in half for a given size RAM. Also note that the /3 GB boot option cannot be used on systems with more than 16 GB of physical memory. For an extended discussion of these issues, see Chapter 6, "Advanced Performance Topics."

Memory\Pool Nonpaged Bytes and Memory\Pool Paged Bytes are the two performance counters that track the amount of virtual memory allocated to these two system memory pools. By using the Kernel Debugger *!vm* extension command, you can see more detail, such as the maximum allocation limits of these two pools. You can also monitor current allocation levels, as illustrated in Listing 5-8.

Listing 5-8 Results of Using the Kernel Debugger *!vm* Extension Command

```
lkd> !vm

*** Virtual Memory Usage ***
   Physical Memory:    130927   (  523708 Kb)
   Page File: \??\C:\pagefile.sys
      Current:    786432Kb Free Space:    773844Kb
      Minimum:    786432Kb Maximum:      1572864Kb
   Available Pages:     73305   (  293220 Kb)
   ResAvail Pages:      93804   (  375216 Kb)
   Locked IO Pages:       248   (     992 Kb)
   Free System PTEs:   205776   (  823104 Kb)
   Free NP PTEs:        28645   (  114580 Kb)
   Free Special NP:         0   (       0 Kb)
   Modified Pages:        462   (    1848 Kb)
   Modified PF Pages:     460   (    1840 Kb)
   NonPagedPool Usage:   2600   (   10400 Kb)
   NonPagedPool Max:    33768   (  135072 Kb)
   PagedPool 0 Usage:    2716   (   10864 Kb)
   PagedPool 1 Usage:     940   (    3760 Kb)
   PagedPool 2 Usage:     882   (    3528 Kb)
   PagedPool Usage:      4538   (   18152 Kb)
   PagedPool Maximum:   138240  (  552960 Kb)
   Shared Commit:        4392   (   17568 Kb)
   Special Pool:            0   (       0 Kb)
   Shared Process:       2834   (   11336 Kb)
   PagedPool Commit:     4540   (   18160 Kb)
   Driver Commit:        1647   (    6588 Kb)
   Committed pages:     48784   (  195136 Kb)
   Commit limit:       320257   ( 1281028 Kb)
```

The *NonPagedPool Max* and *PagedPool Maximum* rows show the values for these two virtual memory allocation limits. In this example, a *PagedPoolSize* registry value of -1 was coded to allow the Paged pool to expand to its maximum size, which turned out to be 138,240 pages (or 566,231,040 bytes), a number slightly higher than the amount of RAM installed. Because pages in the Paged pool are pageable, Performance Monitor provides another counter, Memory\Pool Paged Resident Bytes, to help you keep track of how much RAM these pages currently occupy.

System PTEs are built and used by system functions to address system virtual memory areas. When the system virtual memory range is exhausted, the number of Free System PTEs drops to zero, and no more system virtual memory of any type can be allocated. On 32-bit systems with large amounts of RAM (1–2 GB or more), it is also important to track the number of Free System PTEs.

Because the Paged pool and Nonpaged pool are dynamically resized as virtual memory in the system range is allocated, pinpointing exactly when you have exhausted one of these pools is not always easy. Fortunately, at least one server application, the File Server service, reports on Paged pool memory allocation failures when they occur. Nonzero values of the Server\Pool Paged Failures counter indicate virtual memory problems, which works even for machines not primarily intended to serve as network file servers.

Investigating Pool Usage

As noted earlier, a process could also leak memory in the system's Paged pool. The Process(*n*)\Pool Paged Bytes counter allows you to identify processes that are leaking memory in the system's Paged pool. A memory leak in the system area could be caused by a bug in a system function or by a defective device driver. Faced with a shortage of virtual memory in the system area, it might be necessary to dig into the Nonpaged and Paged pools to determine which operating system functions are allocating memory there at any point in time. This requires using debugging tools that can provide more detail than the performance monitoring counters. The Kernel Debugger can also be used in a post mortem to determine the cause of a memory leak in the Nonpaged or Paged pool that caused a crash dump.

Device driver work areas accessed during interrupt processing and active file I/O buffers require memory from the Nonpaged pool. So do a wide variety of other operating system functions closely associated with disk and network I/O operations. These include storage for active TCP session status data, which imposes a practical upper limit on the number of TCP connections the operating system can maintain. The context blocks that store the server message block (SMB) request and reply messages are also allocated from the Nonpaged pool.

Any system function called by a process allocates pageable virtual memory from the Pageable pool. For each desktop user application program, a Kernel-mode thread is created to service the application. The Kernel thread's stack—its working storage—is allocated from the Pageable pool. If the Pageable pool is out of space, system functions that attempt to allocate virtual memory from the Pageable pool will fail. When the system cannot allocate its associated Kernel thread stack, process creation fails because of a resource shortage. The Pageable pool can be exhausted long before the Commit Limit is reached.

Virtual memory allocations are directed to the Nonpaged pool primarily by device drivers and related routines. Device drivers initiate I/O requests and then service the device interrupts that occur subsequently. Interrupt service routines (ISRs) that service device interrupts run with interrupts disabled. All virtual memory locations that the ISR references when it executes must be resident in RAM. Ordinarily, a page fault occurs if a virtual memory address that is not currently resident in RAM is referenced. But because interrupts are disabled, page faults in ISR code become unhandled processor exceptions and will crash the machine. To avoid the page faults, virtual memory locations that can be referenced by an ISR must be allocated from the Nonpaged pool. Obviously, memory locations associated with I/O buffers that are sent to devices and received back from them are allocated from the Nonpaged pool.

The fact that the Nonpaged pool is of limited size creates complications. So that the range of virtual addresses reserved for the Nonpaged pool is not easily exhausted, other Kernel-mode functions not directly linked to servicing device interrupts should allocate virtual memory from the Paged pool instead. For the sake of efficiency, however, many Kernel-mode functions that interface directly with device driver routines allocate memory from the Nonpaged pool. (The alternative—incessantly copying I/O buffers back and forth between the Nonpaged pool and the Paged pool—is inefficient, complicated, and error prone.) Some of these kernel functions that can be major consumers of Nonpaged pool memory include the standard layers of the TCP/IP software stack that processes all network I/Os and interrupts. Networking applications that plug into TCP/IP sockets such as SMB protocol file and printer sharing, and Internet Web services such as the HTTP and FTP protocols, also allocate and consume Nonpaged pool memory.

Memory allocations in the system area are tagged to make debugging a crash dump easier. Operating system functions of all kinds, including device drivers that run in Kernel mode, allocate virtual memory using the *ExAllocatePoolWithTag* function call, specifying a pool type that directs the allocation to either the Paged pool or Nonpaged pool. To assist device driver developers, memory allocations from both the Nonpaged and Paged pools are tagged with a 4-byte character string identifier. Using the Kernel Debugger, you can issue the *!poolused* command extension to view Nonpaged and Paged pool allocation statistics by tag, as shown in Listing 5-9.

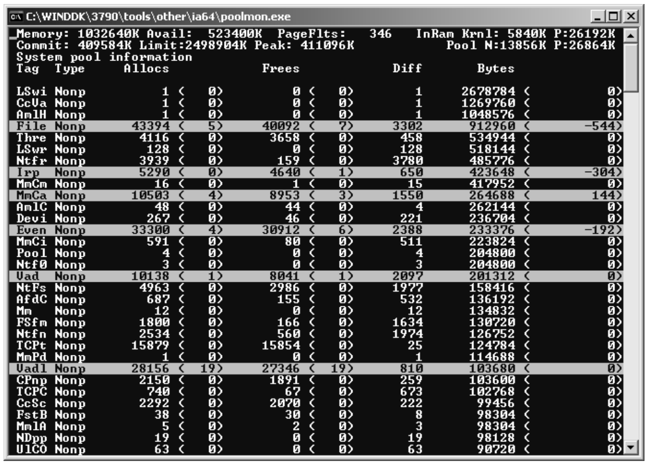
Listing 5-9 Results of Using the *!poolused* Command Extension

```
lkd> !poolused 2
   Sorting by  NonPaged Pool Consumed

  Pool Used:
            NonPaged            Paged
 Tag    Allocs     Used    Allocs     Used
 LSwi        1  2576384         0        0
 NV        287  1379120        14    55272
```

```
File     2983   504920         0         0
MmCm       16   435248         0         0
LSwr      128   406528         0         0
Devi      267   377472         0         0
Thre      452   296512         0         0
PcNw       12   278880         0         0
Irp       669   222304         0         0
Ntfr     3286   211272         0         0
Ntf0        3   196608      1770     54696
MmCa     1668   169280         0         0
Even     2416   156656         0         0
MmCi      554   127136         0         0
CcVa        1   122880         0         0
Pool        3   114688         0         0
Vad      2236   107328         0         0
Mm         12    87128         5       608
CcSc      273    85176         0         0
TCPt       19    82560         0         0
usbp       26    79248         2        96
NtFs     1872    75456      2109    122624
Ntfn     1867    75272         0         0
Io        110    72976        90      3688
NDpp       30    71432         0         0
AmlH        1    65536         0         0
MmDb      409    64200         0         0
CPnp      257    63736         0         0
Ntfi      223    60656         0         0
FSfm     1387    55480         0         0
…
```

Of course, to make sense of this output, you need a dictionary that cross-references the memory tag values, and this dictionary is contained in a file named Pooltag.txt. In the preceding code example, the *LSxx* tags refer to context blocks allocated in the Nonpaged pool by Lanman server, the File Server service; *Mmxx* tags refer to Memory Management functions; *Ntfx* tags refer to NTFS data structures; and so on, all courtesy of the Pooltag.txt documentation.

An additional diagnostic and debugging utility named Poolmon, illustrated in Figure 5-22, is available in the *Device Driver Development Kit (DDK)* that can be used in conjunction with the Pooltag.txt file to monitor Nonpaged and Paged pool allocations continuously in real time. The Poolmon utility can be used, for example, to locate the source of a leak in the Nonpaged pool caused by a faulty device driver. Figure 5-22 illustrates the output from the Poolmon utility, with the display sorted by bytes allocated in the Nonpaged pool.



```
C:\WINDDK\3790\tools\other\ia64\poolmon.exe
Memory: 1032640K Avail:  523400K  PageFlts:    346   InRam Krnl: 5840K P:26192K
Commit: 409584K Limit:2498904K Peak: 411096K            Pool N:13856K P:26864K
System pool information
Tag  Type     Allocs            Frees            Diff      Bytes

LSwi Nonp          1 (    0)         0 (    0)       1   2678784 (       0)
CcVa Nonp          1 (    0)         0 (    0)       1   1269760 (       0)
AmlH Nonp          1 (    0)         0 (    0)       1   1048576 (       0)
File Nonp      43394 (    5)     40092 (    7)    3302    912960 (    -544)
Thre Nonp       4116 (    0)      3658 (    0)     458    534944 (       0)
LSwr Nonp        128 (    0)         0 (    0)     128    518144 (       0)
Ntfr Nonp       3939 (    0)       159 (    0)    3780    485776 (       0)
Irp  Nonp       5290 (    0)      4640 (    1)     650    423648 (    -304)
MmCm Nonp         16 (    0)         1 (    0)      15    417952 (       0)
MmCa Nonp      10503 (    4)      8953 (    3)    1550    264688 (     144)
AmlC Nonp         48 (    0)        44 (    0)       4    262144 (       0)
Devi Nonp        267 (    0)        46 (    0)     221    236704 (       0)
Even Nonp      33300 (    4)     30912 (    6)    2388    233376 (    -192)
MmCi Nonp        591 (    0)        80 (    0)     511    223824 (       0)
Pool Nonp          4 (    0)         0 (    0)       4    204800 (       0)
Ntf0 Nonp          3 (    0)         0 (    0)       3    204800 (       0)
Vad  Nonp      10138 (    1)      8041 (    1)    2097    201312 (       0)
NtFs Nonp       4963 (    0)      2986 (    0)    1977    158416 (       0)
AfdC Nonp        687 (    0)       155 (    0)     532    136192 (       0)
Mm   Nonp         12 (    0)         0 (    0)      12    134832 (       0)
FSfm Nonp       1800 (    0)       166 (    0)    1634    130720 (       0)
Ntfn Nonp       2534 (    0)       560 (    0)    1974    126752 (       0)
TCPt Nonp      15879 (    0)     15854 (    0)      25    124784 (       0)
MmPd Nonp          1 (    0)         0 (    0)       1    114688 (       0)
Vadl Nonp      28156 (   19)     27346 (   19)     810    103680 (       0)
CPnp Nonp       2150 (    0)      1891 (    0)     259    103600 (       0)
TCPC Nonp        740 (    0)        67 (    0)     673    102768 (       0)
CcSc Nonp       2292 (    0)      2070 (    0)     222     99456 (       0)
FstB Nonp         38 (    0)        30 (    0)       8     98304 (       0)
MmlA Nonp          5 (    0)         2 (    0)       3     98304 (       0)
NDpp Nonp         19 (    0)         0 (    0)      19     98128 (       0)
UlCO Nonp         63 (    0)         0 (    0)      63     90720 (       0)
```

Figure 5-22 Poolmon utility from the Device Driver Development Kit (DDK)

In addition to sorting options, Poolmon has options that help you track down functions that are leaking memory in the System pools. You can monitor pool usage continuously and highlight changes in the number of bytes allocated and the number of function calls to allocate memory over time.

Disk Troubleshooting

Mechanical disks are the slowest component of most computer systems. As such, they often become performance bottlenecks. Performance counters are provided for both Logical and Physical Disks that report disk utilization, response time, throughput, and queue length. The statistics are available for each Logical and Physical Disk, and are further broken down into rates for read and write operations. The primary performance indicators that you would use to detect a disk bottleneck are shown in Table 5-8.

Table 5-8 **Disk Performance Counters to Log**

Counter	Primary Indicator	Threshold Values
Physical Disk(*n*)\% Idle Time	Disk utilization	Sustained values < 40% Idle busy should be investigated.
Physical Disk(*n*)\Avg. Disk secs/Transfer	Disk response time	Depends on the "disk," but generally response times > 15-25 milliseconds should be investigated
Physical Disk(*n*)\Current Disk Queue Length	Current number of I/O requests in process for the specified disk.	Numerous observations > 2 active requests per physical disk should be investigated. Observations > 5 active requests per physical disk are cause for alarm.

The primary indicator of disk performance is response time.

> **Important** Starting with Windows Server 2003, disk performance counters are always enabled; it is no longer necessary to enable or disable them manually using the Diskperf command-line utility. The Diskperf utility is still available on the Windows Server 2003 operating system so that computers with older operating systems can be controlled from servers running Windows Server 2003.

Disk response time can be broken down into disk service time and disk queue time, measurements that can be derived from the basic statistics the System Monitor provides. The simple formulas used to calculate these important metrics are provided in Table 5-9.

Table 5-9 Formulas for Calculating Additional Disk Measurements

Metric	Formula
Physical Disk(*n*)\% Disk Busy	100% – Physical Disk(*n*)\% Idle Time
Physical Disk(*n*)\Avg. Disk Service Time/Transfer	Physical Disk(*n*)\% Disk Busy ÷ Physical Disk(*n*)\Disk Transfers/sec
Physical Disk(*n*)\Avg. Disk Queue Time/Transfer	Physical Disk(*n*)\Avg. Disk secs/Transfer – Physical Disk(*n*)\Avg. Disk Service Time/Transfer

Different tuning strategies apply, depending on whether device service time or queue time exceeds expectations. If device service time exceeds expectations, the effective tuning strategies include:

- Using disk defragmentation tools to increase the number of sequential disk accesses and reduce seek distances for random access disk requests
- Upgrading to faster mechanical disks with improved performance characteristics
- Adding or using more effective disk caching

  If device queue time exceeds expectations, the effective tuning strategies include:

- Spreading I/O activity over more physical disks by changing file allocations
- Using array controllers to spread I/O activity over more physical disks automatically
- Using scheduling to ensure that peak workloads, such as backup and database replication, do not overlap with each other or other critical production workloads
- Improving disk service time to lower disk utilization, which indirectly lowers queuing delays

These tuning strategies are discussed in greater detail in a later section entitled "Disk Tuning Strategies."

Disk Performance Expectations

As discussed in Chapter 1, "Performance Monitoring Overview," disk service time is usually broken down into the three components associated with access to a mechanical disk: seek time, rotational delay, and data transfer. Based on the physical characteristics of the disk being accessed, you can set realistic disk service time expectations. If the measured service times of the disk you are monitoring are significantly greater than these expectations, adopt an appropriate tuning strategy to improve disk performance. The most effective strategies for improving disk performance are discussed in the section "Disk Tuning Strategies," appearing later in this chapter.

Logical Disks

A Logical Disk is a partition on one or more physical disks that is usually assigned a drive letter, such as C, and is formatted to represent a single *file system*. A file system supports the disk-resident structures that let you build folders and directories in which you can store individual files. For compatibility with older operating systems, Windows Server 2003 does support the FAT16 and FAT32 file systems. However, rarely would you use those older and less capable file systems on your machines running Windows Server 2003. Instead, you would use the NTFS file system.

NTFS is an advanced file system that provides a number of features that surpass anything available in the FAT file system. Most importantly, NTFS maintains a transaction-based recovery log that allows the operating system to recover the file system back to its last, consistent form following a system crash. With NTFS, you should never have to chase broken file pointer chains again. In addition, many Windows Server 2003–based applications like Active Directory *require* the use of NTFS. Some of the other advanced features you can take advantage of when you use NTFS include:

- A single logical disk can span multiple physical disk partitions and can be expanded on the fly so that you can avoid out-of-space failures.
- Files and folders can be compressed to save space or encrypted for security reasons.
- Permissions can be set on individual files, rather than just folders.
- Disk quotas can be used on shared volumes to monitor and control the amount of disk space used by individual users.

In some circumstances, NTFS provides performance advantages over FAT. In particular, it scales better on large physical disk drives, and it allows you more control over the file system block size that determines the size of disk I/O requests. Because it is simpler, the FAT file system is likely to be marginally faster. If circumstances render recoverability and security considerations of little consequence, you might prefer to use FAT32 for some volumes. For example, if you create a Logical Disk designed exclusively to hold a paging file, FAT32 is an acceptable choice.

Physical Disks

A physical disk is a hardware entity, or at least something that *looks* like a hardware entity, to the operating system. You can view the physical disks that are attached to your machine using the Disk Management snap-in available under Computer Management. A physical disk drive that is an actual hardware entity normally consists of a *spindle* containing one of more *platters* coated with a material capable of being magnetized, which is how digital data is stored. Data on a platter is recorded on *tracks*, which are arranged in concentric circles on the recording media. Tracks are further divided into *sectors*, usually 512 bytes wide, which are the smallest units of data that the disk can address individually. The platters on a spindle rotate continuously. Positioning arms that hold the *actuators*—the recording and playback heads—are interspersed between platters to access data tracks directly. These features of a typical physical disk drive are illustrated in Figure 5-23.

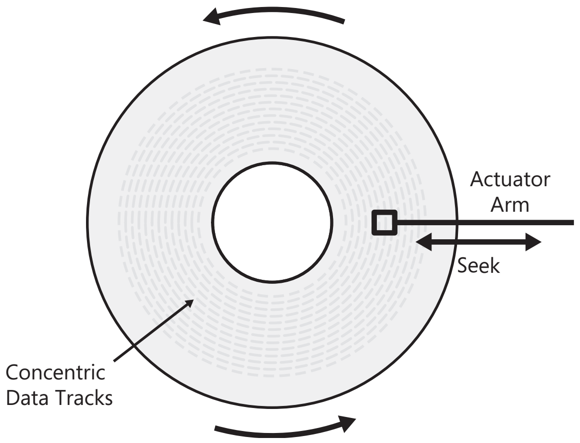


Figure 5-23 Elements of a mechanical disk drive

The time it takes to access data on a spinning disk is the sum of the following mechanical components:

```
Disk service time = seek time + rotational delay + data transfer
```

Seek time is the time it takes to reposition the read/write actuator from the current track location to the selected track location. This is sometimes called a *motion seek* because it requires a mechanical motion to move the disk arm from one spot on the disk to another. Seek time is roughly a linear function of the distance between the current track and the destination track, if you allow for some additional delay to overcome initial inertia, reach maximum speed, and brake at the end of the mechanical operation. A *minimum seek* is the time it takes to position the disk arm from one track to its neighbor. A *maximum seek* moves the arm from one end of the disk to the other. A *zero seek* refers to an operation on the current track that requires no mechanical delay to reposition the read/write arm.

Rotational delay refers to the time it takes a selected sector within the designated track to rotate under the read/write head so that the sector can be accessed. This time delay is often called the device *latency*. This delay is a function of the disk's rotation speed. If the platters spin continuously at 10,000 revolutions per minute (rpm), a complete rotation of the disk takes about 6 milliseconds, which is the maximum rotational delay. The specific sector being accessed that will begin the read or write operation can be located anywhere within the designated track, as the track spins relative to the read/write head. Consequently, an average rotational delay is one-half of a complete revolution.

Data transfer time refers to the time it takes to transfer data from the host to the disk during a write operation, or from the disk to the host during a read operation. The device's data transfer rate, normally specified in MBps, is the product of the track bit density multiplied by the track rotational speed. Whereas rotational speed is constant for all tracks on a platter, the recording density is not. Data tracks that reside on the outer surface of the platter ordinarily contain twice as many data sectors as inner tracks. As a result, data can be transferred to and from outside tracks at twice the data rate of an inner track. The size of the data block being transferred also figures into the data transfer time. For example, if the average block size is 12 KB and the track transfer rate is 60 MBps, data transfer time is approximately 0.5 ms.

Note Normally, the file system allocation unit size determines the size of I/O requests. However, the operating system will transform individual I/O requests into bulk requests under some circumstances. Demand page reads are sometimes grouped into bulk requests, as discussed in Chapter 1, "Performance Monitoring Overview." In addition, write requests subject to the lazy write file cache flushes are almost always transformed into bulk requests. Bulk requests usually increase the average size of data transfers to blocks that are 2–3 times the file system allocation unit size.

The disk manufacturer's specifications provide the basic device performance information you need to determine reasonable disk service time expectations. Table 5-10 shows speeds of a representative server disk.

Table 5-10 **Performance Characteristics of a Typical Disk Drive**

Performance Characteristic	Speed Rating
Spindle rotational speed	10,000 rpm
Average latency	2.99 ms
Seek Time	
Average seek	5.0 ms
Minimum seek (track-to-track)	0.4 ms
Data Transfer Rate	
Inner track (minimum)	43 MBps
Outer track (maximum)	78 MBps

These physical device characteristics are not the only factors that determine I/O performance, however. Additional factors that are quite important include the interface type and speed, and the use of the cache buffer on the physical disk to improve performance.

The common interface types include ATA, SCSI, and Fibre Channel. Each of these interface specifications has distinctive performance characteristics and supports a range of data transfer rates. The performance requirement of the interface is that it meet or exceed the fastest burst data rate of the disk that is connected to it. When multiple devices are configured as a *string* that shares an interface bus, channel, or link, a few active devices can normally saturate the link. This suggests configuring small strings consisting of no more than two or three disks attached to a single controller for optimal results. Any more devices sharing the link will likely create a bottleneck on the channel. This is an especially important consideration when you hang a series of disks off a single channel RAID controller card. Multiple disks that are intended to operate in parallel must serialize when they are attached to the host machine using a single link.

ATA is a simple, lightweight device protocol that is frequently supported using an embedded controller chip integrated on the motherboard. There are several popular variants of the ATA interface, including UltraATA and Serial ATA. Because it relies on a serial interface, Serial ATA (SATA) is on a trajectory to deliver 150, 300, and then 600 MBps transfer rates. Another benefit of some SATA disks is the ability to specify operations that Write-Through directly to the disk, bypassing any on-board disk cache. This is an important data integrity requirement for any disks that you use with critical server workloads, which makes less expensive SATA disks suitable for many environments.

SCSI is a venerable protocol that utilizes parallel bus technology. The UltraSCSI flavor of the protocol can obtain instantaneous data transfer speeds of 320 MBps. The effective throughput of a SCSI connection is far less, however, because of protocol overhead. A time-consuming bus arbitration sequence is required to establish a connection between the device and its controller (or *initiator*, to use SCSI terminology). SCSI bus arbitration can take 0.5 ms to establish a SCSI connection to process read or write commands. Protocol overhead reduces the *effective* bandwidth of a SCSI connection to only about 50 percent of its rated (theoretical) bandwidth. When more than one device on a string tries to gain access to the SCSI bus, a fixed priority scheme, based on the SCSI address (or *target*), is used to break the tie. The notoriously “unfair” scheduling policy SCSI uses can cause serious performance anomalies when heavily used disks share a single SCSI bus.

Fibre Channel is a serial protocol that has emerged to dominate the world of high performance storage devices, including the Storage Area Networks (SANs) used to interconnect server farms to disk farms. Fibre Channel links are available in 100, 200, and 400 MBps flavors. With the ability to intersperse packets from multiple active transfers, the Fibre Channel protocol can utilize 80 percent or more of the effective capacity of its link. While maintaining upward compatibility with the SCSI command set, Fibre Channel eliminates the time-consuming handshaking protocol associated with SCSI and SCSI’s priority scheme. Fibre Channel also provides more flexible cabling and addressing options than SCSI.

For a system with one or two disks, the interface type and speed rarely has much of a performance impact. That is because the bandwidth of the interface is normally several times the maximum data rate at which a single disk is capable of transferring data. As the number of disks attached to the host increases—especially for RAID disk controllers—the interface speed gains importance as a performance consideration.

Almost all commercial disk drives in widespread use are single servers—they can respond to only one I/O request at a time. Requests sent to a device that is already busy servicing a request are queued. Many devices and controllers have the ability to select among queued disk requests to enable them to process shorter requests first. Sorting queued requests in this fashion is “unfair,” but serves to improve disk service time under load. When the disk request queue is sorted to perform requests that can be serviced faster first, disks act as load-dependent servers, as discussed in Chapter 1, “Performance Monitoring Overview.”

Most performance-oriented disks incorporate RAM that is configured as a *multisegmented track buffer*. After tracks of data are stored in the cache, subsequent requests to access data in cache can be satisfied without returning to the disk. On a read cache hit, for example, there is no seek or rotational delay, only data transfer time. Moreover, data can be transferred to and from the cache at full interface speed, usually significantly faster than data can be transferred to and from the disk. This caching is so effective that disks typically can process 10 or 20 times more sequential I/O requests than random I/Os. The *multisegmented* aspect of the on-board disk cache refers to cache management algorithms, which can do read-ahead, delete-behind processing on behalf of several sequential processes simultaneously. There is usually some implementation-dependent limit on the number of independent sequential processes that the disk cache firmware can recognize before performance begins to drop back toward the level of random disk I/O. The disk benchmark example discussed later highlights the benefit of on-board disk cache for sequential disk processing.

Tip The dramatic improvement in disk performance that on-board caches provide favors any tuning strategy, such as disk defragmentation, that serves to increase the amount of sequential disk processing that your machines perform.

The disk's built-in multisegmented cache buffer is the performance-oriented feature that has the greatest impact on disk performance. When the disk workload can be effectively cached, disk I/O throughput can increase several fold. Empirically, this cache effect creates a wide range of performance levels that you are able to observe when you monitor the Logical or Physical Disk statistics, reflecting a workload mix that can shift at almost any time from sequential to random processing. Because no statistics are available on disk cache hit rates, you cannot measure the disk cache's effectiveness directly. When the majority of I/O requests can be satisfied from the built-in disk cache, significantly higher I/O rates and correspondingly lower disk service time are reported. Assuming that

Disk service time = seek + rotational delay + data transfer time + protocol time

for the representative disk described in Table 5-10

Disk service time = 5.0 + 3.0 + 0.2 + 0.1 = 8.3 ms

when it is performing a conventional read or write operation. However, when the operation is a cache hit,

Disk service time = 0 + 0 + 0.1 + 0.1 = 0.2 ms

a potential 40-fold improvement.

Multipathing Multipathing I/O solutions are available for many SCSI and Fibre Channel interface adapters and disks. Multipathing provides for high availability data access by allowing a host to have up to 32 paths to an external disk device. Additional paths are configured for redundancy, automatic failover, and load balancing. Multipathing is not a feature of the operating system, but is supported through the MPIO Driver Development Kit (DDK), which allows storage vendors to develop multipathing solutions designed to work with their equipment. Check with your hardware vendor to see whether multipathing I/O support is available for your storage configuration.

Characterizing the Disk I/O Workload

Physical disk characteristics establish a baseline set of performance expectations. The disk I/O workload characteristics that influence disk performance in your specific environment are described in Table 5-11.

Table 5-11 Disk I/O Workload Characteristics

Counter	Primary Indicator	Notes
Physical Disk(*n*)\Disk Reads/sec	Disk reads	Heightened data integrity concerns mean that physical disk reads are often executed slightly faster than disk writes.
Physical Disk(*n*)\Disk Writes/sec	Disk writes	Caching for disk writes should be enabled only when the disk controller has a dependable redundant power scheme.
Physical Disk(*n*)\Avg. Disk Bytes/Read	Average block size of read requests	Normally, a function of the file system allocation unit. Sequential prefetching increases the size of the average read request.
Physical Disk(*n*)\Avg. Disk Bytes/Write	Average block size of Write requests	Deferred writes because of lazy write cache buffering are usually performed in bulk, increasing the size of the average write request.

No measurements are available to report on the percentage of sequential vs. random disk requests. Instead, the rate of sequential disk requests, which are cached very effectively at the disk level, must be inferred indirectly from improved disk service time measurements at higher I/O rates.

Establishing a Disk Drive Performance Baseline

Because disk drive performance is a function of the hardware and your specific workload characteristics, it is useful to establish a disk performance baseline independent of your workload. The technical specifications for most hard disk drives are available at their manufacturers' Web sites. Average seek time, drive rotation speed, and minimum and maximum data transfer rates are commonly reported. These specifications are a good start, but they do not address the performance of the built-in disk cache, which is a very important component. To understand how the disk cache works, you probably need to measure your disk in a controlled benchmark environment. Doing this is simpler than it sounds. You can determine the capabilities of your disk equipment using Performance Monitor and almost any decent I/O stress-testing program.

Table 5-12 shows a minimal set of disk access specifications that can be used to characterize disk performance for most devices. These access specifications represent a thorough workout for your disk hardware and will allow you to determine the effective limits on the disk's operating characteristics.

Table 5-12 Benchmark Specifications to Characterize Physical Disk Performance

Specification	Block Size	Read/Write	Random/Sequential
4-KB random read	4096	100% read	Random
4-KB random write	4096	100% write	Random
4-KB sequential read	4096	100% read	Sequential
4-KB sequential write	4096	100% write	Sequential
32-KB random read	32768	100% read	Random
32-KB random write	32768	100% write	Random
32-KB sequential read	32768	100% read	Sequential
32-KB sequential write	32768	100% write	Sequential

Comparing and contrasting small 4-KB blocks to larger 32-KB blocks should allow you to extrapolate performance levels across a range of block sizes. Focusing on 100 percent read vs. write workloads allows you to isolate any differences in the ways reads and writes are handled. The sequential workload will allow you to observe the performance of the on-board disk buffer. Cache will have little or no impact on the random I/O workload—this workload will allow you to measure the performance capabilities of the native disk hardware by minimizing any cache effects.

Figure 5-24 shows the results for the 4-KB series of benchmark runs against a simple, one-disk configuration.

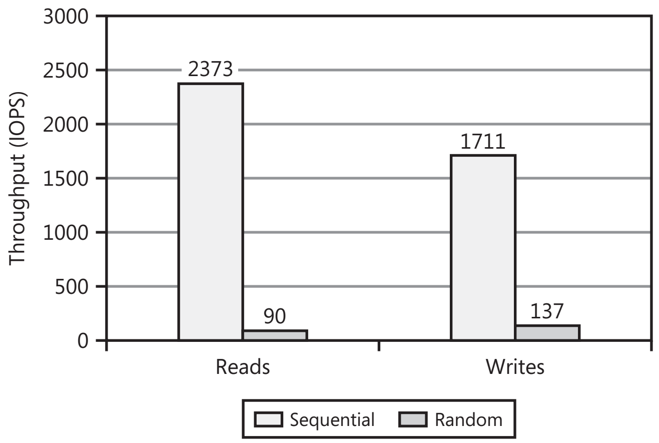


Figure 5-24 Sequential vs. random I/O throughput for a single disk system

The results in Figure 5-24 reveal the performance impact of the disk cache. When sequential reads can be satisfied from the on-board disk cache in this example, the physical disk can sustain I/O rates that are 25 times higher, compared to that of random requests. This disk evidently utilizes its cache to buffer writes, deferring its update of the physical media until an entire track's worth of data can be destaged from the cache to the disk in a single burst. With sequential writes, caching improves throughput and service time by a factor of at least 10, compared to random I/O processing. Random writes apparently also benefit some from the disk cache because throughput rates were 40 percent higher for random writes compared to random reads.

Figure 5-25 shows the corresponding service times for the simple, single-disk system being tested. The average service time of a sequential disk read request was reported as 0.4 milliseconds, compared to an average of 11.1 ms for a random read request. Write operations took slightly longer. Applying the Utilization Law, you can multiply the disk service time by the I/O rate and verify that the disk is 100 percent utilized during the test.

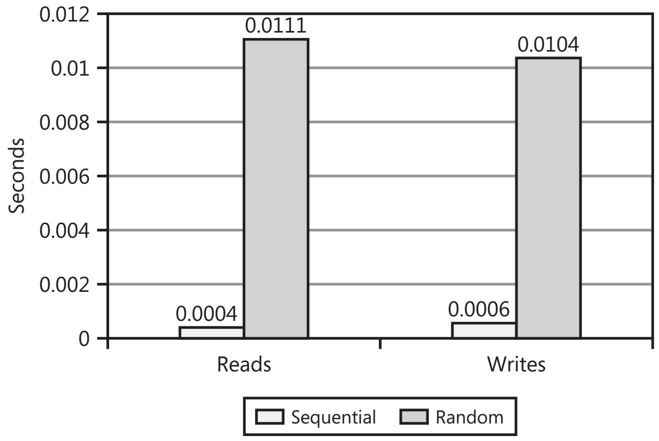


Figure 5-25 Sequential vs. random I/O service time for a single disk system

These empirical results should reinforce the argument that on-board disk caches are highly effective for sequential processing. Almost anything that you can do that will increase the amount of sequential disk I/O performed to a single disk—including disk defragmentation—will improve disk service time. Optimizations built into the system file cache that perform sequential prefetching on user files, and the aggressive anticipatory paging performed by the operating system, also increase the amount of sequential I/O processing.

Figure 5-26 shows the throughput results when large 32-KB blocks are used instead. Throughput of random writes is slightly higher than for reads, indicating that write caching remains a benefit. The performance of large block sequential reads and writes is quite similar, perhaps indicating that some component of the link between the host computer and the disk has itself reached saturation.

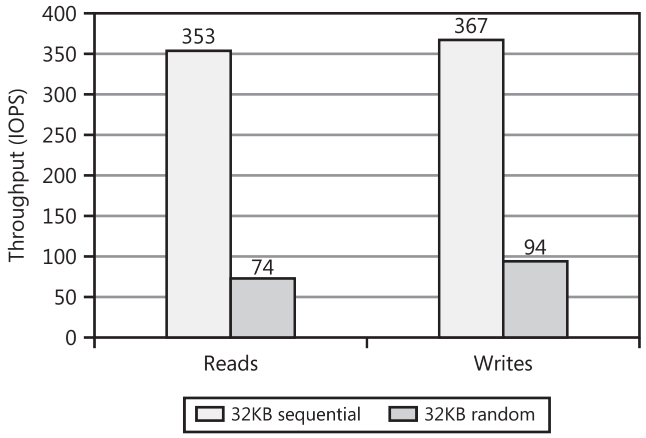


Figure 5-26 Sequential vs. random I/O throughput for a single disk system using 32-KB blocks

A final exercise is to compare service times for small 4-KB blocks to large 32-KB blocks. This comparison is shown for sequential reads and writes in Figure 5-27.

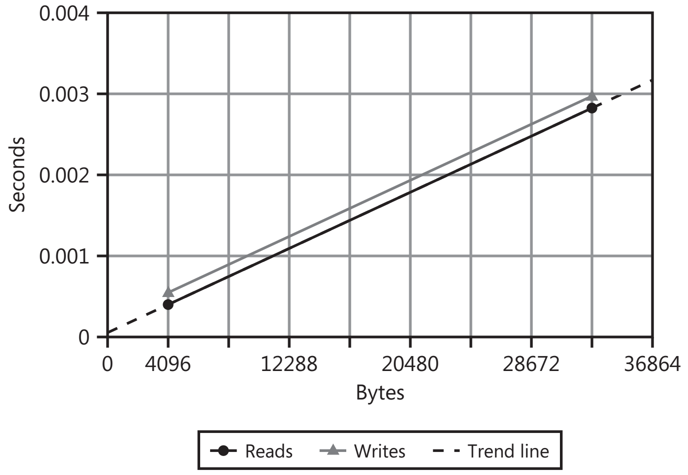


Figure 5-27 Data transfer rate as a linear function of block size

Because sequential reads are subject to near 100 percent cache hits, the service time of cache hits consists almost entirely of data transfer time. The data transfer time is roughly a linear function of the request block size. The trend line drawn between the 4 KB and 32 KB points that were measured is an approximation of this linear function, assuming 32 KB performance is not constrained by some other factor, such as the link speed. The fact that the trend line drawn between the two measured write points runs parallel to the line drawn between the two read points is also convincing evidence that corroborates this interpretation of the results.

Applying the benchmark results to the real world The benchmark results reported here all use uniform workload distributions. Each separate test is either 100 percent read or 100 percent write, 100 percent sequential or 100 percent random, and uses a constant block size. This is the point of this synthetic benchmark—to exercise control over the workload so that the measurement results can be interpreted unambiguously. In the real world, the I/O workload is a mixture of these elements—there is a mix of random I/O requests and sequential I/O requests, for example. The block size also varies depending on whether individual or bulk requests to the disk are issued. Your actual real-world I/O workloads are an amalgamation of these synthetic ones.

The benchmark tests reveal that the single most important factor influencing disk service time is whether the I/O request is sequential or random. The service time of sequential requests is 10–20 times faster than random disk I/O requests. Your real disk I/O workloads contain a mix of sequential and random I/O requests. When the mix is richer with sequential requests, the average disk service time is faster. When a higher proportion of random requests is issued, the average disk service time will be slower. The capacity of your disks to perform I/O operations will vary substantially during any measurement interval, depending on the mix of sequential and random I/O processing. By using short measurement intervals, bursts of sequential activity are often apparent, showing very high I/O rates and very low service times. With longer measurement intervals—1–5 minutes or more—you will usually see a smoothing effect so that the differences between one period and the next are less pronounced.

The available disk performance counters cannot be used to understand this crucial aspect of the disk I/O workload. If you do need to characterize your I/O workload based on the proportion of sequential and random I/O requests, you can gather Disk I/O trace data that will show the sequence of each individual disk request and how long the request took to execute. An example that illustrates the Disk I/O trace data is discussed in the section entitled "Understanding File-Level Access Patterns."

Storage Controllers and Virtual Disks

Disk hardware performance, which is something that you can easily assess for yourself by following the stress testing methodology described earlier, is not the only crucial factor in disk I/O performance. That is because disk operations are often not directed against simple physical disks. An entity that appears to the operating system to behave like a physical disk is often a collection of software and hardware within a storage controller designed to impersonate the characteristics of a physical disk. For want of a more descriptive term, these logical entities are often called *virtual disks*, whereas the software that exposes physical disk images to the operating system and performs the I/O processing that maintains this illusion is often called a *virtualization engine*. Virtual volume management software, usually embedded within a storage controller, maps virtual volume physical disk images into a set of disk sectors arrayed across one or more physical disks.

The components of a typical storage controller that performs virtual volume mapping and other storage functions are depicted in Figure 5-28.

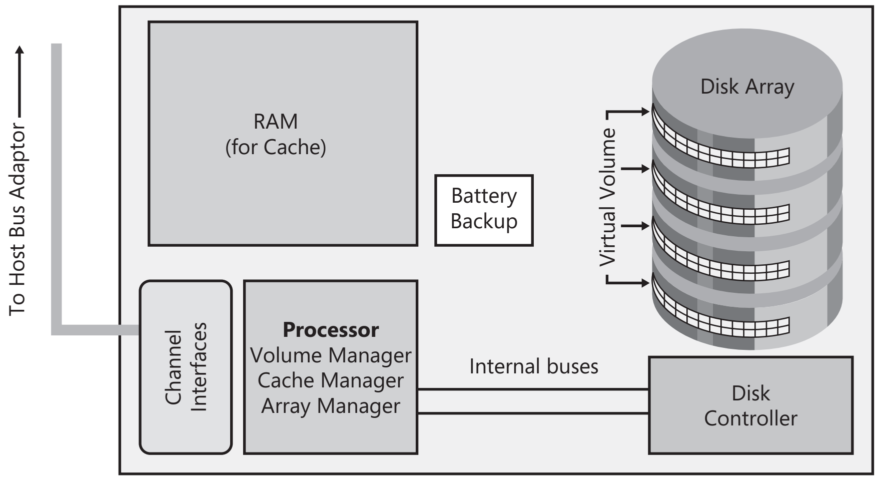


Figure 5-28 The elements of a typical storage controller that performs virtual volume mapping

These components include:

- One or more processors that run volume manager, cache management, and disk array management software
- Internal RAM for working storage and disk caching
- One or more front-end channel interfaces that attach to host buses
- One or more back-end disk controllers that attach to a series of disk arrays
- Internal bus connections that tie these hardware components together into an integrated system
- Redundant power supplies, including a provision for battery backup to maintain power to critical components when the AC power supplies are cut

The fact that storage controllers export one or more virtual disks that look like physical disk entities to the operating system complicates the analysis of disk bottlenecks. Under load, a disk subsystem might reveal an internal bottleneck that might not be apparent externally. Another complication is that multiple virtual volumes could be mapped to physical disks in ways that create hot spots of disk contention. Even though the external signs of this disk congestion are readily apparent, determining why Physical Disk volume performance is degraded might require you to investigate the internal workings of these disk subsystems. You might need to pool all your host-based measurements of virtual volumes and correlate them with performance statistics on the disk subsystem's internal workings.

Some common performance considerations for storage controllers include:

- The organization of physical disks into arrays of redundant disks, or RAID
- The mapping of virtual volumes—perceived as physical disks by an attached host computer—onto disk arrays
- The use of cache to improve disk performance and mask the performance penalties associated with common forms of RAID disk organization

There is a great variety of storage controller architectures—too many to discuss here. Not every consideration discussed here will apply to your disk hardware configuration. These architectural considerations greatly complicate the task of establishing an accurate performance baseline for virtual devices that are managed by one of these storage controllers. An unbiased source of comparative information on disk storage controllers is available from the Storage Performance Council (SPC), the sponsor of the SPC benchmarks that many of the leading disk storage vendors participate in. The SPC-1 benchmark uses a synthetic I/O workload that is designed to be similar to disk I/O activity patterns evident on mail and messaging sy*stems like Microsoft Exchange Server. The audited SPC benchmark results posted at http://www.storageperformance.org/results* provide disk I/O subsystem response time measurements, and disclose the full costs of the hardware configuration tested. This Web site is an excellent source of unbiased, comparative cost/performance data for popular storage controller products from major storage vendors.

RAID Redundant array of independent disks, or RAID, is the organization of independent disks into arrays of disks that store both primary data and redundant data on separate disk spindles. Storing redundant data on a separate disk allows the primary data to be reconstructed in case of a catastrophic failure of one of the disks. RAID technology organizes sectors across several physical disks into a logical volume that can—in theory, at least—be accessed in parallel during a single I/O operation. To perform disk access in parallel, array controllers need dual front-end and back-end data paths, dual internal buses, and dual processors, not just multiple disks. Disk arrays also store data redundantly on disks separate from the data disks so that it is possible to recover from a physical disk failure. Different schemes to accomplish these ends are known as RAID *levels*, based on terminology first coined in the 1980s by researchers at the University of California at Berkeley. Additional disk drives are required to store the redundant data. The different RAID levels represent different ways to organize the primary data that corresponds to virtual volumes and the redundant data that provides protection across multiple physical disks.

Important Contrary to popular belief, RAID technology is not intended to be a performance optimization. Redundant disk arrays are designed to provide high availability and fault tolerance. The performance of RAID configurations is usually worse than simple JBOD (Just a Bunch Of Disks) disk connections, although some RAID levels benefit from automatic load balancing across multiple disks.

RAID disk subsystems require that additional I/O operations be performed to maintain redundant copies of the data stored there. You need to understand the performance trade-offs inherent in the various types of RAID disks. The most common RAID levels and their relative benefits and trade-offs will be discussed briefly.

RAID 0 RAID 0 is also known as *disk striping*. RAID 0 organizes multiple disks (or sections from multiple disks) into a single logical volume that can be accessed in parallel. Strictly speaking, RAID 0 is not a RAID level at all because there is no provision for storing redundant data, which makes disk striping especially vulnerable to disk failures. The advantages of accessing multiple disks in parallel are muted in Windows Server 2003 because the native file systems do not support large enough allocation units to make striping an effective optimization. Block sizes of 32 KB or less rarely benefit from disk striping. One noteworthy benefit of disk striping is that the I/O load tends to be balanced automatically across all disk volumes in the stripe set, something that is difficult to achieve by placing files manually. Queuing delays are minimized when the disk load is spread evenly across multiple devices, so this is a potentially significant performance benefit.

RAID 1 RAID 1 is also known as *disk mirroring*. In RAID 1, data is always written to two separate disks, one of the disks being a mirror image of the other. Strictly speaking, there are data mirroring schemes that do not require that each secondary be an exact duplicate of a primary disk. The only requirement for RAID 1 is that the data exist on some other disk and that a primary volume can be reconstituted from data stored elsewhere so that, in the case of a disk failure, there is always a full backup copy of the data available for immediate access. As a data redundancy scheme, RAID 1 provides little or no performance advantage and some performance penalty. Some controller implementations race the primary and secondary volumes during a read operation. Because independent disks spin independently, racing the disks permits the disk with less rotational delay to complete the operation first. The result is a better-than-average rotational delay. The performance penalty associated with RAID 1 is the duplication of all writes. This duplicate write activity is transparent to the host computer, but it does lead to increased load on physical disks, controllers, and internal buses. From a disk capacity standpoint, RAID 1 requires configuring twice as much physical disk space as conventional disks, which is also an added expense.

RAID 0/1 Combining *disk mirroring* and *disk striping*, RAID 0/1 requires at least four physical disks. This type of RAID is also frequently called RAID 10. Because it combines the advantages of disk striping with redundant data recording, RAID 0/1 is often the choice recommended by performance experts. With the relatively small file system allocation units available in Windows Server 2003, the performance advantage of this RAID scheme is limited to the load balancing across physical disks that striping affords. Nevertheless, in an I/O-intensive database environment, this automatic hardware load balancing is a major advantage because balancing the disk load might be very difficult to accomplish manually. RAID 0/1 does share with disk mirroring the performance penalty of duplicate writes. Like RAID 1, it also requires configuring twice as much physical disk space as conventional disks.

RAID 5 RAID 5 uses *rotated parity sectors*, instead of mirroring. RAID 5 saves space by storing a parity sector, instead of duplicating all the original data. The redundant parity data is generated for each set of associated data sectors and then stored on a disk separate from the data disks. Instead of doubling the amount of disks needed for mirrored copies, RAID 5 adds the extra capacity of only a single parity disk to the array storage requirements. One drawback, compared to mirroring, is performance during reconstruction, which is poor. To reconstruct the original data following a disk failure, all the remaining data disks plus the parity disk must be read. However, because disk failures should be rare events, the performance of RAID 5 during reconstruction is only a minor consideration. Compared to RAID 4, RAID 5 rotates the parity sector among all the disks in the array stripe set to avoid having a single parity disk itself become a hot spot. The effect is that each disk in a RAID 5 array contains mostly data, but also some portion of parity data. On the plus side, RAID 5 does tend to balance disk activity across all the disks in the array.

The most serious drawback to RAID 5 is a severe write performance penalty. On an update, both the data sector and its associated parity sector need to be updated. The most economical way to accomplish this is to read the old parity sector, read the old data sector, "subtract" the old data from the old parity, write the new data sector, recompute the parity sector, and, finally, update the parity sector. This is also known as a *Read-Modify-Update* sequence. With RAID 5, a single write operation requires four I/Os to the array to maintain the parity information. This serious write performance penalty might make RAID 5 unsuitable whenever the write activity is excessive. Writeintensive workloads run the risk of overloading the disks in the array and the internal paths that connect them to the controller.

A number of approaches can reduce the impact of the RAID 5 write performance penalty, but no known method can eliminate it entirely. One effective technique is to perform the two read operations in parallel, followed by the two write operations in parallel. This cuts the overall duration of the delay in half, which is significant, but it does not affect the need to perform the additional I/O operations. It also requires that disks configured as part of the same RAID set be accessible on separate paths so that it is physically possible to perform concurrent disk operations. Other optimizations rely on the use of battery-backed cache memory in the controller to mask the latency associated with RAID 5 write requests.

Tip The Volume Management software included with Windows Server 2003 allows you to implement RAID 0, RAID 1, and RAID 5 disk striping, mirroring, and redundancy options without resorting to special RAID controller cards. For more information, see the "Disk Management" topic in the "Help and Support Center" documentation.

Comparing RAID levels Choosing among the different RAID volume mapping techniques requires understanding the trade-offs between cost, disk capacity, potential performance advantages and disadvantages, and recovery time when a disk in the array fails.

The four most common RAID volume-mapping schemes are pictured in Figure 5-29, emphasizing the additional disk capacity required to implement data redundancy.

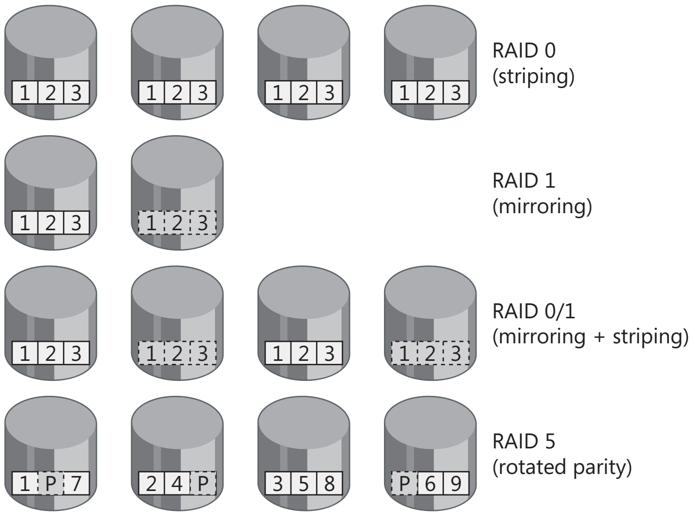


Figure 5-29 The four most common RAID volume-mapping schemes

Simple disk striping requires no additional disk capacity because no additional redundant data is recorded. Mirrored disks require double the amount of disk capacity because all data is duplicated on separate disks. Mirrored disk schemes can also incorporate disk striping (RAID 0/1), as shown. RAID 5 saves on storage requirements because it requires only one additional parity disk per array. In a 4-disk array, like the one illustrated in Figure 5-29, a RAID 5 configuration can store 50 percent more primary data than a 4-disk mirrored configuration.

Disk striping using either RAID 0 or RAID 0/1 can speed up disk operations by breaking very large block requests into smaller requests that can be processed in parallel across multiple disks. In practice, performance improvement can be achieved only when blocks significantly larger than the allocation units supported by the Windows NTFS file system are used. Note that large block requests can occur frequently in Windows Server 2003 bulk paging operations. If the cost of the extra physical disks in RAID 0/1 striping and mirroring is not prohibitive, this alternative provides the best overall performance of any of the redundancy schemes.

Disk striping and RAID 5 spread I/O activity automatically across multiple disks. This balances the I/O load across multiple disks more effectively than any manual file placement effort that you can perform, so this is an assured performance benefit. This benefit should be weighed against the performance penalty associated with write operations to both mirrored disks and RAID 5 arrays. With mirrored disks, each write operation leads to two physical writes to separate disks. With RAID 5, the write penalty is even more extreme. Each write operation in RAID 5 leads to two physical reads followed by two physical writes to separate data and parity disks.

RAID 0 disk striping, without any data redundancy, affords no data protection. Disk striping actually increases the vulnerability of a volume to a disk failure because an error on any disk in the array is a failure that impacts the integrity of the volume. Recovery from a failure when mirrored disks are used is straightforward because the data appears on two separate disks. If either disk fails, its duplicate can be substituted. In RAID 5, the original data has to be reconstituted by reading the parity disk and the remaining data disks. Until the failed volume is fully reconstituted to a spare disk, the damaged array is forced to run at a degraded performance level.

This discussion of the relative benefits and trade-offs of the various RAID level volume mapping schemes is summarized in Table 5-13.

Table 5-13 Cost/Benefits Analysis of RAID Volume Mapping Schemes

RAID Level	Performance Advantages/Trade-Offs
RAID 0	■ Availability: No redundant data protection. ■ Additional disk capacity: None. ■ Performance benefit: Automatic load balancing across multiple disks; very large blocks might benefit from parallel operations . ■ Performance penalty: None.
RAID 1	■ Availability: Duplicate data protects against hardware failure. ■ Additional disk capacity: 2 x the number of data disks. ■ Performance benefit: Might benefit from racing the disks. ■ Performance penalty: 2 x the number of writes.
RAID 0/1	■ Availability: Duplicate data protects against hardware failure. ■ Additional disk capacity: 2 x the number of data disks. ■ Performance benefit: Automatic load balancing across multiple disks; very large blocks might benefit from parallel operations. ■ Performance penalty: 2 x the number of writes.
RAID 5	■ Availability: Data stored on a failed disk can be reconstituted by reading the parity disk and the remaining data disks. ■ Additional disk capacity: +1 parity disk per array. ■ Performance benefit: Automatic load balancing across multiple disks. ■ Performance penalty: Each write request requires 4 physical disk operations (2 reads + 2 writes).

Important When monitoring disk drives in a RAID volume, it is important to understand how the performance counters are collected and displayed for the different types of RAID systems.

Software RAID volumes, also know as *dynamic volumes*, will show one drive letter in the Logical Disk instance list for each volume that is assigned a drive letter. Each physical disk in that set will be listed separately under the Physical Disk instance list.

Hardware RAID disks are listed as a single disk, regardless of how many physical drives make up the RAID set. With hardware RAID, the instance list for the Logical Disk object shows one drive letter per volume configured on that set, or possibly even no listing if the drive is used as a raw disk drive. In the instance list for the Physical Disk object, one physical drive will be listed for the entire RAID set, regardless of how many drives make up the set. The RAID controller hardware operates its disk drives independently from the disk device drivers in the operating system. To the disk drivers in the operating system, the physical disk drives behind the RAID controller are invisible. Whatever virtual physical disk images the RAID controller software exports appear as Physical Disk drives to the operating system.

Caching controllers Cache memory is a popular performance option available on many disk controllers. Its use is heavily promoted by major disk subsystem manufacturers, but its role in improving disk performance is often misunderstood. The main performance-related function of controller resident disk caches is to limit the impact of the write performance penalty associated with the redundant data recording scheme that is employed to provide data protection, as discussed earlier. One of the most important performance-oriented functions of disk controller cache for machines running Windows Server 2003 is to help mask the severe performance penalty associated with RAID 5 writes. Properly configured, disk controller cache allows RAID-5 disk arrays to achieve performance levels comparable to other RAID levels.

Cache inside the controller is less significant on read operations. Cache memory inside the controller is sitting in the data path between disks with on-board cache memory and the potent, host memory-resident caches associated with most server applications. A careful analysis of the data normally resident in the disk controller cache shows it largely duplicates data that is already cached in host memory or in the on-board disk cache. This renders large amounts of disk controller-resident cache memory largely redundant for the purpose of caching reads.

Disk controller cache is effective in masking disk hardware performance problems, especially disk latency associated with the RAID 5 write penalty. With cache, the Read-Modify-Update sequence that requires four physical disk operations to execute can be performed asynchronously from the standpoint of the host I/O request. On a disk write operation, the host writes data to the controller cache first, which signals a successful completion when the data has been safely deposited there. This is sometimes known as a *Fast Write* to cache. Later, asynchronous to the host machine, the controller performs the Read-Modify-Update sequence of physical disk operations that hardens the data stored in the array.

Utilizing fast writes to cache to mask the latency associated with disk write operations creates a delay between the time the host machine considers the I/O operation complete and the time the physical disks in the RAID 5 array are actually updated to reflect the current data. During this delay, there is a data integrity exposure in which the state of the host application is not consistent with the data recorded on the physical disks. Cache controllers must cope with this data integrity exposure somehow, normally by using some combination of cache memory redundancy and fail-safe battery-backup schemes to ensure that all pending write operations ultimately complete successfully. Loss of power to the cache memory is the most serious risk that must be guarded against. Note that failure of one of the physical disk drives involved in the operation is not a major concern because of the redundant data-recording scheme.

Another performance-oriented optimization designed to help improve the performance of disk writes in RAID 5 is deferring writing back to disk until an entire array data stripe can be written. When the controller can defer writes until a full array data stripe is updated—usually by a sequential write operation—the old parity sector associated with the data stripe never needs to be read and has to be written back only once. This results in a significant reduction in the number of back-end disk operations that need to be performed.

In configuring cached disk controllers to achieve the proper level of performance, two considerations are paramount:

- Battery backup of controller cache memory to preserve the updated data stored there until it can be written to the back-end disks. This is essential.
- Providing enough memory to cache most write activity, which typically occurs in large bursts associated with lazy write flushes of host cache memory or database commits.

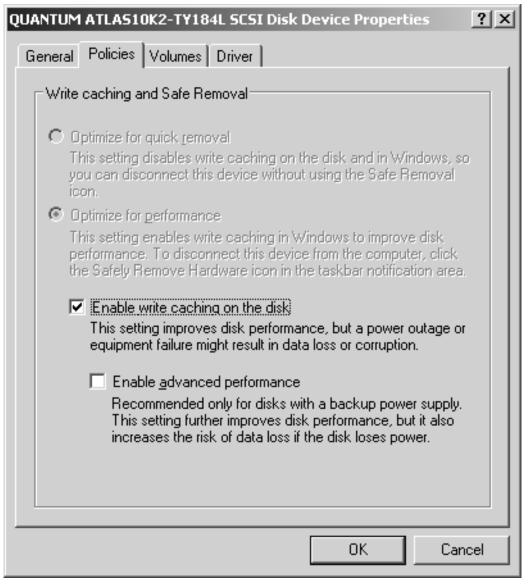
Sizing controller cache A relatively small amount of battery-backed, controller cache can shield your servers from the additional disk latency associated with RAID level 1 and 5 writes. To size the cache, observe the peak write I/O rate for your physical disks. Then, assume the cache needs to be large enough to hold a 1-minute burst of write activity. The following formula estimates the amount of battery-backed cache memory you need per Physical Disk:

Cache memory = peak(PhysicalDisk(n)\Disk Writes/sec) × *file system allocation unit* × *60*

Because write operations ultimately lead to physical disk operations, the overall level of write activity a cached disk subsystem can sustain sets an upper limit on performance. This is largely a function of the number of physical disks that the subsystem contains. Generally speaking, having more disks to perform the work is always better. In RAID levels 1, 0/1, and 5 configurations, remember that maintaining redundant data requires two physical disk writes for every host-initiated write operation. Consequently, the number of writes a RAID disk subsystem can handle is actually only one-half of the capacity of its physical disk drives.

If the rate of write requests exceeds the capacity of the physical disks in the array to perform writes, the cache begins to fill up with data waiting to be written to disk. If this write activity is sustained for any period of time, the cache will fill up completely. Once the cache is full, subsequent write requests must wait until physical back-end operations to disk complete. When the cache saturates, the disk controller can no longer effectively mask the poor performance of RAID write operations.

Windows Server 2003 provides two tuning parameters that allow you a degree of control over write caching. You can access these by using the Disk Management snap-in: right-click a physical disk and open the Properties Pages. The Policies tab shows the write caching parameters, as shown in Figure 5-30. By default, if the physical disk supports write caching, write caching is enabled, as illustrated. If the Enable Advanced Performance option is selected, the disk device driver will not set the *WRITE-THROUGH* flag on any SCSI write commands. This gives a cached disk controller the freedom to defer all physical writes to disk. This option should never be enabled unless you are positive the controller provides full battery backup for cache memory and any attached disks.


QUANTUM ATLAS10K2-TY184L SCSI Disk Device Properties

Write caching and Safe Removal

( ) Optimize for quick removal
This setting disables write caching on the disk and in Windows, so you can disconnect this device without using the Safe Removal icon.

(●) Optimize for performance
This setting enables write caching in Windows to improve disk performance. To disconnect this device from the computer, click the Safely Remove Hardware icon in the taskbar notification area.

[✓] Enable write caching on the disk
This setting improves disk performance, but a power outage or equipment failure might result in data loss or corruption.

[ ] Enable advanced performance
Recommended only for disks with a backup power supply. This setting further improves disk performance, but it also increases the risk of data loss if the disk loses power.

Figure 5-30 Enable advanced performance only for disks that are fully protected by a backup power supply

Virtual disks RAID controllers allow you to define virtual disk images that are mapped to a set of physical disks in the array. The number of physical disks used in the array, their RAID levels, and the number of virtual disk images that are mapped to these disks are usually programmable options. Each virtual disk is identified by a unique Logical Unit Number (LUN) used in the SCSI protocol, which in turn appears as a physical disk to the Windows Server 2003 operating system. The operating system is aware only of the apparent characteristics of the virtual disk. The operating system cannot see what is happening under the covers of the disk controller—how many physical disks are involved, the data redundancy scheme employed, whether there is controller cache memory, and so on. These details are frequently crucial in diagnosing a disk performance problem.

Meanwhile, the disk performance statistics available in the Performance Monitor reliably measure the performance of these virtual disks. The measurement layers in the I/O Manager stack accurately measure the response time of disk requests. However, the details of how a virtual disk is mapped to physical storage are transparent to the host computer. Additional complications arise when the virtual disks are accessed via a Storage Area Network (SAN) that supports connections to multiple host machines. The virtual disk associated with your machine might be mapped to physical disks in the SAN configuration that are *shared* by other host connections. The I/O operations initiated by your host computer might be contending with I/Os from other host computers directed to the same set of physical disks in the SAN.

Host-centric measurements cannot see the impact of I/O operations on virtual disks on physical storage resources like disk drives, paths, and switches. Because these resources are shared across host machines, the activity from any one host can interfere with the activity of another. When disk performance suffers, you will likely need to understand the details of how virtual disks are mapped to physical disks, the topology of connecting links and switches that are shared by the devices, and so on. For a complete picture of what is happening to the devices interconnected on a SAN, you need to pool all the host-based measurements for the virtual disks that are defined. These might also be internal measurements from storage controllers and other virtualization engines that are available regarding physical disk, link, and switch performance for devices in the shared storage pool. This might also include utilization measurements of shared network switches. These internal measurements can be very useful if they can be correlated with the host's view of this disk activity provided by the Performance Monitor.

A unified view of the shared hardware environment is often necessary to understand why disk performance problems are occurring and what can be done about them. During a disk-to-tape backup operation, for example, your host machine is attempting to read data from a virtual disk as fast as possible and write it to the backup tape device. Because tape drives usually have greater bandwidth than the disks, disk speed is the bottleneck during direct disk-to-tape backup operations. (If one of the devices is being accessed remotely, the bottleneck usually shifts to the network bandwidth.) During disk-to-tape operations, the disk is accessed at its saturation level. If the disk is a virtual disk that is in actuality mapped to a set of physical drives within the SAN, the disk activity from the backup operation will likely impact any other attempts to access those physical disks from other hosts. Storage administrators often find that they must schedule disk-to-tape backups carefully in a SAN to minimize contention across different host systems.

Diagnosing Disk Performance Problems

The first step in diagnosing a potential disk performance problem is to gather the disk performance statistics using the Performance Monitor during a representative period of peak usage. To enable you to analyze the performance data you gather, use a background data collection session using the Log Manager utility or the counter logs graphical user interface in the System Monitor console. Gather the counters from the Physical Disk object, as well as from any other counters that might be useful—the Memory counters that report physical disk paging rates, processor utilization, and the interrupt rate from the Processor object; the Processor Queue Length from the System object; and the Logical Disk counters, if the way logical volumes are mapped to physical disks is significant.

The procedure is as follows:

1. Determine the disk workload characteristics: I/O rates, read vs. write rates, and average block sizes. The counters in Table 5-14 provide this information.

Table 5-14 Counters That Characterize Disk Workload

Workload Characteristic	Object\Counter
Read I/O rate	Physical Disk(*n*)\Disk Reads/sec
Write I/O rate	Physical Disk(*n*)\Disk Writes/sec
Read block size	Physical Disk(*n*)\Avg. Disk Bytes/Read
Write block size	Physical Disk(*n*)\Avg. Disk Bytes/Write

2. Based on the manufacturer's specifications or your own direct measurements, determine the expected disk service time for this workload.
3. Compare the physical disk response time—Physical Disk(*n*)\Avg. Disk secs/Read and Physical Disk(*n*)\Avg. Disk secs/Write—that you observe to the expected disk service that the physical device can provide. If the observed disk I/O response time is less than 150 percent of the expected disk service time, any tuning strategies you employ cannot improve performance significantly. It is usually not worth spending lots of time trying to improve disk performance by that modest amount. If you need much better performance than your current disks can provide, buy faster disks.
4. Calculate physical disk utilization:

   Physical Disk(n)\% Busy = 100% —Physical Disk(n)\% Idle Time

If the physical disk is only modestly busy—less than 10–20 percent busy, say— the disk I/O response time is not likely to create an application bottleneck, no matter how bad it is.

If disk response time is poor compared to your service level expectation, and the disk is at least 40–50 percent busy, decompose disk response time into its device service time and device queue time components. To accomplish this, Relog the counters in Table 5-15 into a text format file that you can process using Excel.

Table 5-15 Counters for Decomposing Disk Response Time

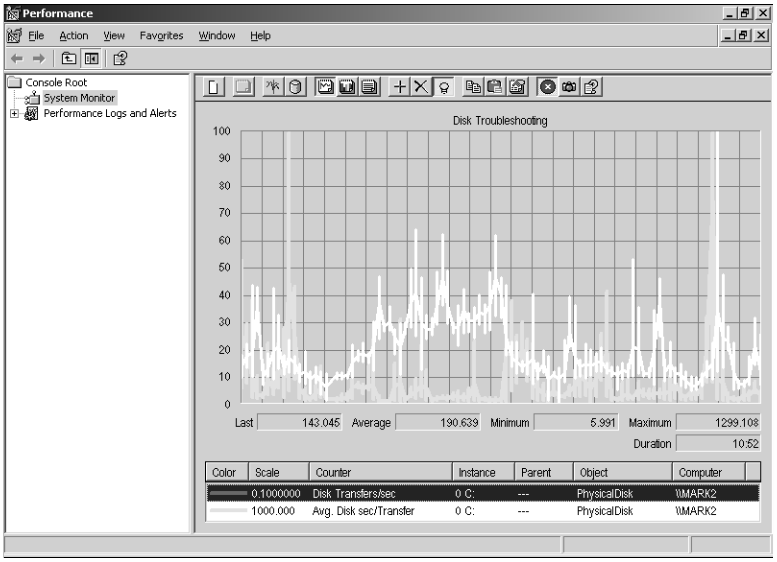
Object\Counter	Corresponds To
Physical Disk(*n*)\% Idle Time	The inverse of disk utilization
Physical Disk(*n*)\Avg. Disk sec/Transfer	The disk response time
Physical Disk(*n*)\Disk Transfers/sec	The I/O rate

Applying the Utilization Law, calculate the metrics in Table 5-16:

Table 5-16 Formulas for Calculating Disk Service Time

Metric	Formula
Physical Disk(*n*)\% Disk Busy	100%–Physical Disk(*n*)\% Idle Time
Physical Disk(*n*)\Avg. Disk Service Time/Transfer	Physical Disk(*n*)\% Disk Busy ÷ Physical Disk(*n*)\Disk Transfers/sec
Physical Disk(*n*)\Avg. Disk Queue Time/Transfer	Physical Disk(*n*)\Avg. Disk secs/Transfer – Physical Disk(*n*)\Avg. Disk Service Time/Transfer

For example, Figure 5-31 shows the disk performance statistics gathered for a single disk on a Windows Server 2003–based machine that appeared to be running slowly.



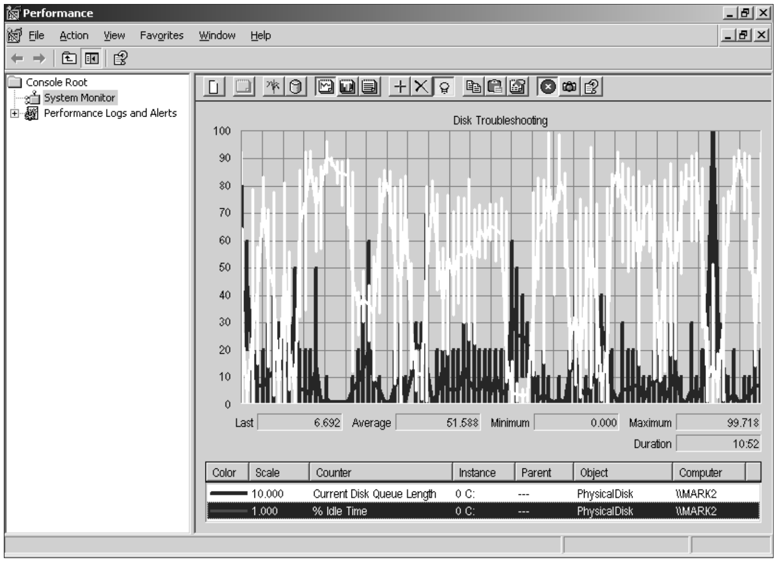
Last 143.045 Average 190.639 Minimum 5.991 Maximum 1299.108
Duration 10:52

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	0.1000000	Disk Transfers/sec	0 C:	---	PhysicalDisk	\\MARK2
▬	1000.000	Avg. Disk sec/Transfer	0 C:	---	PhysicalDisk	\\MARK2

Figure 5-31 Disk performance statistics gathered for a single disk on a Windows Server 2003–based machine

The disk I/O rate and response time for a period of about 11 minutes is shown in the figure. During this period, the I/O rate (highlighted) ranges from 6 to 1300 I/Os per second, with an average of about 190 Disk Transfers/sec. The disk response time over the same interval averaged a very respectable 8 milliseconds, but also showed several spikes in which response time exceeded 20 milliseconds or more. Because the server appeared sluggish during this interval, which was marked by heavy activity from client machines accessing files over the network stored on this machine, further investigation is necessary.

The next appropriate action is to check disk utilization, as shown in Figure 5-32, to see whether the disk is a potential bottleneck.



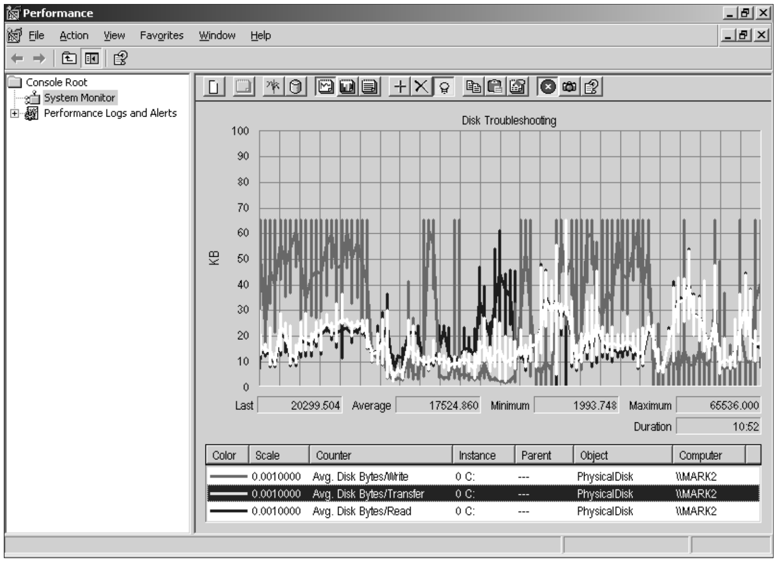
Last	Average	Minimum	Maximum	Duration
6.692	51.588	0.000	99.718	10:52

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	10.000	Current Disk Queue Length	0 C:	---	PhysicalDisk	\\MARK2
▬	1.000	% Idle Time	0 C:	---	PhysicalDisk	\\MARK2

Figure 5-32 Checking disk utilization

% Idle Time is a somewhat counter-intuitive way to look at disk utilization. Subtract % Idle Time from 100 percent to calculate % Disk Busy. The average disk busy over the same interval is only 50 percent, but many peaks exist where disk utilization exceeded 70 percent. The Current Disk Queue Length counter that reports an instantaneous value of the number of outstanding I/Os to the device is also shown. This value is charted using a scaling factor of 10 to make it easier to read alongside % Idle Time. The highest instantaneous disk queue length value reported was 47. The average reported was only 1. Although this is not a large backlog of requests, on average, there is enough evidence of disk queuing delays to warrant further investigation.

At this point, it is helpful to examine the block size reported for both read and write requests. This data is illustrated in Figure 5-33.



Last	Average	Minimum	Maximum	Duration
20299.504	17524.860	1993.748	65536.000	10:52

Color	Scale	Counter	Instance	Parent	Object	Computer
	0.0010000	Avg. Disk Bytes/Write	0 C:	---	PhysicalDisk	\\MARK2
	0.0010000	Avg. Disk Bytes/Transfer	0 C:	---	PhysicalDisk	\\MARK2
	0.0010000	Avg. Disk Bytes/Read	0 C:	---	PhysicalDisk	\\MARK2

Figure 5-33 Block size for both read and write requests

Caution Scaling values for each line on this chart have been set to .001, not to their default values. This allows values for these counters to be plotted against a y-axis calibrated in kilobytes (KB). Modifying the automatic scaling of these values, which is designed to fit them automatically into a range of values from 0–100, to report these measures in KB makes the values reported less susceptible to errors of interpretation. Use scaling factors judiciously when you need to display widely dissimilar values on a single chart.

The overall average block size at about 17 KB is the highlighted line on this chart, sandwiched between the average block size of read requests (the lower value) and the much larger average value for write requests. Be careful when you need to interpret values of these counters that report average values. The summary statistics calculated by the System Monitor can be very misleading because they report an average of the average values that are calculated at each interval. This issue is discussed in Chapter 2 and Chapter 3. Statistically, calculating a weighted average is the proper way to proceed. Calculating the average of a series averaged values can lead to many anomalies. In this example, reads averaged almost 19 KB per disk I/O request, which reflects many bulk requests larger than the 4-KB file system allocation unit. Meanwhile, the average write block was about 29 KB, with the graph showing many intervals in which the average write block size reached 64 KB. The statistical anomaly that results from calculating the average of averages is that Avg. Disk Bytes/Transfer is less than the average of *both* Avg. Disk Bytes/Read and Avg. Disk Bytes/Write. Be sure to relog these measurements to a text format file that can be processed using Excel to calculate valid weighted averages of these counters.

In the example, those large, multiblock write operations are the by-product of lazy write file or database cache flushes, or both, that send writes to the disk in bulk. The average rate of write requests was only 44.5 per second for a read/write ratio of about 3:1. But because the average bulk write request moved more data than the average read request, almost twice as much data is being written to disk than is being read back. This is normal. Effective host-caching reduces the number of disk read operations that need to be performed. Cache can postpone the need to post file updates to the disk, but ultimately the data on disk must be changed to remain synchronized with the application's view of that data.

The next step in the analysis is to Relog the counters shown to a .csv format file that can be processed in Excel. The following command string will accomplish this:

```
relog "Disk performance Test_2003081312.blg" -c "\PhysicalDisk(0 C:)\Avg. Disk sec/
Transfer" "\PhysicalDisk(0 C:)\Disk Transfers/
sec" "\PhysicalDisk(0 C:)\% Idle Time" "\PhysicalDisk(0 C:)\Current Disk Queue Lengt
h" -f csv -o "Disk performance Test_2003081312.csv" -y
```

Use Excel to calculate the disk utilization, disk service time, and disk queue time, as described in Table 5-11. Graphing the results yields the chart shown in Figure 5-34.

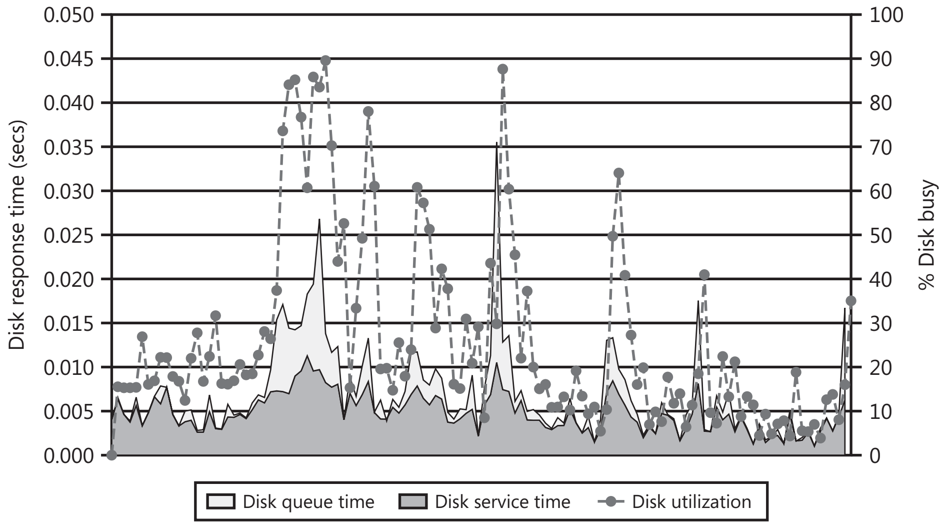


Figure 5-34 Excel graph showing disk utilization, disk service time, and disk queue time

In this Excel combination chart, disk service time and queue time are graphed using a stacked Area chart against the left y-axis. Stacking the queue time on top of the service time in this fashion allows your eye to add the two values visually—this is the advantage of using a professional charting tool like Excel to augment the Performance Monitor Chart view. Disk utilization is also calculated and shown as a dotted line plotted against the right y-axis. When all the data is viewed in this fashion, the relationship between spikes in disk utilization and disk response time degradation are clearly visible.

Breaking down disk response time in this fashion allows you to determine whether you have a disk that is performing poorly or is overloaded, and choose an appropriate tuning strategy. If disk service time is much higher than can be reasonably expected from that type of hardware, you might be able to improve disk performance significantly by using the disk defragmentation utility. Instead, if the disk is overloaded, you will probably need to try a different approach—adding more disks to the system or spreading some of the load from this busy disk to some lower utilized ones in the configuration. The various disk tuning strategies and when to use them are discussed in the next section.

Meanwhile, it might be instructive to complete the analysis of this example. From the chart in Figure 5-34, you can see that disk service time is consistently 5 ms or less, which is better than the expected range of performance for this disk for this specific workload. Because disk performance is adequate, performing a disk defragmentation, or conducting other tuning actions designed to squeeze a bit more performance out of this disk, is not likely lead to a major improvement in service time. (For more information about the benefits of disk defragmentation, see the next section.) A faster disk could perform the same workload as much as 30 percent faster, but you must consider operational issues when making that kind of disruptive change. No doubt, some positive incremental improvements can be made, but there is no magical solution.

The periods of performance degradation that do occur are evidently the result of the disk becoming overloaded and disk requests being queued. Periods in which there is an increase in disk queue time match almost exactly periods in which disk utilization also spikes. The disk utilization spikes probably result from the bursty, bulk write workload that is associated with lazy write cache flushes. These bulk writes tie up the disk for a relatively long period and generate contention with the read workload. In the interval observed, the periods of serious disk congestion are intermittent. When disk contention of this sort starts to become chronic and is impacting the performance of key server applications, it is time to do something about it.

Configuration and Tuning Strategies to Improve Disk Performance

The performance of many applications is constrained by the performance of the slowest computer component used to process client requests. The slowest internal component of most servers is the disk. Server application responses are often found waiting for disk requests to complete—more so than for any other computer processing component. Even when computer disks are performing well—that is, disk service time is within expectations and queuing delays are minimal—server application performance suffers because of delays in processing I/O requests by relatively slow, mechanical disks. Computer performance professionals call these applications *I/O bound*, compared to applications that are *compute-bound* or *network-constrained*. The only way to improve the performance of an I/O bound application significantly is to improve disk performance.

This section discusses the most important and most common configuration and tuning strategies that are employed to improve disk performance. Configuration options include:

- Deploying faster disks
- Spreading the disk workload over more disks
- Using disk array technology and RAID to spread the disk workload over more disks
- Implementing cached disk controllers

In addition, the following disk tuning strategies will be discussed:

- Using memory-resident buffering to reduce physical disk I/Os
- Eliminating disk hot spots by balancing the I/O workload across multiple disks
- Defragmenting disks on a regular basis to increase the amount of sequential disk processing

Adopting the most effective disk tuning strategy begins with understanding where your disk bottleneck is. As discussed in the previous section, "Diagnosing Disk Performance Problems," you need to find out whether the disk is performing poorly or the disk is overloaded. Some of the configuration and tuning strategies discussed here attack the problem of poor disk service time. Improving disk service time reduces the load on the disk, so indirectly these strategies also reduce queuing delays. Other strategies are primarily aimed at workload balancing, which has very little to do with improving disk service time, but directly reduces queuing delays. If disk service time is poor, a strategy that attacks queuing delays will yield little improvement. If disk queue time delay is a major factor, a strategy aimed at balancing the disk workload better is often the easiest and simplest approach. These considerations are summarized in Table 5-17 for each of the disk configuration and tuning strategies under consideration.

Table 5-17 Disk Performance and Tuning Strategies

Strategy		Performance Impact
Configuration	Faster Disks	■ Directly improves disk service time. ■ Indirectly reduces queuing.
	More disks	■ Directly reduces queuing.
	Disk arrays	■ Directly reduces queuing by balancing the load across multiple disks. ■ RAID configurations suffer a write performance penalty.
	Cached disks	■ Directly improves disk service time. ■ Indirectly reduces queuing.
Tuning	Host-based caching	■ Reduces the overall physical disk I/O load. ■ Boosts the relative write content of the physical disk workload. ■ Might increase disk service time.
	Load balancing	■ Directly reduces queuing.
	Defragmentation	■ Directly improves disk service time. ■ Indirectly reduces queuing.

Many server applications are I/O bound, capable of running only as quickly or as slowly as the mechanical disks they must access. Configuring a disk subsystem for optimal performance and tuning it to maintain optimal performance levels is a critical activity in many environments. When disk performance is so important, *any* action that improves disk performance might be a worthwhile pursuit.

Performance-oriented disk configuration options The following list describes the configuration options that can be employed to improve disk performance:

- **Faster disks** Disk manufacturers build a range of devices that span a wide spectrum of capacity, price, and performance. You might be able to upgrade to disk devices with improved seek time, faster rotational speeds, and higher throughput rates. Faster devices will improve disk service time directly. Assuming I/O rates remain constant, device utilization is lowered and queuing delays are reduced.

> **Tip** The highest performing disks are also the most expensive. A device that improves disk service time by 20–30 percent might be 50 percent more expensive than standard models.
>
> When the other strategies under consideration may fail for one reason or another, buying faster disks always accomplishes something. Of course, if you are already using the fastest disk devices available, you will have to try something else.

- **More disks** The driving force behind most disk purchases is storage capacity. When cost and capacity are the only important considerations, you will buy large capacity disks that have the lowest cost per megabyte of storage. When performance is an important consideration, you need to purchase higher cost, faster disks, and more of them. If you are running with the fastest disks possible and you are driving utilization consistently above 50 percent busy, add more disks to the configuration and spread the I/O workload across these additional disks. Spreading the load across more disks lowers the average disk utilization, which indirectly reduces queuing delays.
- **Disk arrays** The simplest way to spread the I/O workload across multiple disks is to install array controllers that automatically stripe data across multiple disks. If you are also interested in adding fault tolerance to large disk configurations, beware of the write performance penalty associated with RAID technology. Unless you are reading and writing very large blocks, do not expect that using disk arrays will improve device service time. However, by balancing the workload automatically across all the disks in the array, you eliminate disk hot spots and reduce overall queue time delays.
- **Cached disks** Cached disk controllers can often mask device latency and improve disk service time dramatically. Cache hits eliminate all mechanical device latency. Data can be transferred from the cache to the channel at full channel speeds, which is faster than data can be read or written to the disk media. For RAID controllers, consider battery-backed caches that are especially effective in masking the device latency associated with RAID write operations.

Disk tuning strategies The following list describes tuning strategies that you can use to improve disk performance:

- **Host-resident cache** The best way to improve the performance of applications constrained by disk I/O is to eliminate as many physical disk I/Os as possible using RAM-resident cache buffering. Replacing an I/O operation to disk—even one to the fastest disk drive around—with host memory access to disk data cached in RAM achieves a splendid degree of speed-up. Providing more host memory for larger application database and file caches should increase cache buffer hit rates and reduce physical disk activity in tandem.

Utilize the performance counters that report on cache effectiveness to help you in this effort. These include the Hits % counters that are provided in the Cache object for each of the three file cache interfaces. At the application level, the cache hit ratio counters in the Internet Information Services Global object and the Web Service cache object for Web-based IIS applications are extremely useful. For SQL Server databases, detailed performance counters are available that provide statistics on the Buffer Manager and the Cache Manager functions, both of which play a crucial role in minimizing physical disk accesses. For Exchange Server databases, the Database counters report on cache effectiveness, but there are other counters that are relevant, too. Information in the MSExchangeDSAccessCaches counters on the effectiveness in buffering the user credentials stored in Active Directory can be very important in improving the responsiveness of Exchange message delivery.

Another area of concern is where ample RAM is available, but application configuration parameters restrict the application from allocating more database buffers. Verify that the RAM available is being used by the applications that can use it effectively to eliminate as many physical disk I/Os as possible.

Tuning application file and database buffers to reduce the I/O load to disk almost always improves performance. However, no amount of host-resident buffering can eliminate all I/O to disk. The most effective memory-resident buffering succeeds in eliminating most disk read activity, except for those initial disk accesses. File and database updates must be propagated to the physical disk eventually, so write activity as a percentage of overall disk activity tends to increase as host-memory caching gains in effectiveness. Effective caching also tends to increase the burstiness of the physical disk operations that remain because that behavior is characteristic of lazy write flushes.

Tip When effective host-resident caching is in place for your server applications, you might need to reconfigure the physical disks so that they are better equipped to handle this bursty, large-block, write-oriented I/O workload. The physical I/O operations that remain after host caching has effectively eliminated most disk reads might take longer to service than the mix of I/Os that were performed prior to optimizing for host caching. Refer to the earlier section entitled "Sizing Controller Cache" for hints on how to configure cached disk subsystems for optimal performance under these conditions.

- **Balance the disks** Disk storage capacity and disk backup and restore—not disk performance—are usually the primary considerations when you set up a new server. But most system administrators initially try to arrange their files so they are at least balanced across available volumes. This is difficult to do well when you don't have the experience yet to know how these disks are accessed in a live production environment. Even if you have done a good job in balancing the I/O workload initially across the available disks, changes in the configuration and the workload over time inevitably create imbalances. Whenever one disk in the configuration becomes overloaded, its disk queue elongates and disk response time suffers. When this occurs, redistributing files and databases to better balance the I/O load will reduce disk queuing and improve response time.

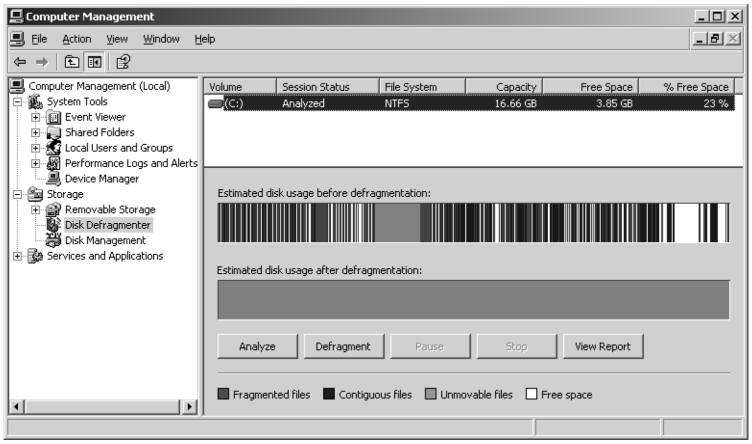
Tip An easy way to balance the I/O load across multiple disk drives is to use array controllers that automatically stripe data over multiple disks. Both RAID 0/1 and RAID 5 disk arrays spread I/O evenly across multiple disks.

- **Defragment disks to increase sequential disk reads** As file allocations start to fill up the logical disk volume (or file system), files are no longer stored on the disk in contiguous sectors. If the sectors allocated to the file are not contiguous, attempts to read the file sequentially (from beginning to end) require more disk head movement because of this *fragmentation*. Instead of sequential operations that need no disk arm seeks between successive disk requests, disk seeks back and forth across the disk are required as successive fragments of the file are accessed. Disk service time is impacted. Given how fast disk drives with built-in cache buffers can process sequential requests, as illustrated in the section entitled "Establishing a Disk Drive Performance Baseline," increasing the percentage of sequential disk requests that are issued will speed up disk processing. Long-running sequential processing requests can especially benefit from defragmentation. These include file transfers, disk-to-tape backup, and other lengthy file copying requests.

Windows Server 2003 provides a built-in file system defragmentation utility. Some tips on using the defragmentation utility effectively, along with a brief case study that illustrates its benefits, are provided in the following sections.

Disk Defragmentation Made Easy

The Disk Defragmenter utility, which you can access from the Computer Management program on the Administrative Tools menu, is shown in Figure 5-35.



Volume	Session Status	File System	Capacity	Free Space	% Free Space
(C:)	Analyzed	NTFS	16.66 GB	3.85 GB	23 %

Figure 5-35 Disk Defragmenter utility

Using Explorer, you can also launch the Disk Defragmenter by right-clicking a volume, accessing its Properties Pages, and selecting the Tools Tab.

You can also execute the Disk Defragmenter utility from the command line. This capability allows you to develop automated procedures to defragment the disks on your production servers on a regular basis.

Using the disk defragmentation utility effectively Setting up automated procedures to run the Disk Defragmenter utility on a daily or a weekly basis by using the Disk Defragmenter command-line utility is a simple matter. However, not every volume in your disk farm is likely to benefit equally from regular defragmentation. Consider the following:

- I/O activity from the Disk Defragmenter utility can tie up large volumes for a considerable length of time.
- The Disk Defragmenter utility needs adequate free space on the volume to run efficiently and do a good job in reorganizing the volume. Defragmentation that runs on volumes that are more than 75 percent allocated are likely to take considerably longer to complete and might not yield dramatic performance improvements.
- For very volatile volumes—where files are subject to continuous allocation, deletion, and growth—the benefit of defragmentation is liable to be very short-lived.

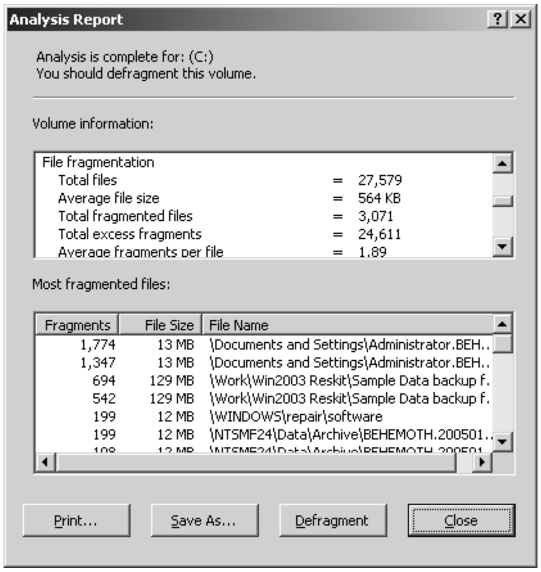
These factors all point to *not* implementing across-the-board defragmentation procedures without performing *some* workload analysis to assess the relative benefits and costs. Fortunately, the Disk Defragmenter utility contains a facility for reporting and analysis, which you can use to fine-tune your regular defragmentation procedures.

Analyzing volumes before defragmenting them When the volume fragmentation analysis function completes, a dialog box displays the percentage of fragmented files and folders on the volume and recommends whether or not to defragment the volume.

Figure 5-36 illustrates the defragmentation analysis report for the logical volume shown in Figure 5-35. An analysis report can also be generated using the Disk Defragmenter line command, as illustrated In Listing 5-10.

Listing 5-10 Results of Using the Disk Defragmenter Command

```
C:\>defrag c: /a /v
Windows Disk Defragmenter
Copyright (c) 2003 Microsoft Corp. and Executive Software International, Inc.

Analysis Report

    Volume size                         = 16.66 GB
    Cluster size                        = 4 KB
    Used space                          = 12.81 GB
    Free space                          = 3.85 GB
    Percent free space                  = 23 %

Volume fragmentation
    Total fragmentation                 = 18 %
    File fragmentation                  = 35 %
    Free space fragmentation            = 2 %

File fragmentation
    Total files                         = 27,633
    Average file size                   = 563 KB
    Total fragmented files              = 3,086
    Total excess fragments              = 24,683
    Average fragments per file          = 1.89

Pagefile fragmentation
    Pagefile size                       = 1.48 GB
    Total fragments                     = 1

Folder fragmentation
    Total folders                       = 2,452
    Fragmented folders                  = 51
    Excess folder fragments             = 293

Master File Table (MFT) fragmentation
    Total MFT size                      = 34 MB
    MFT record count                    = 30,157
    Percent MFT in use                  = 85
    Total MFT fragments                 = 3

You should defragment this volume.
```



Analysis is complete for: (C:)
You should defragment this volume.

Volume information:

```
File fragmentation
    Total files                        =  27,579
    Average file size                  =  564 KB
    Total fragmented files             =  3,071
    Total excess fragments             =  24,611
    Average fragments per file         =  1.89
```

Most fragmented files:

Fragments	File Size	File Name
1,774	13 MB	\Documents and Settings\Administrator.BEH..
1,347	13 MB	\Documents and Settings\Administrator.BEH..
694	129 MB	\Work\Win2003 Reskit\Sample Data backup f.
542	129 MB	\Work\Win2003 Reskit\Sample Data backup f.
199	12 MB	\WINDOWS\repair\software
199	12 MB	\NTSMF24\Data\Archive\BEHEMOTH.200501..

Figure 5-36 Defragmentation analysis report

An important statistic is Average fragments per file, in this case, 1.89 fragments per file. A file is fragmented if the file data is not contiguous. This metric reports the average number of noncontiguous fragments per file. The report also shows the total number of files that are fragmented—slightly over 10 percent in this case. For the files that are fragmented, you can easily compute another useful metric—the average number of fragments:

Avg. Fragments/file = Total excess fragments / Total fragmented files

Also notice in the bottom window of Figure 5-36 the list of the most fragmented files. You can sort this list by clicking the appropriate header tab. Keep in mind that if the large files that have the most file system fragments are ones that are frequently updated or recreated, the benefit from running the Disk Defragmenter utility will likely be short-lived. These files will fragment again the next time they are updated or extended.

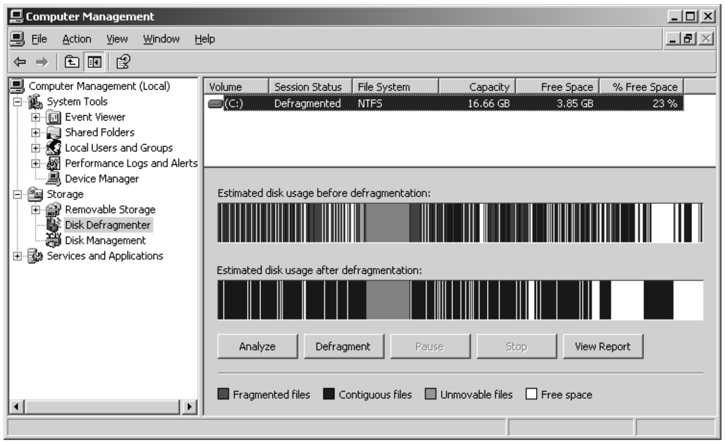
Another important consideration is that the Disk Defragmenter requires ample free space on the disk to function effectively. The Disk Defragmenter works by copying a fragmented file into a large contiguous free area on the volume and then reclaiming the space associated with the old version of the data. This method requires a substantial amount of free space on the volume to begin with. As the volume fills, the Disk Defragmenter utility must move more and more files around, so not only do Disk Defragmenter utility runs take longer, but these longer runs also yield less efficiently reorganized volumes.

Tip For best results, defragment volumes when they have at least 25 percent free space and the average number of file fragments for those files that are fragmented is greater than five. By these criteria, defragmentation of the logical volume illustrated in Figures 5-35 and 5-36 can be expected to be marginally productive.

The duration of a Disk Defragmenter run is also influenced by the size of the volume, the number of files on the volume, and the number and size of the fragmented files that will be moved. Open files cannot be defragmented, so you must choose a period of relatively low file access activity to run the Disk Defragmenter utility. A Disk Defragmenter run against the logical volume pictured in Figures 5-35 and 5-36 took over 2 hours on an otherwise idle system. During that period, the logical volume was so busy with defragmenting activity that the volume could sustain very little other I/O activity from other sources. Try to pick a time to run the Disk Defragmenter utility when the volume is otherwise idle.

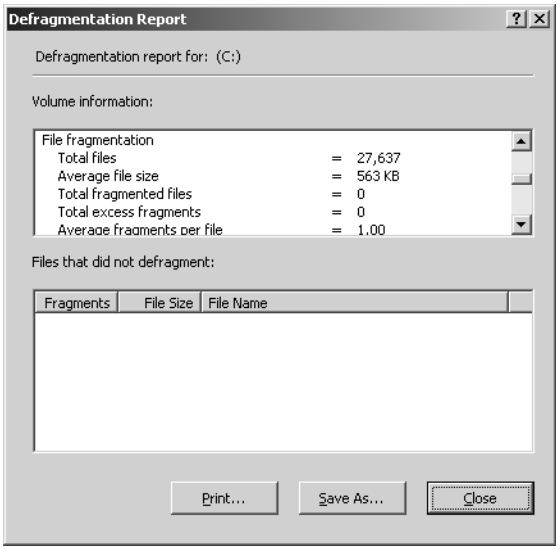
Volumes can become excessively fragmented after any activity that adds a large number of files or folders to the volume. For example, consider defragmenting volumes immediately after installing software or after performing an upgrade or a clean install of Windows. For maximum benefit, defragment a volume just before you run backup or file replication utilities that move large numbers of files from disk-to-tape or disk-to-disk.

Figure 5-37 shows the volume depicted earlier after defragging. The analysis report in Figure 5-38 provides additional detail.


Computer Management
File
Action
View
Window
Help
Computer Management (Local)
System Tools
Event Viewer
Shared Folders
Local Users and Groups
Performance Logs and Alerts
Device Manager
Storage
Removable Storage
Disk Defragmenter
Disk Management
Services and Applications
Estimated disk usage before defragmentation:
Estimated disk usage after defragmentation:
Analyze
Defragment
Pause
Stop
View Report
▨ Fragmented files ■ Contiguous files ▨ Unmovable files □ Free space

Volume	Session Status	File System	Capacity	Free Space	% Free Space
(C:)	Defragmented	NTFS	16.66 GB	3.85 GB	23 %

Figure 5-37 Volume after defragmenting



Defragmentation report for: (C:)

Volume information:

```
File fragmentation
   Total files                          =  27,637
   Average file size                    =  563 KB
   Total fragmented files               =  0
   Total excess fragments               =  0
   Average fragments per file           =  1.00
```

Files that did not defragment:

Fragments	File Size	File Name

Figure 5-38 Analysis report providing additional detail of the defragmentation

The volume contained 30 GB of data on a volume with only 23% free space. Because there is adequate free space to aid and abet the reorganization process, following the defragmentation run, there were no fragmented files remaining. Keep in mind that if there are open files while the utility is running, it may not be possible to defragment them completely. The Disk Defragmenter utility ran for slightly more than 2 hours to achieve this result.

Another way to evaluate the success of a defragmentation is to test the performance of the file system before and after the Disk Defragmenter utility has run. Table 5-18 shows the results of a file copy operation that moved 2.37 GB of data from this disk to another disk on a separate server, connected using a 100-megabit Ethernet network. In this case, both the source and destination disks were defragmented prior to repeating the test.

Table 5-18 **Disk Performance Before and After Defragmentation**

Timing	IOPS	Disk Response Time (ms)	Elapsed Time (min)
Before	34.5	5.6	23
After	33.9	3.2	20
% Improvement	–2%	+43%	+13%

After defragging the disk, disk response time improved 43 percent, reflecting an increased number of uninterrupted sequential accesses. Sequential operations benefit so greatly from an on-board disk buffer, as demonstrated earlier, that any tuning action you take to increase the number of sequential accesses usually pays big dividends.

Understanding File-Level Access Patterns

Whenever you find an overloaded disk volume, it is often important to know which applications are contributing to the physical disk I/O load. To attempt to balance disk I/O manually across physical disks, for example, you need to understand which files are being accessed.

The most effective way to determine which files are in use is to gather trace data. The following command sequence will collect disk I/O trace data and file details, along with process identification data:

```
logman create trace file_detail_trace -p “Windows Kernel Trace” 0x00000301 -
o C:\Perflogs\file_detail_trace -v mmddhhmm -f BIN -rf 600 -u admin “adminpassword"
logman start trace file_detail_trace
```

Caution You must be careful when you collect Disk I/O and File Details trace data, because of the size of the trace data files that might result. Be sure to limit the duration of the trace so that the trace log file does not grow too large.

After you collect the I/O trace event data, use the Tracerpt program to generate a file-level report and also generate a comma-delimited text file, which you can use to investigate file access patterns further using an application like Excel:

```
tracerpt file_detail_trace_08260156.etl -o file_detail_trace_08260156.csv -
report file_detail_trace_08260156.htm -f HTML
```

The File I/O report generated by the Tracerpt program looks similar to Table 5-19.

Table 5-19 Files Causing Most Disk IOs

File		Disk	Reads/Sec	Read size (KB)	Writes/Sec	Write Size (KB)
C:\pagefile.sys		0	9.909	4	4.444	62
Idle	0x00000000		9.909	4	4.444	62
C:\$Mft		0	3.653	3	1.025	5
Idle	0x00000000		3.653	3	1.025	5
D:\temp\PerfLogs\SMB LAN File transfer capture.cap		0	0.000	0	3.682	63
Idle	0x00000000		0.000	0	3.682	63
C:\WINDOWS\system32\config\software		0	2.445	4	0.225	14
Idle	0x00000000		2.445	4	0.225	14

The file I/O report shows the five files that were most active during the trace. (Only the top four entries are illustrated in Table 5-19.) Note that the I/O rates reported reflect the duration of the trace. If a file is active only during a small portion of the trace, the average I/O rate reported will reflect that long idle interval. On most file servers, for example, accesses patterns tend to be distributed over a wide range of files, with very few files active for very long.

Because file system I/O requests are normally routed through the Cache Manager and handled asynchronously, it is difficult to tie the physical disk activity to the originating application process. You might have to do some detective work to tie the file name back to the application that initiated the request. Read activity to Pagefile.sys indicates sustained demand paging operations on program code or data segments. Notice that the average block size of read requests to the paging file are quite large, reflecting aggressive prefetching of contiguous pages from the paging file during the period that the trace was active. Disks with on-board cache buffers handle sequential requests very quickly, so this is an effective disk performance optimization.

File details Directing Tracerpt to produce a comma-delimited output file containing the file I/O event trace detail data that you can access using Excel is useful for finding hidden sources of disk activity. Using Excel to leaf through mountains of trace data is less than ideal, but with a little ingenuity you can make the application work for you. If you open the comma-separated version of the trace file in Excel or Microsoft Notepad, you will see a display similar to Table 5-20.

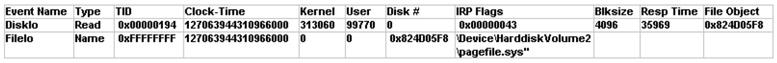
Table 5-20 A File Details Event Trace Example

Event Name	Type	TID	Clock-Time	Kernel (ms)	User (ms)	User Data
EventTrace	Header	0x00000F88	127063944258190000	0	0	131072
EventTrace	Header	0x00000F88	127063944258190000	0	0	769
SystemConfig	CPU	0x00000F88	127063944258591000	10	0	1595
SystemConfig	Video	0x00000F88	127063944258991000	10	0	67108864
SystemConfig	Video	0x00000F88	127063944258991000	10	0	67108864
SystemConfig	PhyDisk	0x00000F88	127063944258991000	10	0	0

Each trace entry contains a set of standard header fields, followed by User Data, which are data fields specific to the event type. The beginning of the file contains a series of *SystemConfig* entries and other trace data that helps you identify processes, threads, and open files. For troubleshooting disk performance problems, the *DiskIo* and *FileIo* events are the most important.

Follow these steps to work your way through a *DiskIo* trace file using Excel:

1. Use the Window, Freeze Panes command to establish the first line as a column header that will remain fixed at the top of the display as you scroll through the data.
2. Replace the User Data label in cell G1 with **Disk Number**. The first User Data field in the *DiskIo* trace data contains the Disk Number.
3. Add the remaining column header names for the *DiskIo* User Data fields. Place the following text in cells H1-L1: **IRP Flags**, **Blksize**, **RespTm**, **Byte Offset**, **File Object**.
4. Find the first *DiskIo* entry in the Event Name column. You will see something like the following results.



Event Name	Type	TID	Clock-Time	Kernel	User	Disk #	IRP Flags	Blksize	Resp Time	File Object
DiskIo	Read	0x00000194	127063944310966000	313060	99770	0	0x00000043	4096	35969	0x824D05F8
FileIo	Name	0xFFFFFFFF	127063944310966000	0	0	0x824D05F8	\Device\HarddiskVolume2 \pagefile.sys"			

Notice that a *FileIo* Name trace entry immediately follows the *DiskIo* entry. The User Data fields for FileIo Name Event entry contain the same File Object ID as the previous *DiskIo* event, followed by the file name. By matching the File Object ID of the *DiskIo* event with a subsequent *FileIo* Name event, you can determine the file being accessed.

The *DiskIo* trace User Data provides information about the type of I/O request, the block size, the byte offset into the file system, and the response time. The response time of the *DiskIo* event is recorded in timer units, which is a function of the *-ct* clock timer parameter for that specific trace log.

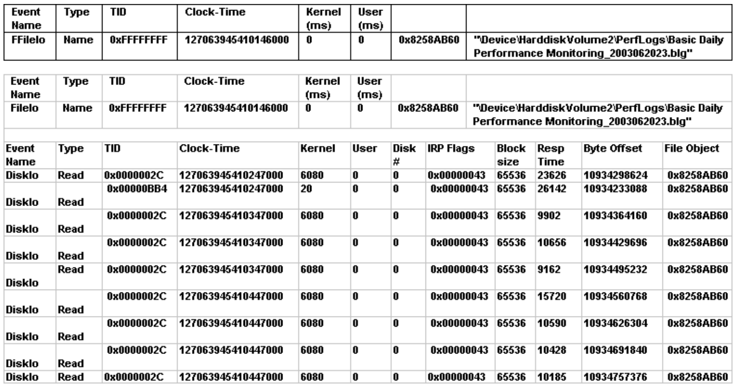
Tip Response time of *DiskIo* events is measured and reported in the trace event records in timer units. The *-ct* parameter determines the clock resolution of the timer units used in the trace. For best results, use *-ct perf* when you are gathering file-level I/O details. Neither the default value for the *system* clock resolution nor the more efficient *cycle* time setting will provide enough granularity to accurately measure the response time of individual I/O events.

The IRP flags tell you the type of operation:

```
#define IRP_NOCACHE                     0x00000001
#define IRP_PAGING_IO                   0x00000002
#define IRP_MOUNT_COMPLETION            0x00000002
#define IRP_SYNCHRONOUS_API             0x00000004
#define IRP_ASSOCIATED_IRP              0x00000008
#define IRP_BUFFERED_IO                 0x00000010
#define IRP_DEALLOCATE_BUFFER           0x00000020
#define IRP_INPUT_OPERATION             0x00000040
#define IRP_SYNCHRONOUS_PAGING_IO       0x00000040
#define IRP_CREATE_OPERATION            0x00000080
#define IRP_READ_OPERATION              0x00000100
#define IRP_WRITE_OPERATION             0x00000200
#define IRP_CLOSE_OPERATION             0x00000400
```

The I/O with IRP flags of 0x00000043 is a hard page fault. Because most file I/O is diverted to use the system file cache, many *DiskIo* events appear to be initiated by the operating system instead of by the original process that requested the data. This is true of write operations as well. Writes to cached files remain in file cache virtual memory until they are flushed by a Lazy Writer task. The write to disk occurs later when it is initiated by a system cache lazy write thread. Alternatively, if the page containing file cache happens to be a candidate selected for page trimming, that page will be written to disk by the Modifed Page Writer thread. In either case, it is not possible to tie the event back to the process that originally was updating the file. Of course, the file name can usually help you determine the process or application that originated the request.

The following is an ETW example showing a sequence of disk writes to a System Monitor binary log file.



Event Name	Type	TID	Clock-Time	Kernel (ms)	User (ms)		
FFileIo	Name	0xFFFFFFFF	127063945410146000	0	0	0x8258AB60	"\Device\HarddiskVolume2\PerfLogs\Basic Daily Performance Monitoring_2003062023.blg"

Event Name	Type	TID	Clock-Time	Kernel (ms)	User (ms)		
FileIo	Name	0xFFFFFFFF	127063945410146000	0	0	0x8258AB60	"\Device\HarddiskVolume2\PerfLogs\Basic Daily Performance Monitoring_2003062023.blg"

Event Name	Type	TID	Clock-Time	Kernel	User	Disk #	IRP Flags	Block size	Resp Time	Byte Offset	File Object
DiskIo	Read	0x0000002C	127063945410247000	6080	0	0	0x00000043	65536	23626	10934298624	0x8258AB60
DiskIo	Read	0x00000BB4	127063945410247000	20	0	0	0x00000043	65536	26142	10934233088	0x8258AB60
DiskIo	Read	0x0000002C	127063945410347000	6080	0	0	0x00000043	65536	9902	10934364160	0x8258AB60
DiskIo	Read	0x0000002C	127063945410347000	6080	0	0	0x00000043	65536	10656	10934429696	0x8258AB60
DiskIo	Read	0x0000002C	127063945410347000	6080	0	0	0x00000043	65536	9162	10934495232	0x8258AB60
DiskIo	Read	0x0000002C	127063945410447000	6080	0	0	0x00000043	65536	15720	10934560768	0x8258AB60
DiskIo	Read	0x0000002C	127063945410447000	6080	0	0	0x00000043	65536	10590	10934626304	0x8258AB60
DiskIo	Read	0x0000002C	127063945410447000	6080	0	0	0x00000043	65536	10428	10934691840	0x8258AB60
DiskIo	Read	0x0000002C	127063945410447000	6080	0	0	0x00000043	65536	10185	10934757376	0x8258AB60

The *FileIo* event that contains the name of the file is followed by nine *DiskIo* events that all reference the same File Object ID. Each of these *DiskIo* events is a 64-KB read operation of the specified file that is issued by the system file cache, which is prefetching the file from disk in response to an initial request to read the file from a process. In this case, the process initiating the file request presumably is the System Monitor console, which is being used to read in this counter log to check out some of the performance data it contains. Notice that the first two requests take in excess of 20 ms, whereas later requests seem to take only about 10 ms. This might reflect the operation of the on-board disk cache, which is able to satisfy requests for data later in the file directly from the device cache.

Network Troubleshooting

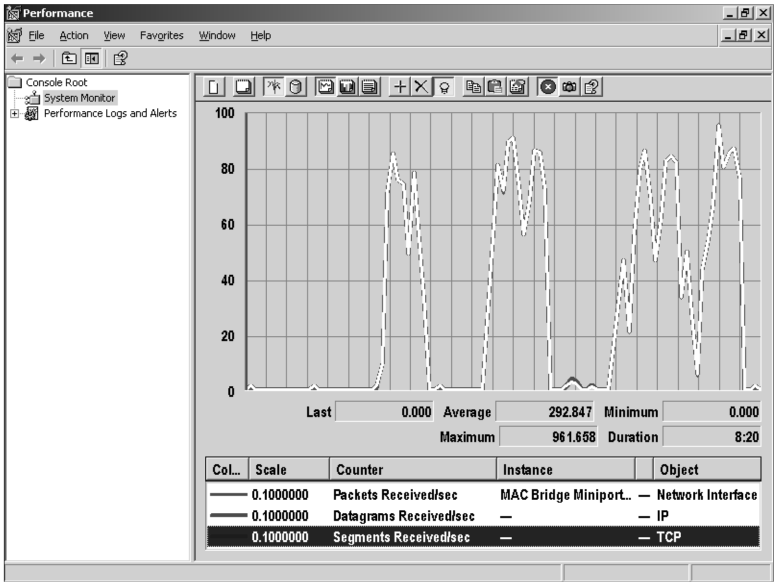
In a networked environment, network problems can affect the performance of applications running on the systems in that environment. Obviously, the network problems can affect the performance of applications that rely on the network to exchange data between server and client. In addition, the network might be a key component in the performance of many lower level tasks. For example, security checks and authentication might use the network, affecting the performance of an application that might not do any other obvious network-related task. Likewise, servers can be constrained to service network requests in a timely manner because of contention from their other workloads. In an increasingly interconnected world of distributed computing, the network plays a major role in the performance of many server applications.

Counters to Log When Troubleshooting Network Performance

The networking measurements that are available parallel the structure of the TCP/IP stack. At the lowest level—closest to the hardware—are the Network Interface counters. The IP layer sits on top of the hardware layer, and TCP sits atop IP. The IP and TCP counters are closely related to each other and to the lower level Network Interface statistics. The TCP segments that are sent and received are close cousins to the processed IP datagrams and the Ethernet packets that circulate on the network. These measures are highly correlated.

Note If you are monitoring the counters for Datagrams Sent on the sender and Datagrams Received on the receiver, and your network adapter supports TCP Large Offload (aka TCP Segmentation Offload), you will see a larger number of datagrams received than are being sent. This is because the sender sends down larger datagrams to the network adapter, while the receiver retrieves regular-sized datagrams off the network.

Figure 5-39 illustrates the degree to which the Network Interface, IP, and TCP counters are interrelated. This relationship among the counters implies that a measure from one layer can often be freely substituted for another. These counters were drawn from a session in which a large file was being downloaded to test the throughput of a wireless network interface. Three related counters are shown: Network Interface\Packets Received/sec, IPv4\Datagrams Received/sec, and TCPv4\Segments Received/sec. The values recorded for Network Interface\Packets Received/sec, IPv4\Datagrams Received/sec, and TCPv4\Segments Received/sec are almost identical. Almost every TCP segment corresponded to a single IP datagram, which in turn was received in a single Ethernet packet.


Performance
File Action View Favorites Window Help
Console Root
System Monitor
Performance Logs and Alerts
100
80
60
40
20
0

Last 0.000 Average 292.847 Minimum 0.000
Maximum 961.658 Duration 8:20

Col...	Scale	Counter	Instance		Object
—	0.1000000	Packets Received/sec	MAC Bridge Miniport...	—	Network Interface
—	0.1000000	Datagrams Received/sec	—	—	IP
—	0.1000000	Segments Received/sec	—	—	TCP

Figure 5-39 The degree to which the network interface, IP, and TCP counters are interrelated

To begin analyzing network performance, log the performance counters listed in Table 5-21.

Table 5-21 Network Performance Counters to Log

Counter	Description
Network Interface(*)*	Network Interface performance counters. Log data for all network interface cards (network adapters) except the Loopback Interface.
IPv4*	All IP layer performance counters.
TCPv4*	All TCP layer performance counters.
IPv6*	All IP layer performance counters.
TCPv6*	All TCP layer performance counters.

Note If you are monitoring a computer that is used primarily for network-based applications, such as a Web server or a file server, you should also log performance counters provided by those services or applications. For example, when monitoring a print server, you would add the Print Queue object to the list of performance counters to log.

The measurements that are available for the three primary networking software layers are throughput-oriented. Bytes, packets, datagrams, and segments sent and received are the primary metrics. These throughput-oriented measurements are mainly useful for capacity planning purposes—how much networking bandwidth is required for the application, what links are currently underutilized, and so on. Diagnosing a performance problem involving poor network response time requires measurements from a different source. This section highlights using the Network Monitor to capture and analyze individual packets to diagnose network performance problems.

Tip Pay special attention to performance counters that report error conditions, including these counters:

- Network Interface\Packets Outbound Errors
- Network Interface\Packets Received Errors
- IP\Fragmentation Failures
- TCP\Segments Retransmitted/sec
- TCP\Connection Failures
- TCP\Connections Reset

If networking problems are being reported, any nonzero occurrences of these error indicator counters should be investigated.

Above the three networking layers of system software are a number of application-oriented protocols, including HTTP, SMTP, FTP, and SMB, that support important networking applications. Each of these networking applications also provides related performance measurements: the Web service counters, the FTP and SMTP counters, and the Server and Redirector counters, among others.

Counters to Evaluate When Troubleshooting Network Performance

Ideally, you should compare the values of the counters listed in Table 5-22 to the value of these counters you derived from your baseline analysis. However, if you do not have a baseline analysis to go by, or the system has changed considerably since you last made baseline measurements, the suggested thresholds listed in Table 5-22 can be used as very rough guidelines..

Table 5-22 Network Performance Counters to Evaluate

Counter	Description	Suggested Threshold
Network Interface(*)\ Bytes Total/sec	Bytes Total/sec is the rate at which bytes are sent and received through the specified network adapter. This value includes both data and framing characters.	Investigate if less than 40% of the value of the Current Bandwidth counter.
Network Interface(*)\ Output Queue Length	Output Queue Length is the length of the output packet queue (in packets).	If this is longer than 2, there are delays and the bottleneck should be found and eliminated, if possible.
Redirector\Bytes Total/sec	Bytes Total/sec is the rate at which the Redirector is processing data bytes. This includes all application and file data in addition to protocol information such as packet headers. Use this value to determine how much of the network traffic indicated by the Network Interface(*)\Bytes Total/sec counter the Redirector service is responsible for.	If this value makes up the majority of the network traffic, this service should be investigated further to determine the cause of this traffic.
Server\ Bytes Total/sec	The number of bytes the server has sent and received through the network. This value provides an overall indication of how busy the Server service is. Use this value to determine how much of the network traffic indicated by the Network Interface(*)\Bytes Total/sec counter the Server service is responsible for.	If this value makes up the majority of the network traffic, this service should be investigated further to determine the cause of this traffic.
RAS Total\ Bytes Transmitted/sec	The number of bytes transmitted per second. Use this value to determine how much of the network traffic indicated by the Network Interface(*)\Bytes Total/sec counter the Remote Access Server service is responsible for.	If this value makes up the majority of the network traffic, this service should be investigated further to determine the cause of this traffic.

Table 5-22 Network Performance Counters to Evaluate

Counter	Description	Suggested Threshold
RAS Total\ Bytes Received/sec	The number of bytes received per second. Use this value to determine how much of the network traffic indicated by the Network Interface(*)\Bytes Total/sec counter the Remote Access Server service is responsible for.	If this value makes up the majority of the network traffic, this service should be investigated further to determine the cause of this traffic.
Process(LSASS)\ % Processor Time	The percentage of the total processor time used by the Lsass process. The Lsass process handles local security, Active Directory, and Netlogon requests.	If this value is high, look at these services to determine the root cause of the Lsass activity.
Process(System)\ % Processor Time	The percentage of the total processor time used by the System process. The System process handles NetBIOS and SMB accesses.	If the System process is unusually busy, look at these services to determine the root cause.

To quickly check current network performance on the local computer, you can also use the Task Manager Networking tab. The important columns to display at the bottom of the network performance graph are:

- **Adapter Name** This is the same name used for the corresponding instance of the Network Interface performance object, as viewed in System Monitor.
- **Network Utilization** This is the percentage of bandwidth available on the specific network adapter that was used during the display update interval. Unless the Task Manager display refresh rate is synchronized with System Monitor data collection, the values displayed for this counter will most likely be slightly different from those observed in System Monitor.
- **Bytes per Interval** This is the textual, or raw data, value that should correspond to the Network Interface(*)\Bytes Total/sec counter in System Monitor. You will need to add this column by choosing Select Columns from the View menu. Unless the Task Manager display refresh rate is synchronized with System Monitor data collection, the values displayed for this counter will most likely be slightly different from those observed in System Monitor.

The purpose of listing the different network service and application performance counters is to try to determine which, if any, is handling the most traffic. As mentioned earlier, if the server being analyzed is being used for a specific purpose, review the performance counters specific to that application or service in addition to the counters listed in Table 5-22.

The Lsass process and the System process might handle a significant amount of network-related traffic, in addition to their other functions. If network traffic is excessive *and* one or both of these processes show higher-than-normal processor activity, the traffic and processor activity might be related.

LAN Performance

In general, Local Area Network problems center on throughput and response time issues. Configuring LANs for optimal throughput and least response time by reducing congestion and minimizing contention for shared resources is the principal concern. The Network Interface counters provide genuinely useful measurement data regarding bandwidth utilization—with three major caveats:

- The Network Interface counters reflect only the host computer's view of network traffic. You must aggregate the traffic reported in the Network Interface counters across all the servers on a LAN segment to get an accurate picture of the entire network. This is best accomplished by relogging the Network Interface counters from separate machines to text format files and processing them together using an application like Excel.
- Network activity associated with retransmits because of congestion or dropped packets is aggregated across the whole networking layer, and there is no way of isolating the retransmits to a single Network Interface for investigation. This makes it difficult to determine which culprit is causing the retransmissions and degrading throughput.
- The Network Interface\Current Bandwidth counter reflects the theoretical maximum capacity of the link. If the link is running in a degraded mode because it negotiated a lower speed with some remote peer, that is not reflected in this counter's value.

The Network Interface counters provide much needed help in planning for network capacity. The statistics are economical and easy to gather. Compared to network packet sniffers, for example, they provide comprehensive, concise, and accurate information on network load factors. Relying on short-lived packet traces to plan for large-scale enterprise networks is very short-sighted, compared to utilizing the Network Interface counters to plan for network capacity. Nevertheless, for troubleshooting network performance problems, packet traces are greatly preferred because they supply a clear and unambiguous picture of events as they occur on the network. For more information about the use of the Network Monitor, the network packet tracing tool included with Windows Server 2003, see the section in this chapter entitled "Network Monitor Packet Traces."

Network Capacity Planning

With typical transmission latencies of 1 μsec or less, LAN technology is characterized by rapid response times over short distances. When LAN performance is not adequate, usually the reason is a shortage of resources on one of the host systems, or congestion on the wire that introduces major delays in successful LAN transmissions

The Network Interface counters indicate current throughput levels on that hardware interface. Even though the theoretical capacity of a link might be 10, 100, 1000, or 10,000 megabits per second, the effective throughput of the link is normally slightly less because of the overhead of protocol layers.

Tip Ethernet segments can usually saturate up to 95 percent of the theoretical network capacity. The effective utilization of an Ethernet link is reduced because of the extra network traffic and latency that gets generated from retransmits or dropped packets. Ethernet retransmits start to occur with some frequency as latency increases on the line or when network adapter buffers become saturated.

When network bandwidth is not adequate for the current tasks, you might need to upgrade your network infrastructure to provide additional bandwidth for LAN traffic. Some or all of the following upgrade paths should be considered:

- **Upgrade hubs to switches** Whereas wiring hubs provide bandwidth that must be shared across all ports, switches provide dedicated bandwidth between any two ports involved in a transmission. Collisions still occur on switched networks when two clients attempt to transmit data to the same station at the same time.
- **Upgrade to faster interfaces** The upgrade path could be accomplished by swapping out all the Ethernet interface cards and the switches used to interconnect them, thus avoiding the need to change the wiring infrastructure. During a transitional phase, until all network interfaces along a segment have been upgraded to support the higher clock transmission rates of the faster protocol, the segment is forced to run at the speed of its slowest component.

Adoption of 10-gigabit Ethernet technology is currently proceeding slowly because it often requires a costly upgrade of the wiring infrastructure, too.

- **Split over-utilized segments into multiple segments** If one network segment appears to be saturated during peak periods, it can often be segmented further to create additional capacity.
- **Use TCP/IP offload** TCP Offload Engine (TOE) is a technology touted by many hardware vendors to increase effective network bandwidth, especially with higher speed interfaces. TCP offload performs most common TCP/IP functions inside the network interface hardware, significantly reducing the amount of host CPU resources that need to be devoted to processing networking traffic.

Note If the underlying network adapter supports TCP Segmentation Offload and the feature is enabled, FTP and other network-oriented protocols might be able to move a 64-KB block from one location to another using a single interrupt.

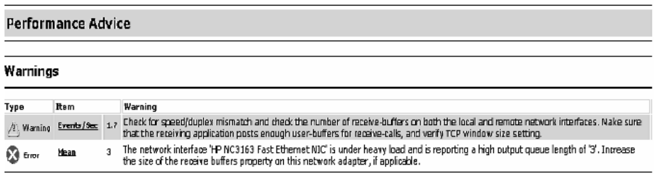
Another strategy for improving the performance of local area networking is to shift the workload for disk-to-tape backup of network clients to direct-attached or SAN-attached devices. Such “LAN-free” backups benefit from the use of the Fibre Channel protocol, which was specifically designed to optimize I/O device throughput. Compared to TCP/IP, Fibre Channel performs packet defragmentation and reassembly in a hardware layer to reduce the load on the host computer. A network-oriented protocol like FTP or SMB requires 45 Ethernet packets to move a 64 KB chunk of a file, with each packet interrupting the host machine so that it can be processed by the TCP/IP software layer in Windows Server 2003. Note that considerably fewer interrupts will be needed if the network adapter uses Interrupt Moderation. A single SCSI command issued over a Fibre Channel link can accomplish the same result with only one host interrupt required.

Server Performance Advisor System Network Diagnosis

Server Performance Advisor (SPA) is a server performance diagnostic tool developed to diagnose root causes of performance problems in a Microsoft® Windows Server™ 2003 operating system, including performance problems for network-oriented Windows components like Internet Information Services (IIS) 6.0 and the Active Directory® directory service. Server Performance Advisor measures the performance and use of resources by your computer to report on the parts that are stressed under your workload. The Server Performance Advisor can be downloaded from Microsoft at http://www.microsoft.com/downloads/details.aspx?FamilyID=61a41d78-e4aa-47b9-901b-cf85da075a73&displaylang=en and is an outstanding diagnostic tool for identifying network-related problems and for capacity planning.

SPA is a tool that alerts the user when a potential problem needs further investigation. In some instances, SPA offers suggestions for how to improve performance and resolve some of the issues identified with your workload and configuration. By using the networking counters and learning the details of the underlying network configuration and the network adapters used on the host system, the user gains a better understanding of the system behavior and how it can be tuned for optimal performance. Some examples of the kind of detail a user gets from SPA include offload features on the network adapter and whether the adapter is able to keep up with the processing of incoming and outgoing traffic. Users also get information about key TCP registry settings like *MaxUserPort* and *TcpWindowSize* and whether they might be affecting their performance.

Figure 5-40 is a snapshot that shows what SPA has to offer. The figure shows that the user got an alert regarding the *Output Queue Length* parameter, which normally means that the network adapter is low on buffer resources and might start dropping packets. This would lead to retransmits on the link and reduce overall network performance. Most network adapters offer the option to increase their Receive Buffers to values higher than their installation defaults.



Performance Advice

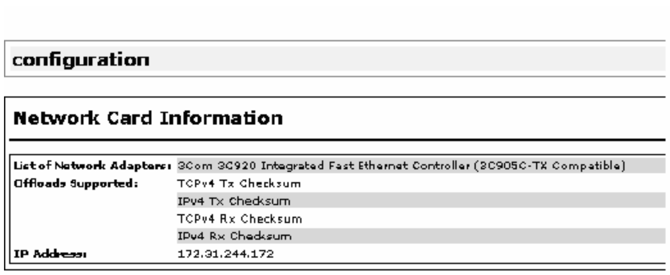
Warnings

Type	Item		Warning
Warning	Events / Sec	1.7	Check for speed/duplex mismatch and check the number of receive-buffers on both the local and remote network interfaces. Make sure that the receiving application posts enough user-buffers for receive-calls, and verify TCP window size setting.
Error	Mean	3	The network interface 'HP NC3163 Fast Ethernet NIC' is under heavy load and is reporting a high output queue length of '3'. Increase the size of the receive buffers property on this network adapter, if applicable.

Figure 5-40 An alert regarding the Output Queue Length parameter

SPA also provides the user with insight about how interrupts are distributed across the processors and whether the network adapter has support for interrupt moderation. Interrupt moderation allows multiple packets to be sent and received within the context of a single interrupt issued. This reduces the overall host CPU consumption and improves overall system performance.

Figure 5-41 illustrates another SPA network performance diagnostic display. Here SPA displays the Network Offload information table for the network adapter in use. Offload features save host CPU cycles as more of the processing gets offloaded to the network adapter, freeing up resources on the host to do more. Web Servers, for example, that use network offload are able to handle more client Web page requests and send response messages quicker.



configuration

Network Card Information

List of Network Adapters:	3Com 3C920 Integrated Fast Ethernet Controller (3C905C-TX Compatible)
Offloads Supported:	TCPv4 Tx Checksum
	IPv4 Tx Checksum
	TCPv4 Rx Checksum
	IPv4 Rx Checksum
IP Address:	172.31.244.172

Figure 5-41 The network offload information table for the network adapter in use

Network Monitor Packet Traces

The Network Monitor is a built-in diagnostic tool that allows you to capture and view network packets. To install the Network Monitor, use the Add/Remove Windows Components function to add the Network Monitoring Tools. This installs both the Network Monitor program software and inserts the Network Monitor Driver into the NDIS layer of the networking stack. The Network Monitor Driver enables the Network Monitor to capture network packets as they are being processed. A more powerful version of the Network Monitor that lets you view all the traffic on a network segment at one time is provided with the Microsoft System Management Server. In a switched network, however, the version of the Network Monitor that is included with the base operating system is perfectly adequate, because the host NIC will see only those packets specifically addressed to it.

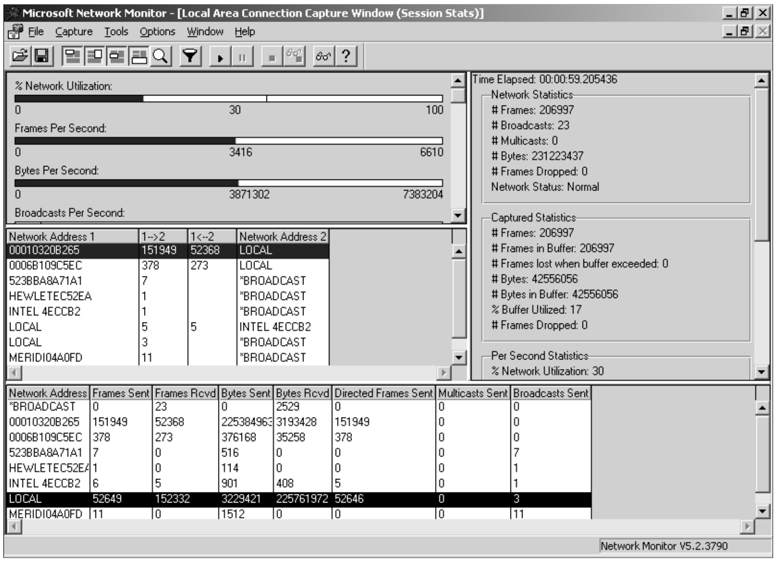
Using the Network Monitor effectively to solve networking performance problems requires an understanding of the network protocols that your machines use to communicate across the network. These networking protocols are standardized across the computer industry. If you understand how the major network protocols work from another context, you can apply that knowledge to the Windows Server 2003 environment. An outstanding reference work is *Microsoft Windows Server 2003 TCP/IP Protocols and Services: Technical Reference*. This book provides encyclopedic coverage of the networking protocols that Windows Server 2003 employs. The discussion here is intended to build on the basic information provided in that book.

The Network Monitor packet traces help resolve networking performance problems, and they are also an extremely useful tool for diagnosing a wide variety of other LAN problems that can impact performance, including:

- Extraneous network traffic associated with error detection and recovery from unreachable destinations and other configuration problems
- "Hidden" network traffic associated with authentication and security protocols
- Extra network traffic originating from hackers, worms, and viruses that have breached network security

Capturing network packets To begin a Network Monitor capture session, follow these steps:

1. Run the Network Monitor capture program, Netmon.exe. It is normally installed in %windir at \system32\netmon.
2. Configure a Network Capture session by selecting a Network and configuring the Network Monitor capture buffers using the Capture menu. If you configure a large number of memory-resident buffers, you will be able to capture packets efficiently, but you might steal RAM from other, active networking applications. If you are interested only in the performance-oriented information contained in packet headers, you can save on the RAM needed for a Capture session by restricting the amount of data you capture per packet.
3. Invoke the Start command from the Capture menu. While the trace session is active, the Station Stats window is shown, as illustrated in Figure 5-42.


Microsoft Network Monitor - [Local Area Connection Capture Window (Session Stats)]
File
Capture
Tools
Options
Window
Help
Network Monitor V5.2.3790

```
% Network Utilization:
0                30                100
Frames Per Second:
0                3416              6610
Bytes Per Second:
0                3871302           7383204
Broadcasts Per Second:
```

```
Time Elapsed: 00:00:59.205436
Network Statistics
# Frames: 206997
# Broadcasts: 23
# Multicasts: 0
# Bytes: 231223437
# Frames Dropped: 0
Network Status: Normal
Captured Statistics
# Frames: 206997
# Frames in Buffer: 206997
# Frames lost when buffer exceeded: 0
# Bytes: 42556056
# Bytes in Buffer: 42556056
% Buffer Utilized: 17
# Frames Dropped: 0
Per Second Statistics
% Network Utilization: 30
```

Network Address 1	1-->2	1<--2	Network Address 2
00010320B265	151949	52368	LOCAL
0006B109C5EC	378	273	LOCAL
523BBA8A71A1	7		*BROADCAST
HEWLETEC52EA	1		*BROADCAST
INTEL 4ECCB2	1		*BROADCAST
LOCAL	5	5	INTEL 4ECCB2
LOCAL	3		*BROADCAST
MERIDI04A0FD	11		*BROADCAST

Network Address	Frames Sent	Frames Rcvd	Bytes Sent	Bytes Rcvd	Directed Frames Sent	Multicasts Sent	Broadcasts Sent
*BROADCAST	0	23	0	2529	0	0	0
00010320B265	151949	52368	225384963	3193428	151949	0	0
0006B109C5EC	378	273	376168	35258	378	0	0
523BBA8A71A1	7	0	516	0	0	0	7
HEWLETEC52E	1	0	114	0	0	0	1
INTEL 4ECCB2	6	5	901	408	5	0	1
LOCAL	52649	152332	3229421	225761972	52646	0	3
MERIDI04A0FD	11	0	1512	0	0	0	11

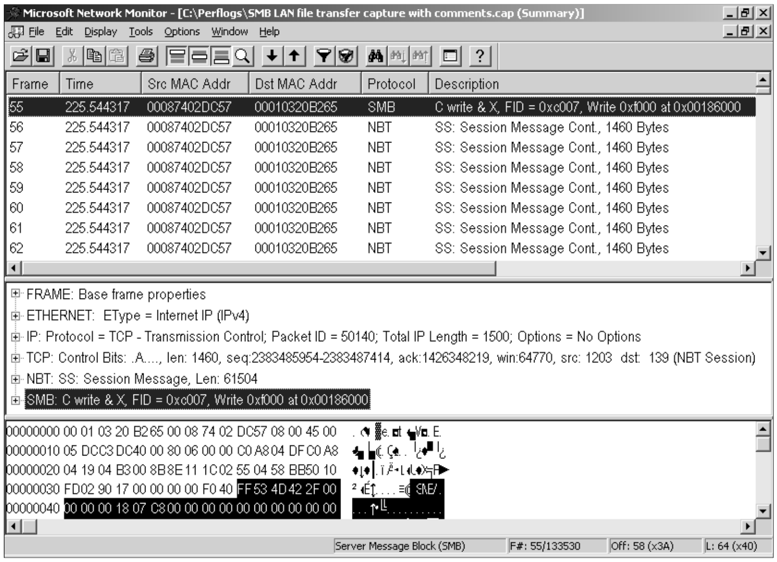
Figure 5-42 The Station Stats window that is shown while the trace session is active

4. Stop the capture session by using the Capture menu.
5. Select Save As from the File menu to save a packet trace to a .cap capture file for subsequent viewing and analysis.

Capture session stats are shown in four separate windows. At the upper left corner is a series of histograms that report on overall network traffic captured during the session. More detailed capture session statistics are provided in the right column. The session statistics reported in this column include the number of frames captured, the amount of RAM buffer space utilized, and the number of frames lost when the buffer overflowed. Some important Network Interface Card (NIC) statistics are also provided when you scroll down the right column display.

The left middle column shows the amount of network traffic (in packets) that was transmitted between any two addresses on the network. The bottom window displays the total activity associated with each network address. For convenience, you can maintain an Address database that will replace network addresses with names more suitable for people.

You can display captured data in the Station Stats window. An example, corresponding to the capture session illustrated in Figure 5-42, is shown in Figure 5-43.


Microsoft Network Monitor - [C:\Perflogs\SMB LAN file transfer capture with comments.cap (Summary)]
File Edit Display Tools Options Window Help

```
Frame  Time        Src MAC Addr  Dst MAC Addr  Protocol  Description
55     225.544317  00087402DC57  00010320B265  SMB       C write & X, FID = 0xc007, Write 0xf000 at 0x00186000
56     225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
57     225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
58     225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
59     225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
60     225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
61     225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
62     225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
```

```
+ FRAME: Base frame properties
+ ETHERNET:  EType = Internet IP (IPv4)
+ IP: Protocol = TCP - Transmission Control; Packet ID = 50140; Total IP Length = 1500; Options = No Options
+ TCP: Control Bits: .A...., len: 1460, seq:2383485954-2383487414, ack:1426348219, win:64770, src: 1203  dst:  139 (NBT Session)
+ NBT: SS: Session Message, Len: 61504
+ SMB: C write & X, FID = 0xc007, Write 0xf000 at 0x00186000
```

```
00000000 00 01 03 20 B2 65 00 08 74 02 DC 57 08 00 45 00
00000010 05 DC C3 DC 40 00 80 06 00 00 C0 A8 04 DF C0 A8
00000020 04 19 04 B3 00 8B 8E 11 1C 02 55 04 58 BB 50 10
00000030 FD 02 90 17 00 00 00 00 F0 40 FF 53 4D 42 2F 00
00000040 00 00 00 18 07 C8 00 00 00 00 00 00 00 00 00 00
```

Figure 5-43 A display of captured data from the Station Stats window

This example begins on frame number 55, which is an SMB write command issued as part of a file copy from the local server to a backup server on the same network segment. The packet display is divided into three segments. The top segment shows the sequence of packets (or frames) and includes a description of the contents of each packet that were captured. The Time field, relative to the start of the capture session, is given in microseconds. The highest-level protocol present in the frame, which determines the packet's ultimate destination application, is also shown. You can scroll the top window until you find frames of interest. Double-click to zoom in the frame sequence window, and double-click again to toggle back to the three-panel display.

Tip On a busy network with many concurrent networking sessions in progress, the packets from each of these sessions are interleaved, presenting quite a jumble of activity on the Network Monitor capture display. Use filters to eliminate packets from the display that are not germane to the current problem. You can define filters based on specific protocols, protocol header fields, or other networking properties.

The middle panel decodes the packet headers associated with the frame highlighted in the top window. At the highest level are the Frame properties. These include the time of capture and, if configured, the time difference in microseconds from the previous frame captured in the sequence.

Important The *Frame* property that calculates the time delta from the previous frame is often the Round Trip Time network latency metric that is of prime importance in network performance and tuning. If the current frame is a reply to the previous frame, the time delta is the Round Trip Time.

This example shows five levels of protocol headers that are all decoded, including these three:

- **Ethernet packet header fields** These show the source and destination network MAC addresses.
- **IP header fields** These include the source and destination network IP addresses and the TTL value that is interpreted as the hop count.
- **TCP header fields** These include source and destination Ports, *Sequence Number*, *Acknowledgement Number*, Flag bits, the *AdvertisedWindow* size, and Options.

Because this packet is an SMB Write command, NBT and SMB headers are also included in this display. The networking application protocols that the Network Monitor can decode for you include HTTP, FTP, SMTP, DNS, ICMP, DHCP, ARP, and MS RPC.

Click on a plus sign (+) to expand the display and see more information about individual header fields.

The bottom panel displays the byte stream associated with each packet that was captured. It is displayed in parallel in hex and in ASCII. The portion of the packet header decoded in the middle panel is highlighted in reverse video in the bottom panel data display.

LAN Tuning

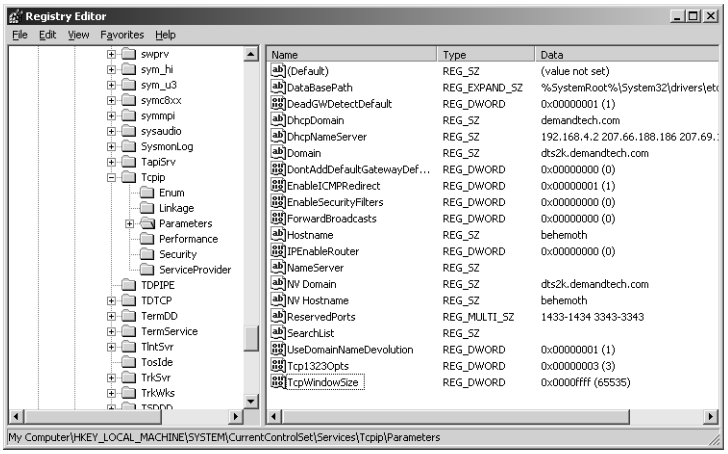
Because latency—the amount of time it takes to deliver and confirm the delivery of network packets—is minimal in a LAN, the important configuration and tuning parameters associated with TCP session control have limited impact on overall performance. You can observe these session control mechanisms in operation, but tweaking the parameters themselves is usually of little benefit on most LANs, except in extreme cases.

The tuning parameters that can have the most influence on LAN performance are the ones related to TCP Flow Control and Congestion Control. For LANs, these include the TCP Advertised Window and Selective Acknowledgement (SACK).

TCP congestion control The TCP Host-to-Host layer of networking software is responsible for session control, ensuring that all packets transmitted by one host are received at their destination. TCP is also responsible for congestion control, which provides adaptive transmission mechanisms that allow the host computer to detect network congestion and adjust accordingly. TCP congestion control revolves around detecting two congestion signals:

- **Receiver congestion** A Window-full condition which forces the receiver to shrink its advertised congestion window. The sender must slow down its rate of data transmission to allow the receiver to catch up and process all the data the sender has sent. In general, this mechanism is designed to keep a fast sender from overloading a slower receiver.
- **Network congestion** TCP assumes that any packet loss results from congestion in the network—an overloaded network router or adapter on route to its destination dropped the packet. Accordingly, retransmission of an unacknowledged segment causes the sender to back off from its current transmission rates so that it does not contribute to further performance degradation.

A key aspect of TCP congestion control is the *AdvertisedWindow* parameter. The Advertised Window represents the maximum number of bytes per connection a sender can transmit at any one time without receiving a corresponding Acknowledgement reply stating the transmission was successful. Once the Advertised Window is full, the sender must wait. The default Advertised Window for a Windows Server 2003–based machine that is running 100-MB Ethernet is about 17 KB. (For 1000-MB Ethernet, the default Advertised Window on Windows Server 2003 is 64 KB.) The default TCP Advertised Window can be changed by adding the TcpWindowSize registry key to HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters, as illustrated in Figure 5-44.



Name	Type	Data
(Default)	REG_SZ	(value not set)
DataBasePath	REG_EXPAND_SZ	%SystemRoot%\System32\drivers\etc
DeadGWDetectDefault	REG_DWORD	0x00000001 (1)
DhcpDomain	REG_SZ	demandtech.com
DhcpNameServer	REG_SZ	192.168.4.2 207.66.188.186 207.69.
Domain	REG_SZ	dts2k.demandtech.com
DontAddDefaultGatewayDef…	REG_DWORD	0x00000000 (0)
EnableICMPRedirect	REG_DWORD	0x00000001 (1)
EnableSecurityFilters	REG_DWORD	0x00000000 (0)
ForwardBroadcasts	REG_DWORD	0x00000000 (0)
Hostname	REG_SZ	behemoth
IPEnableRouter	REG_DWORD	0x00000000 (0)
NameServer	REG_SZ	
NV Domain	REG_SZ	dts2k.demandtech.com
NV Hostname	REG_SZ	behemoth
ReservedPorts	REG_MULTI_SZ	1433-1434 3343-3343
SearchList	REG_SZ	
UseDomainNameDevolution	REG_DWORD	0x00000001 (1)
Tcp1323Opts	REG_DWORD	0x00000003 (3)
TcpWindowSize	REG_DWORD	0x0000ffff (65535)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

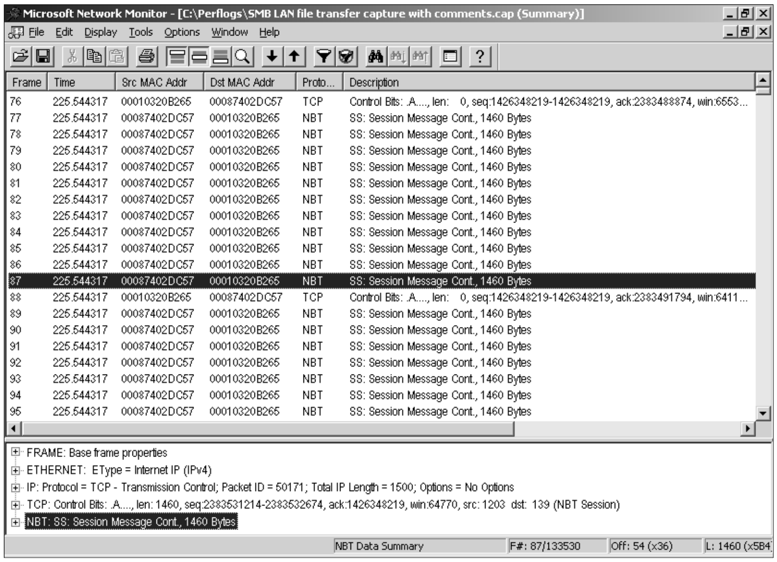
Figure 5-44 Changing the default TCP Advertised Window

In this example, TcpWindowSize is set to 65,535, the largest value that will fit in the TCP header's 16-bit *Advertised Window* field. Tcp1323Opts are also coded in this example to enable Window scaling to support Advertised Window sizes greater than 64 KB. Using Windows scaling, Advertised Windows as large as 1 GB can be specified.

Some additional TCP tuning parameters that complement having a large Advertised Window are also enabled in this example. Tcp1323Opts was coded here to enable the Timestamps option. The Timestamps option offers a significantly more accurate mechanism for TCP to use to calculate network round trip time (RTT), a key element in TCP's decision concerning when to retransmit an unacknowledged packet. If TCP waits too long to retransmit an unacknowledged packet, application response time suffers. On the other hand, if TCP is too quick to retransmit an unacknowledged packet, this speed can further contribute to the network congestion that caused the initial packet to be dropped. With the Timestamps option enabled, TCP uses the round trip time of a packet and its associated acknowledgement packet to maintain a current value for response time. The observed RTT is used to calculate a suitable retransmission time-out value per session. TCP uses a smoothing function applied to the session's response time history that gives greater weight to the most recent values of response time. This allows TCP to adapt to changing network conditions quickly, for example, when a router goes offline, and a longer path is required to deliver packets to the same destination.

The *SackOpts* parameter is also coded explicitly in this example, although this explicit coding is not necessary because SACK (Selective Acknowledgement) is enabled by default in Windows Server 2003. With larger Advertised Windows, it is important to enable SACK so that the sender does not have to retransmit an entire window's worth of data simply because one packet was lost.

LAN tuning example: disk backup The example in Figure 5-45 illustrates how these TCP congestion control mechanisms operate in the context of a disk-to-tape backup operation that can be expected to push the limits of LAN throughput capacity. Because the receiver has an Advertised Window of 64 KB, the sender fires back one packet after another, beginning at Frame 55, to execute the SMB Write command. Using Slow Start, the sender increases the number of packets it sends at a time, up to the full size of the Advertised Window. At this point in the session, the full 64-KB Receive Window is available, so the sender starts sending messages without bothering to wait for a reply. In fact, 20 consecutive messages are transmitted before an ACK from the first send is received at Frame 76. The first reply message received acknowledges receipt of Frame 56, so the Advertised Window is still not full.


Microsoft Network Monitor - [C:\Perflogs\SMB LAN file transfer capture with comments.cap (Summary)]
File Edit Display Tools Options Window Help

```
Frame  Time        Src MAC Addr   Dst MAC Addr   Proto...  Description
76     225.544317  00010320B265   00087402DC57   TCP       Control Bits: .A...., len:    0, seq:1426348219-1426348219, ack:2383488874, win:6553...
77     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
78     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
79     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
80     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
81     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
82     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
83     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
84     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
85     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
86     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
87     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
88     225.544317  00010320B265   00087402DC57   TCP       Control Bits: .A...., len:    0, seq:1426348219-1426348219, ack:2383491794, win:6411...
89     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
90     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
91     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
92     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
93     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
94     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes
95     225.544317  00087402DC57   00010320B265   NBT       SS: Session Message Cont., 1460 Bytes

+ FRAME: Base frame properties
+ ETHERNET:  EType = Internet IP (IPv4)
+ IP: Protocol = TCP - Transmission Control; Packet ID = 50171; Total IP Length = 1500; Options = No Options
+ TCP: Control Bits: .A...., len: 1460, seq:2383531214-2383532674, ack:1426348219, win:64770, src: 1203  dst:  139 (NBT Session)
+ NBT: SS: Session Message Cont., 1460 Bytes

NBT Data Summary    F#: 87/133530    Off: 54 (x36)    L: 1460 (x5B4
```

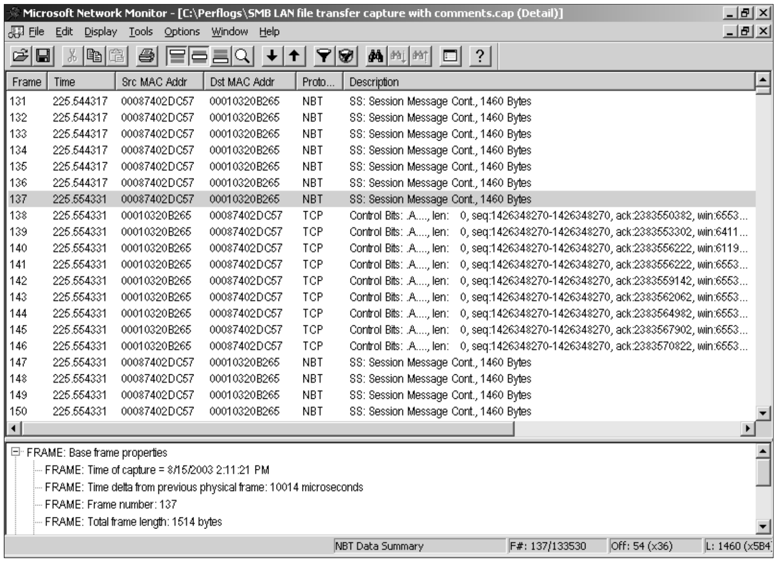
Figure 5-45 TCP congestion control mechanisms in the context of a disk-to-tape backup operation that can be expected to push the limits of LAN throughput capacity

At Frame 88, another ACK is received, this time explicitly acknowledging receipt of Frames 57 and 58. At this point, the sender has transmitted 27 segments that have still not been acknowledged. But the 64-KB Advertised Window is still not full, so the sender can continue to transmit. Frames 89–99 are then transmitted, which completes the SMB Write command. The last frame in the sequence contains only 188 bytes of data to complete the sequence of sending a 60-KB block of data over the wire. It takes 43 Ethernet packets to transmit this block of data from one machine to another. This helps to explain why TCP/IP LANs are so much less efficient than direct attached Storage Area Networks at moving large blocks of data around. A single SCSI disk command can write a 60-KB block to disk and requires servicing only a single device interrupt. On an Ethernet LAN, the same amount of data transfer requires servicing multiple Network Interface Card interrupts. (If the NIC does not support interrupt moderation, all 43 interrupts might need to be serviced.) This should also help to explain why LANs are so hungry for bandwidth during disk backup and other throughput-oriented operations.

Larger TCP Advertised Windows allow two TCP peers much greater latitude to send and receive data on a LAN segment without one machine having to slow down and wait for the other so frequently. Receivers do not have to send an ACK for every packet. Acknowledgements are cumulative, so one ACK packet can acknowledge receipt of many packets. With SACK, the ACK message can even indicate which specific packets in the byte sequence have not been received yet so that the entire sequence does have to be retransmitted.

In Frame 137 (Figure 5-46), the sender finally slows down. Perhaps it was necessary to read the local disk to get the next block of data to send. Frame 137 is transmitted 10.014 milliseconds after the previous frame. This gives the receiver an opportunity to catch up with the sender—a sequence of ACK messages follow in Frames 138–146.

In tuning a LAN to achieve higher throughput levels, one goal is to increase the average size of packets that are transmitted. In TCP, every packet transmitted must be acknowledged. In a poorly tuned LAN, you will observe that every third or fourth packet is an ACK. Because ACKs are usually sent without a data payload, ACKs are minimum-sized TCP packets, normally no bigger than 50–60 bytes, depending on how many options are coded. A full Ethernet packet is 1514 bytes. If every third packet is an ACK, the average packet size is (2×1514+60)/3, or 1029 bytes, substantially reducing the effective capacity of the link.


Microsoft Network Monitor - [C:\Perflogs\SMB LAN file transfer capture with comments.cap (Detail)]
File Edit Display Tools Options Window Help

```
Frame  Time        Src MAC Addr  Dst MAC Addr  Proto...  Description
131    225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
132    225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
133    225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
134    225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
135    225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
136    225.544317  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
137    225.554331  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
138    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383550382, win:6553...
139    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383553302, win:6411...
140    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383556222, win:6119...
141    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383556222, win:6553...
142    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383559142, win:6553...
143    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383562062, win:6553...
144    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383564982, win:6553...
145    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383567902, win:6553...
146    225.554331  00010320B265  00087402DC57  TCP       Control Bits: .A...., len:   0, seq:1426348270-1426348270, ack:2383570822, win:6553...
147    225.554331  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
148    225.554331  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
149    225.554331  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes
150    225.554331  00087402DC57  00010320B265  NBT       SS: Session Message Cont., 1460 Bytes

FRAME: Base frame properties
    FRAME: Time of capture = 8/15/2003 2:11:21 PM
    FRAME: Time delta from previous physical frame: 10014 microseconds
    FRAME: Frame number: 137
    FRAME: Total frame length: 1514 bytes

NBT Data Summary    F#: 137/133530    Off: 54 (x36)    L: 1460 (x5B4
```

Figure 5-46 A sequence of ACK messages follow Frame 137

This example demonstrates how a proper setting for the *Advertised Window* can help boost the effective network capacity by 10–20 percent. Returning to the overall capture session statistics in Figure 5-42, you can see that the average of number of frames per second is about 3400, with more than 1000 bytes per frame. The next opportunity for tuning this disk-to-disk backup operation is to initiate a second backup stream that can run parallel with the first. The second stream can be transmitting data during the slack time that begins at Frame 137. Conceivably, a second parallel backup stream could increase throughput over the link by a factor of 2 since there is ample line capacity.

WAN Performance

Whereas bandwidth considerations dominate the performance of low latency LANs, WAN performance is dominated by latency—the time it takes to send a message to a distant station and receive an acknowledgement in reply. The TCP flow control mechanism allows a sender to fill an Advertised Window with data, but then the sender must wait for an acknowledgement message in reply. The time it takes to send a packet and receive an ACK in reply is known as the round trip time, or RTT.

This section discusses the impact of round trip time on WAN throughput and capacity. It shows how to calculate the RTT of a remote TCP session using the Network Monitor and the Tracert utility. It also shows how to use the RTT to determine the effective capacity of a link that will be used for long-haul transmissions. This is useful when you need to size a link that is going to be used to transfer large amounts of data regularly between two points on your network that are separated by a large distance.

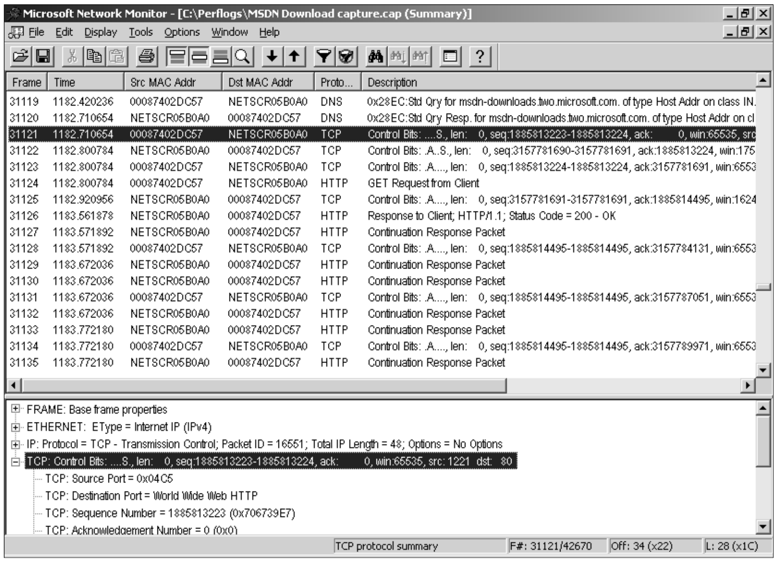
Round Trip Time

Round trip time (RTT) is the time it takes to send a message and receive an Acknowledgement packet back from the receiver in reply. Together with the Advertised Window size, RTT establishes an upper limit to the effective throughput of a TCP session. If the RTT of the session is 100 ms for a long distance connection, then the TCP session will be able to send a maximum of only 1/RTT windows' worth of data per second, in this case, just 10 windows per second. RTT effectively limits the maximum throughput you can expect on that connection to be:

Max throughput = AdvertisedWindow RTT

RTT is a crucial factor if you are planning to provide a networking capability that needs to move large amounts of data across relatively long distances. For planning purposes, you can use ping, the Tracert utility, or the Network Monitor to observe the actual RTT of a TCP connection.

Figure 5-47 shows a SYN packet being sent from a Windows Server 2003–based machine to establish a TCP connection with a remote machine that happens to be a Web server at http://www.msdn.microsoft.com, the Microsoft Developer Network Web site. Note the time of the transmission for Frame 31121: 1182.710654. The sender requests a session on Port 80, which is the HTTP socket for Web servers, and advertises a TCP window of 64 KB.


Microsoft Network Monitor - [C:\Perflogs\MSDN Download capture.cap (Summary)]
File Edit Display Tools Options Window Help

```
Frame  Time         Src MAC Addr  Dst MAC Addr  Proto...  Description
31119  1182.420236  00087402DC57  NETSCR05B0A0  DNS   0x28EC:Std Qry for msdn-downloads.two.microsoft.com. of type Host Addr on class IN.
31120  1182.710654  NETSCR05B0A0  00087402DC57  DNS   0x28EC:Std Qry Resp. for msdn-downloads.two.microsoft.com. of type Host Addr on cl
31121  1182.710654  00087402DC57  NETSCR05B0A0  TCP   Control Bits: ....S., len:   0, seq:1885813223-1885813224, ack:        0, win:65535, src
31122  1182.800784  NETSCR05B0A0  00087402DC57  TCP   Control Bits: .A..S., len:   0, seq:3157781690-3157781691, ack:1885813224, win:175
31123  1182.800784  00087402DC57  NETSCR05B0A0  TCP   Control Bits: .A...., len:   0, seq:1885813224-1885813224, ack:3157781691, win:6553
31124  1182.800784  00087402DC57  NETSCR05B0A0  HTTP  GET Request from Client
31125  1182.920956  NETSCR05B0A0  00087402DC57  TCP   Control Bits: .A...., len:   0, seq:3157781691-3157781691, ack:1885814495, win:1624
31126  1183.561878  NETSCR05B0A0  00087402DC57  HTTP  Response to Client; HTTP/1.1; Status Code = 200 - OK
31127  1183.571892  NETSCR05B0A0  00087402DC57  HTTP  Continuation Response Packet
31128  1183.571892  00087402DC57  NETSCR05B0A0  TCP   Control Bits: .A...., len:   0, seq:1885814495-1885814495, ack:3157784131, win:6553
31129  1183.672036  NETSCR05B0A0  00087402DC57  HTTP  Continuation Response Packet
31130  1183.672036  NETSCR05B0A0  00087402DC57  HTTP  Continuation Response Packet
31131  1183.672036  00087402DC57  NETSCR05B0A0  TCP   Control Bits: .A...., len:   0, seq:1885814495-1885814495, ack:3157787051, win:6553
31132  1183.672036  NETSCR05B0A0  00087402DC57  HTTP  Continuation Response Packet
31133  1183.772180  NETSCR05B0A0  00087402DC57  HTTP  Continuation Response Packet
31134  1183.772180  00087402DC57  NETSCR05B0A0  TCP   Control Bits: .A...., len:   0, seq:1885814495-1885814495, ack:3157789971, win:6553
31135  1183.772180  NETSCR05B0A0  00087402DC57  HTTP  Continuation Response Packet

+ FRAME: Base frame properties
+ ETHERNET:  EType = Internet IP (IPv4)
+ IP: Protocol = TCP - Transmission Control; Packet ID = 16551; Total IP Length = 48; Options = No Options
- TCP: Control Bits: ....S., len:   0, seq:1885813223-1885813224, ack:        0, win:65535, src: 1221  dst:  80
    TCP: Source Port = 0x04C5
    TCP: Destination Port = World Wide Web HTTP
    TCP: Sequence Number = 1885813223 (0x706739E7)
    TCP: Acknowledgement Number = 0 (0x0)

TCP protocol summary    F#: 31121/42670    Off: 34 (x22)    L: 28 (x1C)
```

Figure 5-47 A SYN packet sent from a Windows Server 2003–based machine to establish a TCP connection with a remote machine

Figure 5-47 shows the SYN-ACK reply from the remote location. The Web server reply is received at 1182.800784, 90 milliseconds later. This RTT of the initial SYN, SYN-ACK sequence is not used by TCP in any way. Subsequently, TCP samples the actual RTT of a session once per *Advertised Window*, or, if the Timestamps option is enabled, is able to measure the RTT for each ACK message returned. In this instance, RTT was observed to be 90 ms.

It is a good idea to gather more than one sample of the RTT. Tracing a few of the subsequent Send-ACK sequences in the Network Monitor capture will normally do the trick. Returning to Figure 5-47, an HTTP GET Request to the Web server is issued at Frame 31124. Frame 31125 is an ACK response to Frame 31124, received some 120 ms. later. Frame 31126, the beginning of an HTTP Response Message generated by the Web server, is received 640 milliseconds following the ACK. Interestingly, this 640 ms includes the Web server application response time.

Using Tracert to measure RTT The Tracert utility can also be used to observe and measure the RTT between two IP addresses. With Tracert, as opposed to ping, you can see the route details for representative packets dispatched from the sender to the destination. Compared to a Network Monitor packet trace, the RTT that Tracert calculates ignores any server application response time. In that sense, using Tracert is similar to using the RTT of the initial SYN, SYN-ACK sequence, because no application layer processing is involved in that exchange either.

The Web server's IP address that Tracert is attempting to contact has an IP address of 207.46.196.115, according to a Network Monitor trace. An additional complication in this case is that this IP address is likely to be a virtual IP address associated with a Web server cluster, which might lead to some additional variability in the RTT values you can observe.

The RTT for a long distance request is a function of both the distance between the source and destination stations and the hop count. The TTL field in the IP header for the SYN-ACK is 47. TTL is decremented by each router on the path to the final destination. However, a TTL value can be difficult to interpret because you do not know the starting TTL value used by the machine. The Windows Server 2003 default for a starting TTL value is 128. It seems unlikely that the Web server returning the SYN-ACK is 81 hops away. A TTL starting value of 64 is more likely, suggesting that a packet on the connection makes 17 hops on the way to its final destination. You can verify the number of hops in the route using the Tracert utility, as illustrated in Listing 5-11.

Listing 5-11 Using the Tracert Utility

```
C:\>tracert 207.46.196.115

Tracing route to 207.46.196.115 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  192.168.0.1
  2     1 ms     1 ms     1 ms  192.168.1.1
  3    51 ms    49 ms    50 ms  user1.net295.fl.sprint-hsd.net [209.26.27.1]
  4    65 ms    58 ms    57 ms  user109.net590.fl.sprint-hsd.net [65.41.11.109]
  5    60 ms    58 ms    56 ms  209.26.245.33
  6    58 ms    55 ms    58 ms  209.26.245.18
  7    57 ms    59 ms    57 ms  sl-gw11-orl-10-2.sprintlink.net [160.81.34.73]
  8    62 ms    63 ms    60 ms  sl-bb20-orl-0-0.sprintlink.net [144.232.2.232]
  9    80 ms    80 ms    77 ms  sl-bb21-atl-10-2.sprintlink.net [144.232.19.169]
 10    93 ms    88 ms    92 ms  sl-bb21-chi-11-0.sprintlink.net [144.232.18.33]
 11    92 ms    90 ms    89 ms  sl-bb25-chi-13-0.sprintlink.net [144.232.26.90]
 12   133 ms   133 ms   134 ms  sl-bb21-sea-1-0.sprintlink.net [144.232.20.156]
 13   133 ms   136 ms   132 ms  sl-bb21-sea-15-0.sprintlink.net [144.232.6.89]
 14   134 ms   146 ms   137 ms  sl-microsoft-23-0.sprintlink.net [144.224.113.146]
 15   137 ms   184 ms   140 ms  pos0-0.core2.sea2.us.msn.net [207.46.33.185]
 16   139 ms   137 ms   137 ms  207.46.33.237
 17   137 ms   136 ms   139 ms  207.46.36.78
 18   133 ms   134 ms   138 ms  207.46.155.21
 19     *        *        *     Request timed out.
 20     *        *        *     Request timed out.
 21  ^C
```

Because the destination is a virtual address, Tracert will likely not succeed in finding the complete route. At Hop 15, Tracert succeeds in locating an address on the Microsoft core network backbone. From there it should be only one or two hops to the final destination, although Tracert is unable to locate it, and eventually requests start to time out. Seventeen hops to the destination Web server seems about right. The approximately 140 ms RTT that Tracert reports for the route is also consistent with the Network Monitor capture.

Effective capacity of a WAN link The requirement to wait for an explicit acknowledgment message before sending the next block of data constrains the performance of a TCP/IP session that is established over a Wide Area Network (WAN). The time it takes to send a message to a remote computer and receive an acknowledgement message, conveniently thought of as round trip time, affects the effective capacity of the link. In WAN performance, the effective link capacity is reduced to the number of trips/sec that the link can support. Because the number of trips per second = 1/RTT, and a maximum of AdvertisedWindow bytes can be transmitted once per RTT, effective capacity is reduced as follows:

Effective session capacity = AdvertisedWindow RTT

If, for example, the RTT of a link is 20 milliseconds, the sender can transmit only 50 *Advertised Windows* full of per second. Assuming an *Advertised Window* of 17520 Bytes—the Windows Server 2003 default, which is 16K rounded up to twelve 1460-byte segments—the effective capacity of a WAN connection is this:

```
876 KB/sec = 17520  0.020
```

To facilitate bulk data transfers over long distances, using large Advertised Window values and SACK might be necessary. In the previous example, a 64-KB Advertised Window improves the effective capacity of the link to 3200 KB/sec.

Be advised that this formula is only a first-order approximation. Because of the manner in which the TCP Congestion Window functions, the effective capacity of a connection over a remote link is apt to be even less.

Congestion Window TCP Congestion Control establishes a Congestion Window for each session. The Congestion Window ignores the receiver's *AdvertisedWindow* at the start of a session. It uses *Slow Start* instead, initially sending only two segments and then waiting for an ACK before sending more data. As part of congestion control function discussed briefly in Chapter 1, "Performance Monitoring Overview," a TCP sender paces its rate of data transmission, slowly increasing it until a congestion signal is recognized. TCP recognizes two congestion signals for a connection:

- **Receiver congestion** Recognized whenever TCP must delay transmission of the next segment because the receiver's Advertised Window is full
- **Network congestion** Recognized whenever TCP must retransmit a segment

When a congestion signal is received, the TCP sender backs off sharply, either cutting its current send window in half or moving back into slow start, depending on the type of network congestion. This is known as *Additive increase/Multiplicative decrease* to open and close the send window. TCP starts a session by sending two packets at a time and waiting for an ACK. TCP slowly increases its connection send window one packet at a time until it receives a congestion signal. When it recognizes a congestion signal, TCP cuts the current send window in half (for the most common type of congestion signal) and then resumes additive increase. Additive increase widens the congestion window one packet at a time until the Congestion Window equals the *AdvertisedWindow*, the *AdvertisedWindow* is full, or a network congestion signal is received. The operation of these two congestion control mechanisms produces a send window that tends to oscillate, as illustrated in Figure 5-48, reducing the effective capacity of a TCP connection.

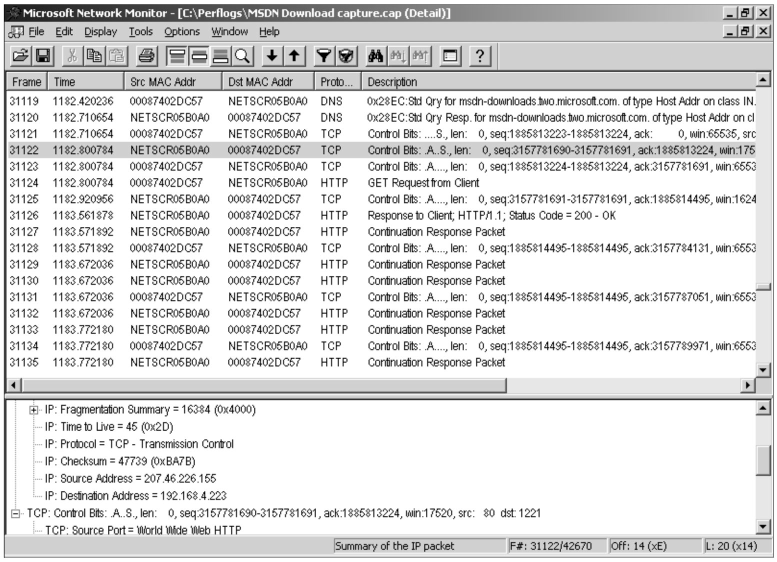

```
Microsoft Network Monitor - [C:\Perflogs\MSDN Download capture.cap (Detail)]

Frame  Time         Src MAC Addr   Dst MAC Addr   Proto...  Description
31119  1182.420236  00087402DC57   NETSCR05B0A0   DNS       0x28EC:Std Qry for msdn-downloads.two.microsoft.com. of type Host Addr on class IN.
31120  1182.710654  NETSCR05B0A0   00087402DC57   DNS       0x28EC:Std Qry Resp. for msdn-downloads.two.microsoft.com. of type Host Addr on cl
31121  1182.710654  00087402DC57   NETSCR05B0A0   TCP       Control Bits: ....S., len:   0, seq:1885813223-1885813224, ack:         0, win:65535, src
31122  1182.800784  NETSCR05B0A0   00087402DC57   TCP       Control Bits: .A..S., len:   0, seq:3157781690-3157781691, ack:1885813224, win:175
31123  1182.800784  00087402DC57   NETSCR05B0A0   TCP       Control Bits: .A...., len:   0, seq:1885813224-1885813224, ack:3157781691, win:6553
31124  1182.800784  00087402DC57   NETSCR05B0A0   HTTP      GET Request from Client
31125  1182.920956  NETSCR05B0A0   00087402DC57   TCP       Control Bits: .A...., len:   0, seq:3157781691-3157781691, ack:1885814495, win:1624
31126  1183.561878  NETSCR05B0A0   00087402DC57   HTTP      Response to Client; HTTP/1.1; Status Code = 200 - OK
31127  1183.571892  NETSCR05B0A0   00087402DC57   HTTP      Continuation Response Packet
31128  1183.571892  00087402DC57   NETSCR05B0A0   TCP       Control Bits: .A...., len:   0, seq:1885814495-1885814495, ack:3157784131, win:6553
31129  1183.672036  NETSCR05B0A0   00087402DC57   HTTP      Continuation Response Packet
31130  1183.672036  NETSCR05B0A0   00087402DC57   HTTP      Continuation Response Packet
31131  1183.672036  00087402DC57   NETSCR05B0A0   TCP       Control Bits: .A...., len:   0, seq:1885814495-1885814495, ack:3157787051, win:6553
31132  1183.672036  NETSCR05B0A0   00087402DC57   HTTP      Continuation Response Packet
31133  1183.772180  NETSCR05B0A0   00087402DC57   HTTP      Continuation Response Packet
31134  1183.772180  00087402DC57   NETSCR05B0A0   TCP       Control Bits: .A...., len:   0, seq:1885814495-1885814495, ack:3157789971, win:6553
31135  1183.772180  NETSCR05B0A0   00087402DC57   HTTP      Continuation Response Packet

   + IP: Fragmentation Summary = 16384 (0x4000)
     IP: Time to Live = 45 (0x2D)
     IP: Protocol = TCP - Transmission Control
     IP: Checksum = 47739 (0xBA7B)
     IP: Source Address = 207.46.226.155
     IP: Destination Address = 192.168.4.223
 - TCP: Control Bits: .A..S., len:   0, seq:3157781690-3157781691, ack:1885813224, win:17520, src:  80  dst: 1221
     TCP: Source Port = World Wide Web HTTP

Summary of the IP packet    F#: 31122/42670    Off: 14 (xE)    L: 20 (x14)
```

Figure 5-48 A send window that tends to oscillate

An approximation of the effective capacity of a WAN link that accounts for the Additive increase/Multiplicative decrease algorithm, which controls the size of the TCP *Congestion Window*, is:

$$\textit{Effective session capacity} = (\textit{AdvertisedWindow} / \textit{RTT}) \times 1/\sqrt{p}$$

where *p* is the probability of the session receiving a congestion signal as a percentage of all packets transmitted. The overall effect of the Additive increase/Multiplicative decrease algorithm normally is to reduce the effective capacity of a WAN link by another 25 percent.

To maximize network throughput over a WAN route, it is important to keep *p*, the probability of receiving a congestion signal, low. Scanning through a large packet trace looking for congestion signals—transmission delays caused by an *Advertised Window* full condition or a TCP Retransmission following a time out—can be quite tedious, however. The TCP\Segments Retransmitted/sec counter is very useful in this context. Almost any nonzero values of the TCP\Segments Retransmitted/sec counter warrant investigation, especially on machines where bulk data transmissions are occurring.

Large values for the TCP *Advertised Window* serve to diminish the number of *Advertised Window* full congestion signals that are received.

Adaptive Retransmission

The calculation of round trip time by the TCP layer is an important component of a mechanism known as *adaptive retransmission*, which also looms large in WAN performance. The measured RTT of the connection is used to calculate a Retransmission Timeout (RTO) value that partially governs the error recovery behavior of each TCP session.

The only means of error recovery available in TCP is for a sender to retransmit datagrams that were not received correctly, either because of data transmission errors or IP packet loss. The sender compares the Acknowledgement sequence number with the sequence number of the next packet in the transmission stream. If the Acknowledgement sequence number is less than the sender's current SequenceNumber, at least one packet is missing. The sender cannot tell what happened to the packet—only that no Acknowledgement was received. If the sender does not perform fast retransmission (discussed later), a transmission timer will fire, assuming that something probably has gone wrong.

Here is what could have happened to the unacknowledged packet:

- A router or NIC card detected a damaged packet being sent and dropped it.
- A router was forced to drop a perfectly good packet on its way to its destination because of capacity constraints.
- The ACK packet was damaged in transit and was dropped.
- An overloaded router dropped the ACK packet because of network congestion.

It is not possible for a sender to distinguish between these possibilities. TCP's only recourse is to resend the unacknowledged packet, after delaying long enough to ensure that an ACK is not forthcoming. Because the WAN technologies in use across the Internet have become so reliable, fewer and fewer packets are damaged during transmission. At the time of writing, the main reason unacknowledged packets need to be retransmitted is that an overloaded router or overloaded network adapter somewhere along the route dropped them. However, there are exceptions to that general rule for two types of network traffic where transmission errors are more common. These two types of network traffic are satellite transmissions and data sent via wireless links.

Tip The TCP\Retransmitted Segments/sec counter reports the rate of retransmissions. Almost any nonzero values of the TCP\Segments Retransmitted/sec counter warrant investigation, except when wireless or satellite transmissions are involved.

Adaptive retransmission refers to the mechanism TCP uses to determine how long to wait before retransmitting an unacknowledged packet. If the retransmission timeout is too small, TCP sessions will be sending unnecessary duplicate data transmissions. In this situation, if the TCP session were able to wait a little longer for ACK messages, these retransmissions could be avoided. Retransmitting a packet unnecessarily wastes network bandwidth. Suppose transmission failures result from an overloaded Layer 3 router that is forced to drop packets. If the route is overloaded, TCP senders need to be extra careful about retransmissions. You do not want to have error retransmissions occurring so frequently that they further overload any routers that are already saturated.

On the other hand, if a TCP session waits too long before retransmission, application response time suffers. If, in fact, packets are being dropped before they can reach their intended destination, waiting too long unnecessarily slows down the intended interaction with the application.

Retransmission timeout Because routes can vary so much, TCP maintains a unique retransmission timeout (RTO) value for each active TCP connection. TCP's error retransmission behavior is based on the RTT of the specific session. TCP establishes the RTT of the session at the outset using the *TcpInitialRTT* parameter, which governs reconnect transmissions. By default, *TcpInitialRTT* is set to 3 seconds. There is an exponential backoff component to retransmissions whereby the retransmission delay based initially on *TcpInitialRTT* is doubled for each retransmission attempt. Once a SYN-ACK packet is returned and a session is established, TCP can calculate the RTT of the actual connection from subsequent transmission acknowledgements. TCP uses those measured values of the actual RTT over the connection to determine a good retransmission timeout value.

During the course of a session, TCP measures the actual RTT. By default, TCP samples values of RTT once per send window, rather than doing it for every packet sent and acknowledged. For relatively small send windows, this sampling technique is adequate.

Timestamps option For larger send windows, the rate of sampling RTT values might be too small to adequately assess current network conditions, which can change rapidly. Alternatively, Windows Server 2003 supports the RFC 1323 TCP Timestamps option, which makes it easy for TCP to calculate RTT precisely for every packet. When the Timestamps option is set, a TCP sender plugs into the header a current 4-byte timestamp field. The receiver then echoes this timestamp value back in the ACK packet that acknowledges receipt of the original send. The sender looks in the ACK packet for its original timestamp, which is then subtracted from the current time to calculate the session RTT. When the Timestamps option is specified, TCP can calculate accurately the RTT associated with every send-ACK sequence in the data transmission stream.

Calculating RTO TCP bases RTO on the actual RTT calculated for the session. The actual RTO calculation uses a weighted smoothing formula as recommended in RFC 1122. This formula is based on the current RTT and the variance between the current RTT and the mean RTT. This generates an RTO value that responds quickly to sudden changes in network performance.

TCP will retransmit an unacknowledged packet once the current RTO timeout value is exceeded. The *TcpMaxDataRetransmissions* registry parameter determines the number of times TCP will attempt to retransmit a packet before it decides to abandon the session altogether. The default value of *TcpMaxDataRetransmissions* is 5. After each unsuccessful retransmission, TCP doubles the RTO timeout value used. This use of exponential backoff parallels Ethernet's collision avoidance mechanism; once retransmission needs to occur, it is good idea for TCP to extend the delay time between retransmissions to avoid making a congested network situation even worse.

Karn's algorithm A problem arises in calculating RTT for a successfully retransmitted packet. If Timestamps are not in use, TCP cannot tell whether the ACK received is for the initial packet or for one or more of the retransmitted packets. Windows Server 2003 ignores RTT values for any retransmitted packets. However, if time stamps are used, the echoed time stamp identifies unambiguously which packet transmission is being acknowledged. The strategy of not measuring RTT on retransmitted packets, as well as exponentially backing off the retransmission timeout on multiple retransmissions, is collectively known as Karn's algorithm.

Fast retransmit Suppose a TCP receiver receives a packet out of sequence. In other words, the packet received contains a SequenceNumber that is higher than the expected sequence number. The implication is that a missing packet is somewhere in the sequence. Hopefully, this packet is on its way via a slower route and will arrive shortly. Alternatively, perhaps the missing packet was dropped by a router in flight somewhere along the route. (IP, of course, by design is not a reliable delivery service.)

Fast Retransmit is a mechanism for a receiver that has received a packet out of sequence to inform the sender so that the packet can be retransmitted without waiting for the retransmission timeout value to expire. This is a bit tricky, because if the packet is on its way via a slower route, retransmitting it is a waste of network bandwidth. On the other hand, forcing the receiver to wait for the RTO value to expire might engender an unnecessarily long wait, especially as additional packets from the same session start to pile up at the receiver's end.

The receiver, of course, cannot acknowledge a packet it has not received. What the receiver does upon receipt of an out-of-order packet is send an ACK showing the still current high water mark of the contiguous byte stream it has received. In other words, the receiver sends a duplicate ACK. This shows the sender that the connection is alive and that packets are getting through, although it does not tell the sender precisely which packets arrived. When these duplicate ACKs arrive back at the sender, the sender responds by retransmitting unacknowledged packets regardless of whether RTO triggers the event. This is known as Fast Retransmit. (If SACK is enabled, the sender can more easily identify missing packets in the byte transmission stream and just retransmit those. Without SACK, the sender has to start retransmitting all unacknowledged packets.)

A registry parameter named *TcpMaxDupAcks* determines how many duplicate ACK packets a sender must receive before retransmitting unacknowledged packets. Valid values for *TcpMaxDupAcks* are 1, 2, and 3. Windows Server 2003 sets *TcpMaxDupAcks* to 2 by default, which is slightly more aggressive than the RFC-2581 recommendation of 3. The Windows Server 2003 value reflects a more current sense of the reliability of the Internet.

Chapter 6

Advanced Performance Topics

In this chapter:

Processor Performance 538
Memory Performance 608
The System Monitor Automation Interface. 628

This chapter discusses a variety of advanced performance topics. These topics are primarily important when planning for very large application servers, and they do not lend themselves to ready characterization and easy answers. Tackling them demands a strong technical background, some of which this chapter can supply. The discussion here assumes you have read and understood the major concepts that were introduced in Chapter 1, "Performance Monitoring Overview," and reinforced in their practical implications throughout the other chapters.

This chapter also documents the use of several advanced configuration and tuning parameters that are available. It is highly recommended that you attempt to manipulate these tuning parameters only after conducting a thorough study of the current environment, and only after you understand what can go wrong if you make a change in any of these sensitive areas.

The first part of the chapter focuses on a variety of processor hardware performance concerns, beginning with the pipelined superscalar architectures of the server-class machines that run the Microsoft Windows Server 2003 operating system. This leads to a discussion of multiprocessing architectures common to machines used as servers. Because most of these machines are classified as shared memory multiprocessors with uniform memory latencies, the bulk of the discussion is on the performance, tuning, and scalability of this type of machine. Hyper-Threaded multiprocessors are also briefly considered, as well as large-scale *Cache Coherent Non Uniform Memory Access* (ccNUMA) architectures that surpass the capacity limits of shared memory multiprocessors.

Several advanced memory performance and capacity planning topics are also discussed. One section focuses on the extended virtual addressing configuration options available for 32-bit Windows Server 2003–based machines and when to use them when you encounter 32-bit virtual memory constraints on your servers. Note that running the 64-bit version of Windows Server 2003 and 64-bit native server applications provides access to a massive virtual address range that renders these tuning and configuration options moot. The memory section also discusses a technique to use for memory capacity planning that is both useful and easy to apply.

Finally, a section is devoted to documenting the System Monitor ActiveX control that allows you to script performance monitoring sessions using this graphical user interface.

Processor Performance

This section focuses on a variety of processor hardware performance concerns. It begins with a brief account of the dominant architectures used by server-class machines that run the Windows Server 2003 operating system. Processor time-slicing by the operating system thread Scheduler is then reconsidered; this is followed by a discussion of the configuration parameter that is provided to control the time-slicing values used by the Scheduler. This control usually does not have a major impact on performance on most systems. But because it is one of the tuning knobs within easy reach, it generates more discussion than would ordinarily be warranted for a control that normally does so little. The examination here tries to place its usage into perspective.

The section then returns to hardware performance issues, with a discussion of multiprocessing scalability. The most widely available form of multiprocessing, namely *shared memory multiprocessors* with uniform memory latencies, is the primary focus. Some of the obstacles to successful shared memory multiprocessor scalability are described, as are the configuration and tuning strategies designed to overcome these obstacles. A central concern in this section is the application-oriented tuning options available. These are configuration and tuning options for multithreaded server applications designed to help them scale effectively on large *n*-way multiprocessors.

This section also introduces Windows System Resource Manager (WSRM), a new policy-based performance management tool expressly developed to help automate the performance management of large-scale multiprocessors running the Windows Server 2003 operating system. WSRM is a component available with both Enterprise Edition and Datacenter Edition of Windows Server 2003.

Hyper-Threaded multiprocessers that implement simultaneous multithreading are briefly considered. These are machines that run two separate instruction streams concurrently on a single processor core. This architecture generates unique performance considerations. Finally, the ccNUMA architectures used to build the very largest servers capable of running the Windows Server 2003 operating system are considered. These machines provide nonuniform memory latency, depending on whether application threads are addressing local locations or remote memory locations. Major changes to the Windows Server 2003 operating system's Scheduler and virtual memory manager functions were necessary to support these ccNUMA machines. Additional scalability issues that might apply to applications running on these ccNUMA machines are also highlighted.

Instruction Execution Throughput

A basic understanding of processor hardware is relevant to many performance and capacity planning issues. This background is also useful when selecting among the wide variety of single and multiple processor hardware alternatives available. It is also a prerequisite to using some of the advanced processor performance tuning mechanisms on your Windows Server 2003–based machines. This section looks inside a microprocessor from the standpoint of instruction execution throughput, and offers steps you can take to improve performance on a large-scale multiprocessor.

Processor Hardware Basics

Processors are machines designed to execute the arithmetic and logical instructions that form the heart of any computer program. Table 6-1 contains examples of computer instructions used to build typical computer programs.

Table 6-1 Example Computer Instructions

Instruction	Examples
Arithmetic operations	Add, subtract, multiply, and divide integers
Logical operations	Compare two values; test for zero or nonzero values, or positive or negative numbers; perform logical OR and AND operations
Control operations	Conditional and unconditional branch, procedure call, return, trap
Data transfers	Load register, store register
String operations	String move, string compare
Graphics instructions	Perform byte and bit array operations associated with graphical processing
System control functions	Set system state, task switching, interrupt, signal processor
Floating point operations	Add, subtract, multiply, and divide using high-precision numerical values

Instructions operate on data stored in computer memory or in fast, internal memory locations called *registers*. Register locations are either named or numbered, as in GPR0, GPR1, and GPR2, which stand for general purpose register 0–2, respectively. Any memory location is uniquely and unambiguously identified by its physical address, which is limited in size by the number of bits used to form the address. Thus, 32-bit registers can obviously be used to address 2^{32} bytes, or 4 GB of RAM. This limitation led to the introduction of new addressing techniques allowing 32-bit systems to address up to 36 bits of physical addresses, or 2^{36} bytes (64 GB of RAM). 64-bit machines can in theory address 2^{64} bytes, which is an extraordinarily large number equal to 18 quintillion bytes, or 1.8×10^{19}. However, current implementations of 64-bit architectures implement fewer actual physical address bits, because as of this writing, no machine can be built with that much memory.

Note Each 1 kilobyte–segment of computer memory contains 1024 individual bytes. This shorthand notation constantly leads to confusion. All references here to 1 MB refer to 1024 kilobytes of memory. For example, 2^{32} bytes equal 4,294,967,296 bytes. For the sake of simplicity, this quantity is spoken of as 4 GB. Similarly, 2^{64} bytes equal 18,446,744,073,709,551,616, which is 18 exabytes. That might seem like an impossibly large number, but according to a study conducted at the School of Information Management and Systems at the University of California at Berkeley, in 1999, human beings across the world produced about 1.5 exabytes of storable content. This is equivalent to about 250 megabytes for every man, woman, and child on earth. Details are available at http://www.sims.berkeley.edu/how-much-info.

A basic throughput-oriented measure of a computer's capacity is its Instruction Execution Rate, or IER. A processor's IER is a function of its clock speed and the number of instructions executed each clock cycle. Instruction execution rate also depends on the specific instruction mix and other factors. Separately, you might be able to obtain a measure of the processor's internal IER by using tools that are available from the manufacturer. This internal measure of the processor's current throughput in instructions executed per second is something far different from the counters you can access using Performance Monitor, which reports processor utilization as a percent busy. External measurements of processor utilization that are available using Performance Monitor often reflect internal processor instruction execution throughput, but not always.

Processor instruction execution throughput is a complex function of many factors, both internal and external. The internal factors include the clock speed; the instruction set; and architectural considerations such as the processor's pipeline and the sizes of internal caches. Many of the external factors are workload-dependent: the instruction mix, the degree of instruction-level parallelism, the degree of multithreading, the amount of shared data (on a multiprocessor system), and so on.

Processor Performance Hardware Mechanisms

Processors contain several complex internal mechanisms designed to improve instruction execution throughput. Most of these complex mechanisms are designed to be transparent to the execution of typical computer programs, and it is beyond the scope of this discussion to describe them in any depth. However, the architectural features that do interact with program threads and have significant performance considerations are the focus of this section. In the case of those processor architecture features that have the greatest potential performance impact, specific Windows Server 2003 operating system functions are available to boost instruction execution throughput.

Some of the important internal features of the processor architecture that impact instruction execution throughput include these:

- **Pipelining** Pipelining breaks down the individual instructions specified in a single execution thread into discrete steps, which are known as the pipeline *stages*. Individual instructions are executed within the pipeline stage by stage during successive processor clock cycles. Multiple instructions are then loaded into the pipeline, and their execution is overlapped so that portions of multiple instructions are executed in parallel, similar to a factory assembly line.

> **Note** Pipelining is one reason that equating the performance of a processor with its *clock speed* is a mistake. The processor clock speed, or the MHz rating of the processor, is an important indicator of its performance relative to machines with a similar architecture. Nevertheless, using MHz alone to compare the performance of machines is unreliable, especially across very dissimilar instruction execution architectures. For example, one machine running at twice the clock speed might also have an instruction execution pipeline that has twice as many stages. The number of clock cycles it takes to execute individual instructions also varies because of many external factors, such as cache effectiveness. Workload-dependent variability in processor performance also makes it very difficult to compare the performance of two machines with different architectures by using simple measures like clock speed. Consequently, clock speed can be a very misleading indicator of processor performance. Historically, that is why the processor's MIPS rate (MIPS stands for millions of instructions per second), is often humorously referred to as a Misleading Indicator of Processor Speed.

- **Superscalar execution** Superscalar execution is the term used to describe processors that contain two or more instruction execution pipelines that can operate in parallel. Superscalar processors can fetch, execute, and retire multiple instruction stages in a single clock cycle.
- **Out-of-order execution** Out-of-Order execution refers to the ability of a processor to execute instructions in a different sequence than was originally specified. When one instruction stage *stalls* the pipeline, the pipeline might be able to operate on an instruction that is later in the program's sequence if that instruction can be processed successfully in that stage. Of course, if instructions are executed out of order inside the pipeline, the processor must later ensure that they are *retired* in the sequence originally specified by the program.
- **Microarchitecture** A complex instruction set computer (CISC), such as one from the Intel IA-32 line, breaks complex instructions down into simpler micro-operations, which are similar to the simple instructions used by a reduced instruction set computer (RISC) system. Micro-operations are designed to be processed efficiently by the instruction execution pipeline. The microarchitecture of the Intel Pentium Pro and later 32-bit processors in that line also feature both superscalar execution and out-of-order execution.
- **Predication and speculation** Predication and speculation are two techniques designed to take advantage of the extensive processing resources available on some processors. Currently, the Intel IA-64 instruction set that is found on 64-bit Itanium machines supports both predication and speculation. *Predication* refers to a method for executing conditional instructions. Historically, pipelined processors relied on *branch predication* to load instructions that are subject to conditional IF-THEN-ELSE–type of logic. Assuming sufficient resources are available to operate on instructions in parallel, it can be more efficient to use predication to execute mutually exclusive sets of conditional instructions and later throw away any results that are not needed.

  Another technique called *speculation* attempts to increase instruction level parallelism by speculatively loading instruction operands from memory and performing computations on those values before it is entirely certain what those memory-resident values should be. The processor is then prepared to abandon the results of those operations and redo the operations if intermediate instructions change the data values that were loaded speculatively.

- **Caches** Caches are extremely fast banks of memory located on the processor chip that are used to store recently referenced data and instructions. Processors often contain several types of caches. To speed up the translation of virtual addresses to physical memory addresses, for example, there is a special cache called the Translation Lookaside Buffer (TLB). Using separate caches for data and instructions is a common practice. The processor can access information stored in any of these caches significantly faster than it can by accessing RAM directly, which is why cache effectiveness is so important to instruction execution throughput. Several performance optimizations built into Windows Server 2003 are associated with the performance of internal caches on multiprocessors. These operating system tuning options are discussed in the "Multiprocessors" section of this chapter, which discusses these multiprocessing optimizations.
- **Simultaneous multithreading** Simultaneous multithreading entails running two or more instruction execution threads in parallel on a single physical processor core. Intel calls its processors that support simultaneous multithreading *Hyper-Threading*, which is available on many of the company's current processors. Hyper-Threading is discussed in more detail in the "Hyper-Threading" section in this chapter.

64-Bit Architectures

It is also possible to select different architectures from among machines featuring 64-bit addressing so that you can run those workloads that require extended addressing capabilities. 64-bit machines can address much larger-sized RAM configurations than 32-bit processors can. 64-bit architectures can definitively alleviate any virtual memory constraints that you encounter in 32-bit server applications, such as those discussed in much greater depth later in this chapter. Under those circumstances, the extended addressing capabilities of 64-bit platforms might be far more significant than any other processor-related performance factor. Windows Server 2003 supports up to 64 64-bit processors on a single machine, double the number of 32-bit processors that are supported, so for computer-bound workloads, it is also possible to configure 64-bit computers that are much more powerful than the largest 32-bit server machines.

You are currently able to select between two approaches to 64-bit computing for those applications that demand it. The 64-bit Itanium processors from Intel use an entirely different instruction set from 32-bit Intel processors, but are still capable of running your existing 32-bit applications in an emulation mode. Itanium processors, which implement the Intel IA-64 architecture, use what is known as an EPIC design. (EPIC stands for Explicitly Parallel Instruction Execution.) IA-64 processors rely upon compiler technology that can generate efficient code optimized for execution on these machines. Microsoft compilers in the .NET family of languages currently support code generation of native IA-64 programs.

Note To take advantage of 64-bit addressing and other advanced features of the Intel IA-64 architecture, you must port your 32-bit applications to the Win64 platform and recompile them to execute in 64-bit mode. It is a good idea for developers to use a single, common source-code base for the 32-bit and 64-bit versions of their applications. See the Win64 platform SDK discussion of this subject beginning with an article called "Getting Ready for 64-bit Windows," available at http://msdn.microsoft.com/library/en-us/win64/win64/getting_ready_for_64_bit_windows.asp for more information.

The AMD-64 architecture is the other 64-bit computing alternative currently capable of running the Windows Server 2003 operating system. The AMD-64 architecture provides a 64-bit addressing mode, known as *long mode*, while still maintaining instruction level compatibility with current 32-bit x86 programs. Support for the AMD-64 architecture is provided in Windows Server 2003 Service Pack 1. In its *legacy mode*, the AMD-64 architecture executes standard x86 instructions that are limited to 32-bit addresses. In long mode, it can still execute, at full speed, compatibility-mode programs that use 32-bit addresses. In long mode, programs also have access to twice the number of internal registers compared to a 32-bit x86 machine.

Operating system support for 64-bit platforms The Windows Server 2003 operating system is available in 64-bit versions for both IA-64 and AMD-64 machines. This 64-bit version is a complete port of the operating system, functionally identical to the 32-bit version, including all the supporting services, administrative tools that plug into a 64-bit version of MMC, and other applications. You even get a 64-bit version of Microsoft Notepad!

Running 32-bit applications on 64-bit Windows-based systems The 64-bit version of Windows Server 2003 runs 32-bit applications using the WOW64 compatibility layer. WOW, which stands for Windows On Windows, is intended to run 32-bit personal productivity applications needed by software developers and administrators. It is not intended to run 32-bit server applications.

WOW converts programs running in 32-bit mode that issue 32-bit Win32 function calls to 64-bit mode so that they can be serviced, and then converts them back again.

This automatic conversion of 32-bit function calls to 64-bit is known as *thunking*. The WOW64 layer performs thunking automatically. WOW64 is implemented in User mode, as a layer between Ntdll.dll and the Kernel. It consists of the following modules. These are the only 64-bit DLLs that can be loaded into a 32-bit process.

- **Wow64.dll** This DLL provides the core emulation infrastructure and the thunks for the function calls to Ntoskrnl.exe.
- **Wow64win.dll** This DLL provides the thunks for the function calls to Win32k.sys.
- **Wow64cpu.dll** This DLL performs x86 instruction emulation. It also executes mode-switch instructions on Intel Itanium processors.

WOW allows you to run both 32-bit console and GUI applications, as well as 32-bit services. The WOW compatibility layer handles tasks such as maintaining different registry hives for 32-bit and 64-bit programs. A separate system directory for 32-bit binaries is also provided. The 64-bit binaries still use the System32 directory, so when a 32-bit application is installed on the system, the WOW layer makes sure to put the 32-bit binaries in a new directory called SysWOW64. It does this by intercepting calls to APIs like *GetSystemDirectory* and returning the appropriate directory, determined by whether the application is running under WOW.

Similar compatibility-mode situations arise with the registry. To allow 32- and 64-bit COM applications to coexist, WOW64 presents 32-bit applications with an alternate view of the registry. The 32-bit applications see an HKEY_LOCAL_MACHINE\Software registry tree completely separate from the true HKEY_LOCAL_MACHINE\Software tree. Having separate registry trees isolates HKEY_CLASSES_ROOT because the per-machine portion of this tree resides within the HKEY_LOCAL_MACHINE\Software tree. Because both 32-bit and 64-bit COM servers can be installed on the system under the same class identifier (CLSID), the WOW layer must redirect calls to the registry to the appropriate 32-bit or 64-bit hives. Keep in mind that 32-bit processes cannot load 64-bit DLLs, and 64-bit processes cannot load 32-bit DLLs. The system does provide for interoperability between 32-bit applications and 64-bit system services using COM interfaces. And for desktop applications, they can use the Clipboard to move data between 32-bit and 64-bit applications.

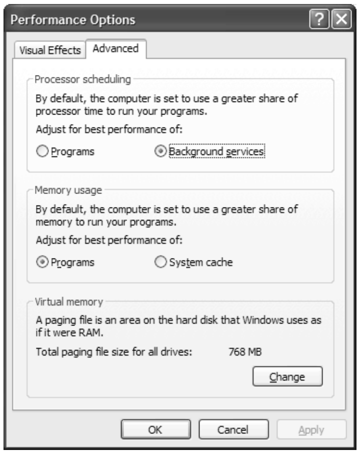
A 32-bit application can detect whether it is running under WOW64 by calling the *IsWow64Process* function. It can obtain additional information about the processor environment by using the *GetNativeSystemInfo* function.

Time-Slicing Revisited

As discussed in Chapter 1, "Performance Monitoring Overview," *time-slicing* is a technique that keeps CPU-bound threads from executing continuously when other threads with equal priority are waiting to be scheduled. Windows Server 2003 provides a default time slice interval (or *quantum*) that is based on the workload expected for most application servers. The time slice default value is chosen with larger, multiple processor configurations running server applications like Microsoft SQL Server in mind. Executing with the Windows Server 2003 defaults, compute-bound threads are given a healthy slice of processor time before they are subject to preemption by the Scheduler thread.

Note The actual duration of a time slice is hardware-dependent. The default time-slice value for Windows Server 2003 is on the order of 100–200 ms, which allows compute-bound threads to execute for a relatively long time slice before being preempted. By contrast, the time-slice value for a workstation running Microsoft Windows XP is in the range of 20–40 ms. The smaller time-slice value that Windows XP uses is suitable for most interactive workloads.

Windows Server 2003 does allow you to choose between two time-slicing default values. To bring up the Performance Options dialog box illustrated in Figure 6-1, from Control Panel, select the System application, click the Advanced tab, and click the Performance Options button. The Processor Scheduling option presents you with a choice between Background Services, which runs the long Windows Server 2003 default time slice; or Programs, which runs with the shorter time-slice values assigned in Windows XP by default.



Processor scheduling

By default, the computer is set to use a greater share of processor time to run your programs.

Adjust for best performance of:

Programs [✓ Background services]

Memory usage

By default, the computer is set to use a greater share of memory to run your programs.

Adjust for best performance of:

[✓ Programs] System cache

Virtual memory

A paging file is an area on the hard disk that Windows uses as if it were RAM.

Total paging file size for all drives: 768 MB

Figure 6-1 The Advanced tab of the Performance Options dialog box

Most program threads have instruction streams that issue voluntary waits (either to perform disk or network I/O, or to wait on a lock or an event) long before their time-slice value expires. However, if memory-resident caching is effectively used to avoid most I/Os to disk, allowing server application threads to execute for a healthy time slice will improve the responsiveness of many applications. Although the default time slice might not be optimal for all types of server workloads, if you lack strong evidence that this control is inhibiting performance, you should not change it.

Exceptions to this general rule do exist, however, for some servers. The most important exception is for Terminal Services workloads. Servers set up to manage a large number of Terminal Services sessions need to run large numbers of processes and threads associated with desktop applications. Most of these desktop applications run in short bursts of activity that are not CPU-bound. However, on occasion, interactive users perform long-running, compute-bound tasks, like recalculating a complex spreadsheet. A long-running, compute-bound thread can then hold onto the processor for the duration of a long time slice under the default setting. This potentially causes other interactive threads long delays waiting in the Ready Queue to be dispatched. Under these circumstances, the shorter time-slice interval associated with Windows XP–based workstations is probably a better choice for a server managing a large number of Terminal Services sessions.

Tip If interactive workloads associated with Terminal Services dominate on a server, change the default setting to favor programs instead of background services.

You should also be aware that convincing evidence of the time-slice value being wrong for your system workload is difficult to come by. The Scheduler is invoked hundreds of times per second per processor, and any ETW trace data that you gather on context switching is liable to be voluminous. The only performance counters that can shed much light on how time-slicing might be impacting the performance of your applications are the Thread Wait State Reason codes, and gathering them is also subject to high volume and high overhead considerations. The % Processor Time measurements available are sampled at a relatively slow rate compared to the number of times the Scheduler is invoked each second on a busy system. The counters that measure the rate of Context Switches/sec would be helpful if only scheduler-initiated context switches were counted, but that is not the case.

A final consideration that should caution you about the need to change the default time-slice setting is that time-slicing accomplishes something only when waiting threads are running at the same dispatching priority as a CPU-bound application whose threads are monopolizing the processor. In the absence of a lengthy processor queue, there is no justification for setting the time-slice value one way or another. Even in the presence of a lengthy processor queue, manipulating the time-slice parameter cannot always improve the situation.

There is a registry setting called *Win32PrioritySeparation* under HKLM\SYSTEM\CurrentControlSet\Control\PriorityControl that is manipulated when you use the Performance Options dialog box. For Windows Server 2003, *Win32PrioritySeparation* is set to a binary value of 0x18 by default. The bit setting for *Win32PrioritySeparation* is interpreted as three binary 2-bit fields. Working from the left, two bits at a time, *Win32PrioritySeparation* encodes this information:

- Whether a short (workstation) or long (server) time-slice interval is used.
- Whether fixed or variable time-slice intervals are used.
- Whether time-slice stretching is performed on behalf of the foreground application. Time-slice stretching is something designed for the interactive workloads found on most workstations to boost the performance of a foreground thread slightly.

Transform the hex code into binary and you get '011000'b. The '011000'b encoding translates into the long (server) time-slice value, fixed length intervals, and no time-slice quantum stretching for the foreground application. Changing the parameter to optimize performance for applications changes the *Win32PrioritySeparation* code to x'02'. This translates into a short time-slice value, variable length intervals, and stretching the quantum of the foreground application.

Multiprocessors

Whenever a workload strains the capacity of a single processor, one potential solution is to add processors to share the load. Machines configured with multiple processors are called *multiprocessors*. This section discusses several important aspects of multiprocessor scalability for servers running Windows Server 2003.

Multiprocessing technology lets you harness the power of multiple microprocessors running a single copy of the Windows Server 2003 operating system. Enterprise-class server models with multiple CPUs abound. When is a 2-, a 4-, an 8-, or even a 64-way multiprocessor machine a good answer for your processing needs? How much better should you expect the performance of a server with multiple engines to be? What sorts of workloads lend themselves to multiprocessing? These are the difficult questions this section helps you answer.

Shared Memory Multiprocessors

The most common approach to building multiprocessors allows multiple processors to access a single, common block of shared RAM. This is known as a *shared memory* multiprocessor, as depicted in Figure 6-2.

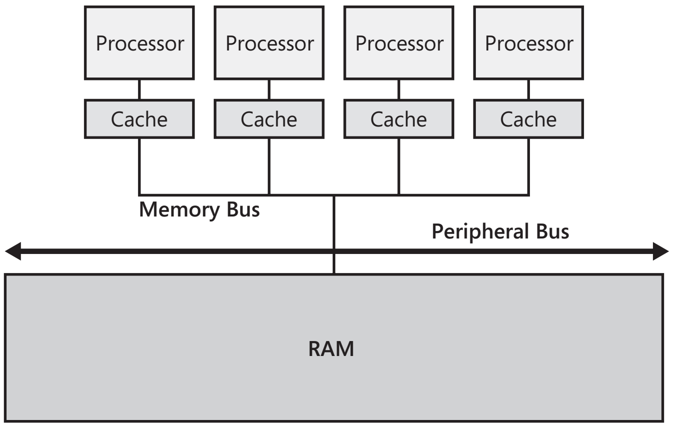


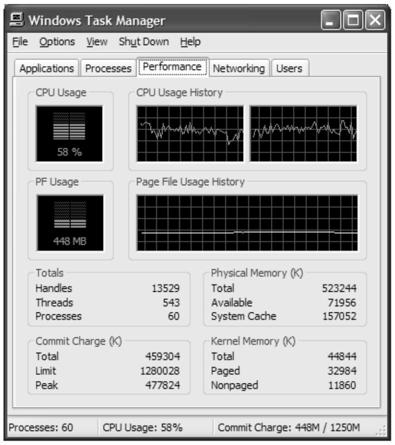
Figure 6-2 A shared memory multiprocessor

Figure 6-2 shows a series of microprocessors connected to memory via a shared memory bus. Larger enterprise systems use multiple processor buses because of electrical limitations. Each processor can access all the physical memory configured. The processors share—via the operating system's serialization mechanisms—access to other resources as well. The keyboard, mouse, video display, disks, network adaptors, and so on, are all shared by the available processors.

Figure 6-2 does show that each microprocessor has its own private cache or caches. An earlier discussion of processor caches emphasized that caches play a key role in the performance of a single processor. Caches are no less essential to the performance of multiprocessors. Each processor element in a multiprocessor accesses and updates memory through it own private caches. This raises memory synchronization issues that do not arise with a single processor system. What happens, for example, when instructions running concurrently on two separate processors need access to the same item in memory at the same time? More importantly, what happens when instructions running concurrently on two separate processors need to access and change the same item in memory? The way that memory locations that are accessed and changed by instructions running on one processor in its private cache copy are communicated to other processors that might need access to the same data items is known as the *cache coherence* problem.

Single system image A typical multiprocessor computer presents a single system image. This multiprocessor relies on one copy of the operating system to manage all the available system resources and ensure that these resources are used safely by threads running concurrently on the separate processors. Note that each processor in a multiprocessor system is capable of executing work independently of another. Separate, independent threads are dispatched, one per processor, and they all run in parallel.

Only one copy of the Windows Server 2003 operating system runs, and it controls which threads run on which processors. This is accomplished by coordinating per-processor Ready Queues, a good example of a collective resource shared among independent threads that are executing in parallel on separate processors. From a performance monitoring perspective, you see multiple instances of the processor object reported in both Task Manager (as illustrated in Figure 6-3) and System Monitor. Separate idle threads are created, one per processor, making it possible to account for processor utilization on a per-processor basis.



Totals	
Handles	13529
Threads	543
Processes	60

Physical Memory (K)	
Total	523244
Available	71956
System Cache	157052

Commit Charge (K)	
Total	459304
Limit	1280028
Peak	477824

Kernel Memory (K)	
Total	44844
Paged	32984
Nonpaged	11860

Processes: 60 CPU Usage: 58% Commit Charge: 448M / 1250M

Figure 6-3 Task Manager reporting multiple instances of the processor object

Symmetric multiprocessing The operating system is capable of scheduling Ready threads on any available processor. This type of operating system support for a shared memory multiprocessor is called *symmetric* multiprocessing, or SMP for short. Windows Server 2003 is fundamentally an SMP operating system, but it does allow you to configure asymmetric configurations when they are desirable. When asymmetric configurations might be desirable for performance reasons is discussed in more detail in a later section, "Multiprocessor Configuration and Tuning Strategies."

One of the reasons the shared memory approach to multiprocessing is popular is that most programs will run efficiently on a multiprocessor without major modifications. To take advantage of multiple processors, however, an application program must, at a minimum, be multithreaded. Running in isolation, a single-threaded process can execute no faster on a machine with multiple processors than it can on a single processor machine. On the other hand, a process with two threads might be able to execute twice as fast on a multiprocessor than it could on a single processor system. This performance boost is easier to accomplish in theory than it is in practice. In practice, writing multithreaded programs that scale linearly on multiprocessors is very challenging work.

A multithreaded process running on a multiprocessor introduces serialization and synchronization issues. Multiple threads running inside the same process have identical addressability to the same virtual memory locations within the process address space. Multithreaded applications must be implemented in a *thread-safe* manner, which implies that some sort of synchronization be established so that threads can access and update shared resources safely. The Windows Server 2003 operating system supplies a set of synchronization objects and related API services for multithreaded application threads to utilize. These synchronization objects include *critical sections*, *semaphores*, and *mutexes*. You will find corresponding performance counters that show you the number of these synchronization objects that the operating system has been requested to create in the Object object. For more information about the use of these synchronization objects, see the SDK documentation at http://msdn.microsoft.com/library/en-us/dllproc/base/synchronization_objects.asp. Other application programming-oriented implications of multiprocessors are discussed later in the "Thread Pooling" section.

Multiprocessing Scalability

Under ideal conditions, if you added a second processor to a single processor system, you would expect to have a system with twice the processing power. If you added a third and a fourth processor, you would expect a system with three and four times the processing power. Ideally, a computer system's performance would scale linearly with the number of processors installed. In actuality, the improvement in performance levels you are likely to observe will diminish as you continue to add processors to a shared memory multiprocessor system.

Workload parallelism One of the crucial factors affecting the scalability of server application workloads running on shared memory multiprocessors is insufficient parallelism in the workload. For example, if you are considering using an 8-way multiprocessor to run one of your server application workloads, the first question to ask is whether your workload generates enough parallel execution threads to take full advantage of the processor resources available.

If an application process such as Microsoft SQL Server, for example, does not always have eight ready Worker threads, the application is unable to take advantage of all the parallel processing resources available. Moreover, always having at least eight Worker threads ready with work to process on an 8-way multiprocessor would seldom be desirable because queuing delays would inevitably develop that would slow down application response time. Matching the parallelism in the workload to the number of processors you need is also complicated by the fact that you need to allow for scheduling at least *some* operating systems threads to perform critical system-oriented tasks from time to time. To configure sufficient processors to ensure that no critical processing thread is ever waiting for resources is probably not cost-effective. As discussed in Chapter 1, "Performance Monitoring Overview," it is almost impossible to optimize for throughput, response time, and cost at the same time.

Linear scalability of multithreaded applications is difficult to achieve for other reasons. As the name implies, shared memory multiprocessors share memory and other resources, and this resource sharing naturally leads to contention. Another of the shared resources subject to contention that can impact multiprocessor scalability is the front side bus, which is used to connect the processors to shared memory. Locking structures required to mediate access to shared data structures stored in memory also inevitably slow down access to memory when there is lock contention. Caching considerations are also very important in shared memory multiprocessors, as is discussed in more detail later in this chapter. In addition, software that is not carefully engineered to run efficiently on multiprocessors can face other scalability problems.

Lock contention To ensure that shared data can be updated with integrity when multithreaded applications are running concurrently on multiple processors, locking structures are required to serialize access to these resources. Locks are software constructs that block one thread from accessing a shared data structure while another thread is modifying that data. Locks can become a source of very serious contention for all types of software running on large-scale multiprocessors, ranging from operating system services to device drivers to multithreaded User-mode applications. Designing more granular locking structures is one technique to reduce lock contention. Another approach is to replace monolithic locks with per-processor data structures that can be accessed without blocking. The per-processor work queues that the File Server service builds are a good example of this approach. This server application architecture that relies on thread pooling is discussed in more detail in the section entitled "File Server Service" later in this chapter.

Note Locks are created and managed by special instructions. The CMPXCHG instruction is a good example of an instruction used for setting a lock on a multiprocessor. CMPXCHG tests and sets a memory location that is used as a lock word. CMPXCHG compares a value stored in one 32-bit GPR to a value stored in the lock word stored in memory. If the comparison is true, CMPXCHG updates the memory location by storing the value from an implied second GPR. Both the testing of the memory location and assigning a new value to it are performed in a single, *atomic* instruction that runs to completion uninterrupted. Multiprocessors also ensure that the results of atomic instructions that change memory are immediately visible to all other processors.

One performance implication of executing instructions that test and set locks stored in memory is that activity on one processor can stall another processor in the shared memory multiprocessor configuration.

Note In addition to Critical Sections, Windows Server 2003 provides other User-mode application synchronization services. These include *semaphores*, which provide more flexible access to critical code sections so that multiple Readers can examine shared data while providing exclusive access to the same data for Writers intent on changing that data. Another synchronization service is the *mutex*. One important use of mutexes, which is short for *mutual exclusion*, is to provide a synchronization service that can be used by threads across multiple processes. A mutex ensures that a thread in one process waiting on a thread in another process is notified when the thread that the first thread is waiting for terminates unexpectedly. Without notification from the mutex, a thread waiting on an out-of-process event would hang indefinitely when the other process failed.

Spin locks The need to serialize access from multiple, concurrently executing threads to critical sections of code can have significant performance implications for operating system functions as well. Operating system functions, including device drivers, utilize *spin locks* to serialize access to shared data structures. Spin locks are tight loops in which the waiting thread is occupied continuously testing and retesting a lock word guarding a critical section. The waiting thread executes the spin lock code continuously because the delayed thread can do nothing else useful until the shared resource is released. Critical sections of code guarded by spin locks need to be carefully engineered to execute swiftly so that waiting threads are not delayed excessively in unproductive spin loops. The amount of time code spends in unproductive spin loops testing locks that are guarding heavily accessed critical sections also tends to increase with the number of processors.

Device drivers, for example, use spin locks to synchronize access to critical sections accessed by ISRs running at an elevated dispatching priority. Consider a server with more than one network adapter. When interrupts from two separate adaptors occur at roughly the same time, the driver code for the network adaptors can be executing concurrently on multiple processors. ISRs, DPCs, and other device driver functions that read and modify shared data structures such as queues and other linked lists must synchronize access to these shared resources.

Similar to the synchronization services that the Windows API provides application developers are the synchronization services for device drivers and other operating system functions that the operation system kernel provides. The operating system provides Kernel-mode synchronization services such as *KeInitializeSpinlock*, *KeAcquireSpinlock*, and *KeReleaseSpinlock* that device driver threads can utilize. For more information about the Kernel-mode synchronization services, see the *Device Driver Development Kit* documentation at http://msdn.microsoft.com/library/en-us/kmarch/hh/kmarch/Synchro_7cd71aeb-4d52-40d0-9e61-fe0fecbaba17.xml.asp.

Note Kernel synchronization services such as *KeInitializeSpinlock*, *KeAcquireSpinlock*, and *KeReleaseSpinlock* are a great boon to portability. Device drivers and other system functions written for Windows Server 2003 can call these kernel functions to implement synchronization without having to know what hardware-specific synchronizing instructions are available on the target platform. The operating system kernel, of course, relies on hardware abstraction layer (HAL) functions to implement the appropriate hardware-specific serializing instruction that makes the spin lock function work correctly.

If critical sections are not crafted carefully, serialization delays are apt to become major factors impacting multiprocessing scalability. The developers of Windows Server 2003 use extreme care to ensure that critical sections associated with high-volume operating system services do not become performance bottlenecks. For example, operating system services often use *queued spin locks* that operate more efficiently on multiprocessors. Processor usage profiling tools like Kernrate, discussed in Chapter 5, can quantify the extent of the serialization delays that system functions and device drivers face.

More Info For more information about using queued spin locks in device drivers, see http://msdn.microsoft.com/library/en-us/kmarch/hh/kmarch/Synchro_7cc46160-bbcd-416f-98ea-d41bf80516eb.xml.asp.

Cache coherence Multiple threads associated with device driver code or other functions executing inside the Windows Server 2003 kernel can attempt to access the same memory locations simultaneously. Propagating changes to the contents of memory locations cached locally to other engines with their own private copies of the same shared memory locations is a major issue in designing multiprocessors to operate correctly. This is known as the *cache coherence* problem in shared-memory multiprocessors. The term reflects the emphasis on propagating changes applied to memory locations that must be made immediately visible to instructions currently being executed in other processors.

A common scheme adopted to maintain cache coherence across multiple processors is known as *snooping*. A snooping protocol requires that each processor place the memory addresses of any shared cache lines being updated on the shared memory bus. Each processor "listens" to the shared-memory bus for changes in the status of cache resident memory locations that are being performed by instructions executing on other processors. Snooping keeps every processor's private cache memory synchronized with a minimal performance impact on instruction execution. However, the instruction execution throughput of threads is often impacted when other threads from the same application are running concurrently on different processors. When these threads attempt to access and change the *same* memory locations (for example, the lock status words that control thread serialization), the progress of instructions through the execution pipeline slows and throughput is impeded. The practice of placing memory addresses being accessed on the shared memory bus also generates contention for that shared resource. As more and more processors are added to a shared memory multiprocessor configuration, the capacity of the front side buses (FSBs) that link processor chips to memory can become a hardware bottleneck.

> **Note** Concern that saturation of the shared memory bus limits performance of shared memory multiprocessors underlies approaches to building large-scale parallel multiprocessors using building blocks usually containing four or eight processors with associated local memory. Multiprocessor components are then interconnected using crossbar switches or similar technology that, in effect, scale the bus capacity upwards to match up better with CPU power. Characteristically, in this kind of multiprocessing architecture, processors can access local memory much faster than memory resident on another multiprocessor component. Such machines are called NUMA architectures, which stands for Non-Uniform Memory Access. As discussed later in this chapter, Windows Server 2003 supports cache coherent NUMA (ccNUMA) machines, which are appropriate for parallel processing workloads like data warehousing applications with exceptionally large processor requirements.

False sharing Another performance problem that plagues many multithreaded applications on multiprocessors is known as *false sharing*. False sharing involves two or more threads that access independent data in the same cache line. If one thread updates data in a line of cache that is resident in another processor's cache, the second processor must take notice of the event and either invalidate the cache line if it is clean or write the cache line back to memory if it is dirty (if it contains updated data not yet reflected in main memory). This can slow down the execution of the instruction execution pipeline on the second processor, especially when the second processor subsequently tries to access the cache line and suffers a cache miss.

False sharing is illustrated in Figure 6-4 and Figure 6-5. A single cache line in memory contains a number of data variables. In Figure 6-4, Processors 0 and 1 have recently read variables A and B, respectively, so clean copies of the cache line can be found in both processors' caches. Suppose Processor 0 now needs to write to variable A, as illustrated. To do this, it must obtain exclusive access to the line, which means sending a message out over the front side bus advertising this action. Through some means, such as snooping or directory-based protocols, the message is received by Processor 1. Its copy of the cache line is clean, so it is invalidated. Processor 0 is then free to update variable A.

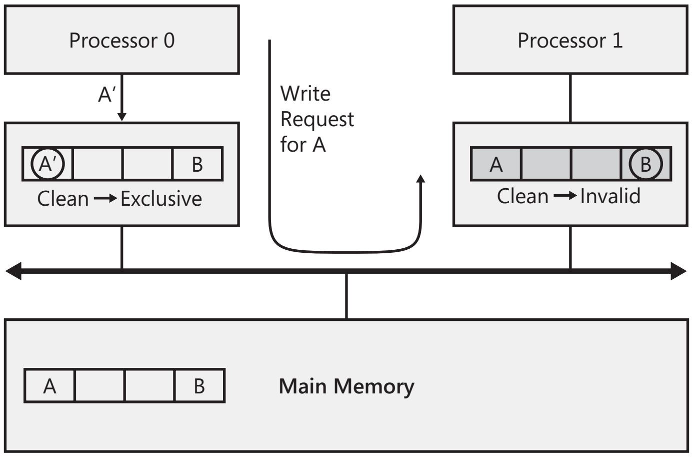


Figure 6-4 False sharing: processor 0 writes memory location A in a shared cache line

Figure 6-5 continues the example instruction processing sequences. Now suppose Processor 1 needs to access variable B again, either for reading or writing. It suffers a cache miss and sends out its request over the front side bus. Processor 0 sees this request and immediately writes out the dirty line to memory (although in some architectures the line might not be written to memory, but rather passed directly to the requesting processor). Figure 6-5 illustrates this example for the case where Processor 1's request is to read Variable B. If Processor 1's request was to write the memory location at B, Processor 0 would be forced to invalidate the entire cache line in its cache. Processor 1 could then read the line from memory and continue its processing.

With false sharing, the cache line bounces back and forth between the two processors. This is unnecessary, as each processor is accessing a different piece of data on the cache line. By separating the two variables, either through careful data structure layout or by compiler optimization, the pointless "cache thrashing" can be eliminated. However, data layout considerations introduce hardware dependencies into the software development process since the size of a line of cache and how the cache is organized can vary from machine to machine.

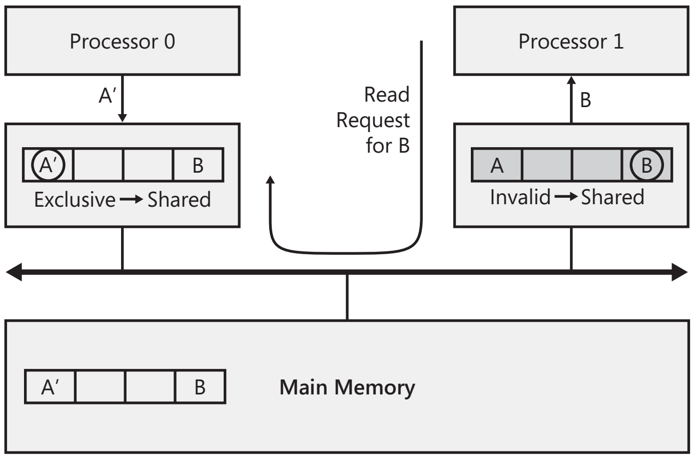


Figure 6-5 Processor 1 subsequently reading memory location B in a shared cache line

The point of this discussion is to illustrate how typical multiprocessing resource conflicts can impact scalability as more and more processors are added to a multiprocessing configuration. These and other conflicts on multiprocessors over access to shared resources tend to cause delays that increase in number and in duration as the number of parallel processing threads increases. Moreover, in a complex multithreaded application, it is often quite difficult to determine precisely where the resource bottlenecks causing performance degradation exist.

Optimizations to Improve Multiprocessor Scalability

Several features of the Windows Server 2003 operating system are designed to enhance the scalability of large-scale multiprocessors. This section discusses the mechanisms built into the operating system that function automatically to optimize performance on multiprocessors. The section that immediately follows this one discusses optional tuning strategies that you can implement to improve multiprocessor scalability even further for your specific workloads.

Processor affinity Because cache performance is so important, the Windows Server 2003 Scheduler attempts to dispatch a thread on the same processor it was dispatched on last time. Given that the breed of processors available at the time of writing contains copious amounts of Level 1, 2, and sometimes Level 3 cache, when a Ready thread is redispatched on the processor it last ran on, the likelihood is good that some of its code and data are still resident in the caches. If the processor caches do retain some of the data and instructions from the last execution, the thread will experience significantly higher instruction execution throughput, consistent with a *warm start* in cache. When switching to a new processor, the thread will face a *cold start* in cache, reducing instruction execution throughput for some initial period until the cache is loaded with the working set of the thread.

A thread is said to have a *soft* processor affinity for the specific processor on which it was dispatched last, called its *ideal processor*. If the thread's ideal processor is not available, the thread can be dispatched on any other idle processor. Because the Scheduler also implements priority-based preemptive scheduling, if a lower priority thread is running on a Ready thread's ideal processor, the higher priority thread will preempt the lower priority one.

It is also possible to restrict the threads of a processor so that they are dispatched only on a specific subset of the processors available. This is known as *hard processor affinity*, a tuning strategy associated with *partitioning*. Both hard processor affinity and partitioning are discussed later in the section entitled "Multiprocessor Configuration and Tuning Strategies."

Queued spin locks Kernel synchronization services that support queued spin locks are available in both Windows Server 2003 and Windows XP. In regular spin locks, all waiting threads test the same lock word in memory. When the lock finally becomes available, each thread that was waiting for the lock has to refresh its value of the lock word in its private cache. This creates a burst of memory operations on the shared memory bus that slows down instruction execution. Queued spin locks solve that problem because each waiting thread tests its own local per-processor lock word variable. The operating system also ensures that each per-processor local lock word is on a separate cache line from the other lock words associated with the lock. Additional overhead is associated with setting up and maintaining a queued spin lock, so this approach is not always the right choice for a device driver. But almost any heavily contested lock will perform better on a large-scale multiprocessor when queued spin locks are used instead of conventional ones.

More Info For more information about using queued spin locks inside device drivers designed for Windows Server 2003, see the *Device Driver Development Kit (DDK)* documentation at http://msdn.microsoft.com/library/en-us/kmarch/hh/kmarch/Synchro_7cc46160-bbcd-416f-98ea-d41bf80516eb.xml.asp.

Idle loop On a single processor machine, when there is no work for the processor to do, there is no harm in executing an idle loop until an external interrupt occurs that signals real work is available. On a multiprocessor, it is important that the code executing inside an idle loop not generate any shared memory bus requests or produce any other side effects that could impact scalability. The idle loop in Windows Server 2003 is engineered with multiprocessor scalability in mind. No instructions are executed that generate memory bus requests inside the idle loop.

Assessment of Multiprocessor Scalability

Adding processors to a computer system increases instruction execution throughput capacity as a function of the number of processors. Ideally, the performance improvement should increase *linearly* as a function of the number of processors. In practice, shared memory multiprocessors encounter unavoidable issues that prevent their performance from scaling linearly. Pipeline stalls as a result of serialization instructions are one unavoidable problem. The snooping protocol used to maintain consistent and coherent private processor caches is another multiprocessor effect that limits scalability. In addition, more processor cycles get wasted during the execution of spin lock code as the number of processors increases.

Because of delays associated with resource sharing, the same sequence of instructions will execute slower on a multiprocessor than it will on a single processor. Figure 6-6 shows execution time measurements for a multithreading program executed on 1-, 2-, and 4-processor systems. It shows how locking and synchronization code run rapidly on a single processor because there is no contention. The same code takes much longer to execute when multiple threads are active on separate processors because there is resource contention. The code that accesses shared data runs slower, too, because of problems like false sharing that cause contention for cache memory. Even code that accesses nonshared data runs a bit slower in this example.

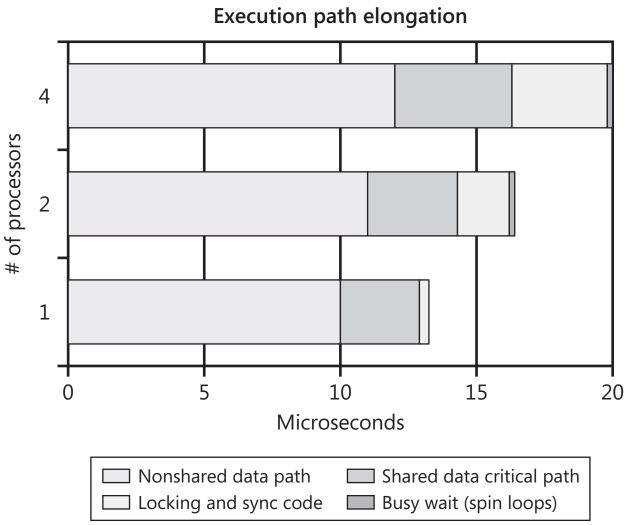


Figure 6-6 Measurements of the execution time of a multithreading program

TPC Performance Benchmarks

The Windows Server 2003 operating system has proven multiprocessing capabilities designed to take full advantage of the hardware's capabilities. Ample evidence to attest for these capabilities can be found in many published Transaction Performance Council (TPC) performance benchmarks at that organization's Web site, http://www.tpc.org. The TPC is a not-for-profit consortium of leading academic researchers and industry experts that takes the lead in defining representative database benchmark workloads and supervises the process of reporting benchmark results. Once the claims of the organization submitting the benchmark for compliance with the TPC rules are carefully audited, the results are posted on the TPC Web site. TPC publishes summary performance statistics that focus on the transaction throughput of the measured system running one of its specified benchmark workloads and the cost of the system's hardware and software, as well as a Full Disclosure report that details the full hardware and software configuration used to obtain them.

To assess Windows Server 2003 scalability, it is useful to review some of the results published for multiprocessor systems for the TPC-C benchmark, which reflects a generic transaction processing workload that utilizes a database of customer accounts to process inquiries, orders, payments, and other similar transactions. (The TPC-C specification is widely accepted as being a reasonably realistic representation of the sort of accounting debit-credit transaction processing systems that many commercial organizations employ.) TPC-C results primarily report transaction throughput metrics and the cost of the hardware/software configuration as a function of transaction throughput. Because published TPC-C results are scrupulously audited, they can also be relied upon to be objective. Here they are used to shed some light on the general issue of shared memory multiprocessor scalability.

Figure 6-7 shows the TPC-C transaction throughput reported for a series of benchmarks run on the same hardware and software base configuration in which the only independent variable is the number of processors installed. (The benchmarks use IA-32 hardware, running the Windows 2000 Server operating system. Unfortunately, published results for Windows Server 2003 clearly showing multiprocessor scalability were not yet available when this document was written.) The series identified as Vendor A reports results for identical 2, 3, 4, and 8-way configurations. This series is graphed using a heavy dark line. The four data points are marked with a diamond and labeled with their corresponding TPC-C measured transaction rate. A trendline is also shown that represents perfect linear scalability based on the measured throughput of the baseline 2-way machine. This is the ideal to which all multiprocessor systems aspire.

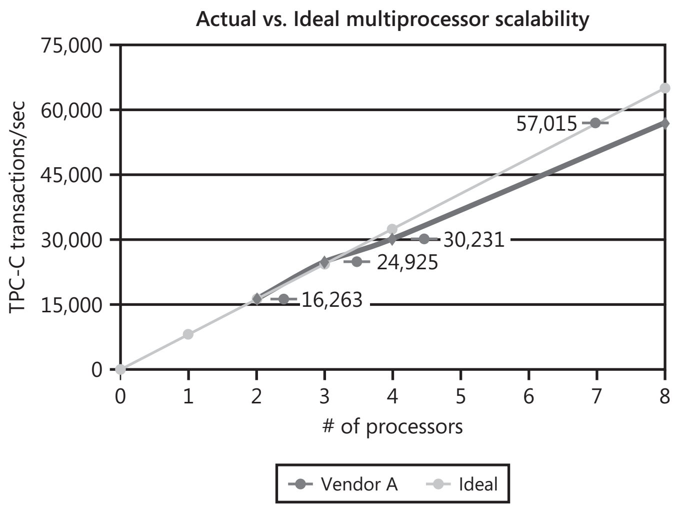


Figure 6-7 TPC-C transaction throughput reported for a series of benchmarks

Notice that when a third processor is added to the baseline 2-way configuration, transaction throughput scales linearly, improving by a factor of 50 percent. (Performance actually improves by a factor slightly better than 50 percent, which is a minor anomaly that is not overly surprising with a benchmark as complex as the TPC-C specification.) But when the fourth processor is added, the results clearly fall short of the ideal linear trend. Instead of a 33 percent improvement in transaction throughput, the 4-way system provides less than a 25 percent boost. The result for the 8-way system shows only a 90 percent improvement compared to the 4-way machine. Extrapolating from the 2-way system's transaction rate per processor, linear scaling would predict an ideal rate in excess of 65,000 transactions per second. The 8-way multiprocessor reported here can muster only 57,000 transactions per second. Still, this number needs to be put into perspective. Relative to the performance of the baseline 2-way system, the 8-way processor provides 250 percent more transaction processing throughput capability. This compares quite favorably to a 300 percent upper limit on the throughput that an ideal scalable multiprocessing architecture would provide. This is an impressive result, arguing for the cost-effectiveness of both Intel hardware and the Microsoft operating system software that is managing the multiprocessor platform.

The limitations of a simple multiprocessing architecture are apparent once the number of processors exceeds the capacity of the shared bus. When the bus shared by the processors saturates, continuing to add processors to the configuration is no longer cost-effective. This is why designers of large-scale parallel processing architectures explore alternative ways to scale bus capacity by interconnecting multiprocessor *nodes*. Nodes are building blocks that typically contain four or eight processors, usually configured with some amount of local memory. Scalability considerations for this type of highly parallel processing machine are considered in the "ccNUMA Architectures" section about cache coherent NUMA machines that Windows Server 2003 also supports.

Multiprocessor Configuration and Tuning Strategies

The hardware innovations and the operating system support for multiprocessor systems discussed earlier contribute substantially to improved scalability in Windows Server 2003, as compared to earlier versions of Intel hardware and Microsoft system software. However, the most important considerations impacting multiprocessor scalability are highly dependent on the specific characteristics of your workload. Applications that you intend to run on parallel processing hardware must be carefully crafted to achieve cost-effective scalability. For example, the TPC-C benchmark workload used here as a point of reference for comparing the relative effectiveness of large-scale hardware and software solutions is designed to support parallel processing. What are the essential characteristics that workloads like TPC-C have that enable them to scale in such parallel processing environments?

To take full advantage of multiprocessors, server applications must be multithreaded, that is, capable of processing many tasks in parallel. Inevitably, threads from the same application reach synchronization points where serial processing must be performed. To aid in multiprocessor scalability, serialization delays at synchronization points must be minimized.

How many threads should a multithreaded application create? If too few application threads are created, there might not be enough processing threads to expedite client requests. However, creating too many application threads might waste resources and increase serialization delays at application synchronization points. Because the optimal number of threads depends on both the kinds of hardware on which the application runs and the characteristics of the workload, knowing in advance how many threads to create is not easy. Many multithreaded server applications use *thread pooling* techniques for flexibility and to enhance scalability. Server applications that implement thread pooling are discussed in this section. Also discussed are the set of controls provided by the applications for configuring the thread pool and the measurements these applications gather, which are designed to help you set these controls correctly.

As discussed earlier, shared memory multiprocessors cannot be expected to scale linearly. As the number of processors increases, serialization delays, pipeline stalls resulting from cache conflicts, and other multiprocessor effects tend to degrade performance. Running symmetric multiprocessors, a situation in which any application thread can be dispatched on any processor, is not always optimal. *Partitioning* strategies, in which some application threads are restricted to running only on certain processors, can sometimes provide improved performance. Considerations for setting up asymmetric partitioning on large-scale multiprocessors are also discussed in this section.

Minimizing serialization delays One form of synchronization a multithreaded application performs is designed to protect critical sections of code that must be processed serially. As noted earlier, it is important to identify synchronization points in any parallel processing application where independent processing threads are forced to serialize their access to shared resources. Serialization delays at these choke points in your application tend to grow progressively longer as additional parallel processing threads are launched. Just as the Windows Server 2003 developers work hard to identify potential serialization bottlenecks in the operating system code that must execute in a parallel processing environment, your application developers must likewise mount an effort to identify and remove obstacles to multiprocessor scalability. Processor utilization profiling tools like Kernrate, discussed in Chapter 5, "Performance Troubleshooting," along with profiling tools available from other vendors, are extremely useful in this context.

Tip Never rely solely on anecdotal information from developers regarding where parallel processing choke points exist in the applications these developers are responsible for. Always try to acquire data on processor utilization from profiling tools like Kernrate to confirm and support their speculations. Even the most experienced programmers can focus on areas of the code in which they spent the most time developing and debugging, not on the actual performance choke points in their programs. Only empirical data carefully gathered by profiling tools measuring performance on representative workloads can present an objective picture of what actually occurs when the code executes in a parallel processing environment.

Finding serialization points that can become bottlenecks as you increase the number of parallel processing resources is hard enough, but designing an application whose performance scales with the resources that you assign is an even more challenging task. Application threads executing in parallel must be able to utilize the processor power supplied. In addition, memory, disks, and network bandwidth must also be able to support a similar level of parallel processing. All the resources that an application thread needs must be available at a comparable level of parallelism. Moreover, the application must be able to utilize all these resources effectively in parallel.

Note A formal argument known as Amdahl's Law, originally proposed in 1967, suggests that the speeding up of a parallel processing application is constrained by the portion of the application that must run serially. Amdahl's Law breaks the total amount of time spent processing an application request into two parts; *q*, the part that can be processed in parallel; and *1–q*, the part that must be processed serially. By introducing parallel processing resources, *p*, the time it takes to process *q* can be reduced proportionally to *q/p*. As more and more parallel processing resources are applied and *p* grows large, *q/p* becomes an insignificant factor. However, parallel processing has no impact on the time it takes to perform the *1–q* serial portion of the application. No amount of parallel processing resources can reduce the amount of time needed to process *1–q*, so finding ways to minimize the amount of time processing in serial mode is always very important. Amdahl's formulation was originally proposed to justify building faster processors, but it came to be regarded as a succinct, general statement describing the obstacles to speeding up application response time using parallel processing.

Thread pooling An application process running on a multiprocessor needs to be multithreaded to be capable of taking advantage of parallel processing resources. This section discusses *thread pooling*, an approach to building multithreaded applications that can scale on a wide range of machines. Thread pooling is a feature that many server applications rely on to scale effectively on multiprocessors. Most of the server applications Microsoft has developed for Windows Server 2003 use some form of thread pooling. These include Windows Server 2003 file services, IIS, Microsoft SQL Server, Microsoft Exchange, and portions of the operating system itself. In addition, both COM+ and the .NET Framework package include generic thread pooling services that software developers can readily use to ensure that their applications scale effectively.

More Info For information about COM+ Just-In-Time activation and object pooling, see the section entitled "COM+ Services" which can be found in the *Platform SDK* at http://msdn.microsoft.com/library/en-us/cossdk/htm/services_toplevel_8uyb.asp.

This section discusses the key elements of a thread pooling server application architecture. It also highlights those measurements that should be provided by a thread pooling server application to help you configure it for optimal scalability on multiprocessors. Examples of those measurements and how to use them are also provided. This section further discusses how to configure and tune a thread pooling server application, as well as settings to let you determine how many threads to create in a thread pool.

For scalability across a broad range of hardware and workloads, server applications require a flexible method for increasing the amount of resources they can command as the processing load increases. One method for accomplishing this is to create a pool of Worker threads, then automatically increase or decrease the number of threads that are active, depending on the workload demand. Often, these Worker threads are dispatched from the pool of available threads as work requests are received by the server application, as depicted in Figure 6-8.

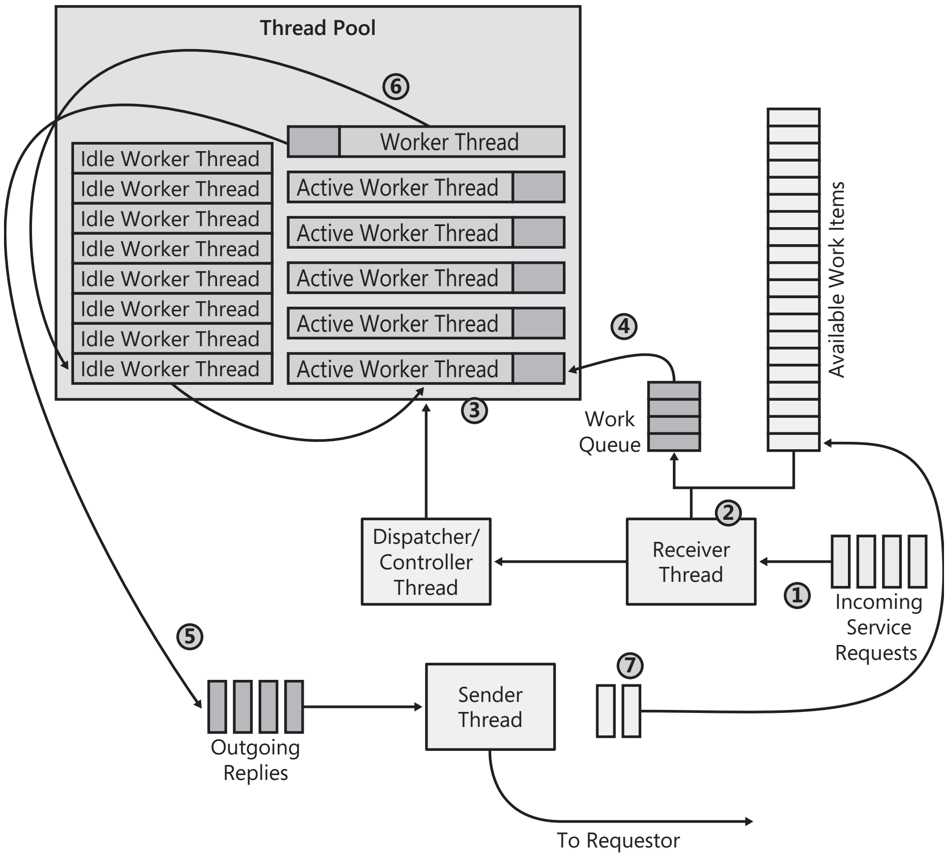


Figure 6-8 Worker threads are dispatched from the pool of available threads as work requests are received by the server application

As work requests arrive, the multithreaded server application releases processing threads, as indicated in Figure 6-8. The flow of work requests through a thread pooling server application proceeds as follows:

1. A newly arrived request wakes up a Receiver thread.
2. The Receiver thread immediately stores the work request into a work item from the available queue, queues the request, and signals a Dispatcher/Controller thread that new work has arrived.
3. The Dispatcher/Controller assigns an available Worker thread from the thread pool to the work item to begin processing the request. If no Worker threads are available and the application is not at its Max Threads setting, the Dispatcher/Controller creates an additional Worker thread.
4. The assigned Worker thread processes the request.
5. When a Worker thread completes its processing on behalf of some unit of work, it places the completed work item on the queue of outgoing requests, and posts the Sender thread to return data to the Requestor.
6. Prior to going back to sleep, the Worker thread that has just finished processing the work request checks the work queue to see whether any pending requests exist.
7. Finally, the completed work item is returned to the available work item queue. To facilitate reuse of the same threads and work items, the list of available work items and available threads are usually maintained as First In, First Out (FIFO) queues.

Usually, control-oriented Sender and Receiver threads are simple programs that concentrate on managing the input and output request queues. It is the Worker threads that perform the bulk of the actual application work request processing. Increasing the number of Worker threads created in the pool (and the number of available work items) is one way to ensure the application will acquire more resources on multiprocessors. To assist in scaling properly on multiprocessors, the initial number of threads the server application creates is often calculated as a function of the number of processors. There should also be tunable parameters that allow you to configure the thread pooling application properly for your workload and your hardware. Key tuning parameters include:

- The Worker thread dispatching priority
- The maximum number of Worker threads that will be created
- The number of work items allocated

Key measurements that will assist you in setting these tuning parameters correctly include:

- The rate at which work requests arrive
- The current number of active Worker threads
- The current number of queued requests
- The average request service time
- The average request queue time
- The current number of available (free) and occupied (busy) work queue items
- The number of times that arriving work was delayed or deferred because of resource shortages

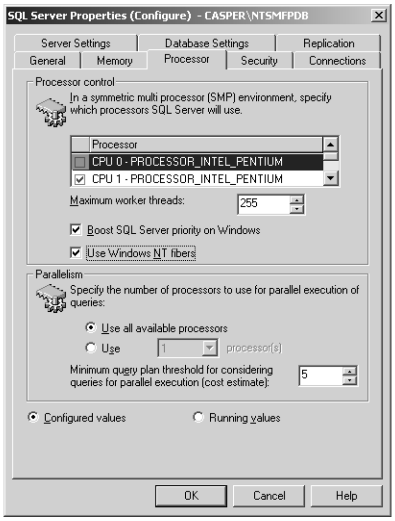
In addition, it is important to know how much % Processor Time is used to process requests on average. Normally, this can be computed by dividing the amount of processor time reported at the process level in the Process\% Processor Time counter by the request rate.

Microsoft SQL Server, IIS, ASP, and the network File Server service all use thread pooling. COM+ server and .NET Framework applications can also take advantage of built-in thread pooling facilities.

SQL Server SQL Server manages user connections dynamically using a pool of Worker threads. (SQL Server also has an option that allows you to use lighter-weight *fibers* instead of regular threads.) When SQL Server receives a set of SQL statements across a connection to process, the request processing is handled by assigning a thread from the thread pool. SQL Server allocates an initial set of worker threads based on the number of processors and the amount of memory that it is configured to use. As SQL requests are received, they are assigned to Worker threads from the pool. If no Worker threads are available, SQL Server creates more Worker threads, up to the limit specified by the *max worker threads* parameter. The default value for *max worker threads* is 255. The option can be changed using either the Enterprise Manager (illustrated in Figure 6-9) or the *sp-configure* stored procedure.

SQL Server does not report thread pool performance statistics, but you can monitor the Process(sqlserver)\Thread Count counter to obtain a close approximation of the number of Worker threads allocated. Transaction-level statistics are available in the SQL Statistics object. The rates of SQL Compilations/sec, SQL Re-Compilations/sec, and Batch Requests/sec, among others, are available. An overall rate for Transactions/

sec is available for each Database instance in the SQL Server Databases object. The current number of transactions being processed is also provided—the counter is called Active Transactions, and it is also from the SQL Server Databases object. The Logins/sec, Logouts/sec, and User Connections counters in the General Statistics object report on overall demand at a higher level. Unfortunately, SQL Server does not measure request service or response time.


SQL Server Properties (Configure) - CASPER\NTSMFPDB

Server Settings | Database Settings | Replication
General | Memory | Processor | Security | Connections

Processor control

In a symmetric multi processor (SMP) environment, specify which processors SQL Server will use.

	Processor
☐	CPU 0 - PROCESSOR_INTEL_PENTIUM
☑	CPU 1 - PROCESSOR_INTEL_PENTIUM

Maximum worker threads: 255

☑ Boost SQL Server priority on Windows

☑ Use Windows NT fibers

Parallelism

Specify the number of processors to use for parallel execution of queries:

(●) Use all available processors

( ) Use 1 processor(s)

Minimum query plan threshold for considering queries for parallel execution (cost estimate): 5

(●) Configured values ( ) Running values

OK Cancel Help

Figure 6-9 Changing the default value for max worker threads

> **Note** The names of the SQL Server performance objects vary depending on whether a single default instance of SQL Server is installed or one or more *named instances* of SQL Server are installed. For more information about installing named or multiple instances of SQL Server, see http://msdn.microsoft.com/library/en-us/instsql/in_runsetup_2xmb.asp. If a single default instance is installed, the SQL Server performance objects are prefixed with *SQL Server:*—for example, the SQL Server:General Statistics\Logins/sec counter. If named or multiple instances of SQL Server are installed, the prefix *MSSQL$* followed by the instance name is concatenated in front of the object instance name, as in MSSQL$VSDOTNET:General Statistics\Logins/sec.

SQL Server also provides an option to boost the priority of all SQL Server Worker threads. Because this option is normally used in conjunction with another option that allows you to also set a hard processor affinity for SQL Server threads, these two tuning options are discussed later in this chapter in the context of application-level settings. Table 6-2 summarizes the parameters used to control SQL Server thread pooling and the measurement data available to help with tuning this server application.

Table 6-2 SQL Server Thread Pooling Configuration and Tuning

Control	Parameter	Where Set/Usage Notes
Max Pool Threads	*max worker threads*	*sp-configure* stored procedure; Enterprise Manager GUI, SQL Server Properties tab.
		Default is 255 threads; optionally, lightweight fibers can be used instead.
Measurement	**Object\Counter**	**Usage Notes**
Current threads	SQL Server:Databases(*instance*)\Active Transactions	
	Process(sqlserver#*n*)\Thread Count	Includes all threads, not just Worker threads.
Request rate	■ SQL Server:SQL Statistics\SQL Compilations/sec ■ SQL Server:SQL Statistics\SQL Re-Compilations/sec ■ SQL Server:SQL Statistics\Batch Requests/sec	Each SQL request is allocated to a Worker thread.
	SQL Server:Databases(*instance*)\ Transactions/sec	Overall transaction rate.
Response time	Not Available	
Processor usage	*Calculate:* Process(sqlserver#*n*)\% Processor Time SQL Server:Databases(_*Total)*\ Transactions/sec	

File Server service The File Server service in Windows Server 2003 relies on thread pooling to scale effectively. File server processing is initiated by SMB requests issued by network client machines. The File Server service consists of a Kernel-mode component to field those requests, plus User-mode threads organized into dedicated thread pools per processor. SMBs usually request some file-oriented operation, which is then satisfied by reading or writing a local disk via the file cache. The File Server Kernel-mode component communicates with the File Server process using Context Blocks. A measure of overall activity is available using the Server\Context Blocks Queued/sec counter. However, it should be noted that Context Blocks Queued/sec includes both client-originated SMB requests and notifications regarding the completion of Kernel-mode disk I/O operations.

When network client SMB requests are received, they are assigned to an available work item from a Server Work Queue. The File Server service features dedicated work queues per processor to minimize the amount of time spent serializing access to these data structures on a multiprocessor. The File Server service creates one instance of a Server Work Queue for each processor. Each Server Work queue is accessed by its own dedicated thread pool, with *hard processor affinity* set to ensure that Worker threads are dispatched only on the specific processor associated with the Work queue. For example, a Kernel-mode thread processing an SMB request coming off the Network Interface stack executing on processor 2 places the request in a Context Block that is passed to the processor 2 Work Queue. Processing for the request will then be completed by a User-mode Worker thread that is executing on processor 2. Serialization delays while accessing the dedicated Work Queues are minimized because normally only Worker threads associated with a processor-specific Work Queue change the status of the queue. The use of dedicated per-processor Work Queues also reduces the amount of interprocessor signaling.

If one of the Work Queues is depleted before the others, available work items can be "borrowed" temporarily to avoid work item shortages. In addition, there is a single Blocking Queue in which work items in process are queued while waiting for disk I/O operations to complete. Having a provision that allows Work Items to be borrowed does mean that these Worker threads still must routinely lock and unlock the Work Queue whenever they access it. When no Work Items on other Work Queues are available to be borrowed, a Work Item shortage forces the File Server service to reject a client SMB request.

Several useful tuning options are available, and the operation of the thread pool is extensively instrumented. A *MaxThreadsperQueue* tuning parameter is available to establish an upper limit on the maximum number of threads created per processor Work Queue. The default number of Worker threads allocated per processor is 10. In addition, a *MaxRawWorkItems* parameter can be set to increase the number of Work Items allocated in the Nonpaged pool to avoid Work Item shortages. *MaxRawWorkItems* defaults are based on the size of RAM. For example, 125 Work Items are allocated by default on a machine with 512 MB.

Performance counters exist at the Server Work Queue level that report both the number of Active Threads and the number of Available Threads in the thread pool. When the total number of Worker threads is at or near the *MaxThreadsPerQueue* value, and the Server Work Queue(*#n*)\Queue Length counter starts to increase, you might want to consider increasing *MaxThreadsPerQueue*. Work Item Shortages can be avoided by allocating more Work Items, allocating more Worker threads, or a combination of the two. Table 6-3 lists the parameters you can use to tune File Server thread pooling and shows the performance counters that help you understand the performance of this thread pooling server application.

Table 6-3 File Server Service Thread Pooling

Control	Parameter	Where Set/Usage Notes
Max Pool Threads	*MaxThreadsper-Queue*	HKLM\SYSTEM\CurrentControlSet\Services\lanmanserver\parameters
		Defaults to ten per processor
Max Work Items	*MaxRawWorkItems*	Depends on the amount of RAM and the number of processors
Min Free Work Items	*MinFreeWorkItems*	Depends on the amount of RAM and the number of processors
Measurement	**Object\Counter**	**Usage Notes**
Current threads	Server Work Queues(*instance*)\Active Threads	Total Threads = Active Threads + Available Threads.
	Server Work Queues(*instance*)\Available Threads	
Resource shortages	■ Server \Work Item Shortages ■ Server Work Queues(*instance*)\Work Item Shortages ■ Server Work Queues(*instance*)\Available Work Items ■ Server Work Queues(*instance*)\Borrowed Work Items	Overall statistics and measurements broken down by dedicated processor work queue.
Request rate	■ Server\Context Blocks Queued/sec ■ Server Work Queues(*instance*)\Context Blocks Queued/sec	The best overall indicator of the activity rate, but not a true transaction request rate.
Requests Queued	Server Work Queues(*instance*)\Queue Length	Work Items queued that are not currently assigned to an active thread.
Response time	Not available	
Processor Usage	Process(svchost#*n*)\% Processor Time	Run `Tasklist /svc` to determine which svchost process contains the File Server service.

The File Server process that contains these thread pools and Work Queues is an instance of svchost. You can determine which svchost process is hosting the File Server service by using the Tasklist command; it helps you track down which process has Srvsvc.dll loaded:

```
C:\>tasklist /svc /fi "Modules eq srvsvc.dll"
```

This code will return information similar to the following:

```
Image Name                     PID Services
========================= ====== =============================================
svchost.exe                    872 Browser, CryptSvc, EventSystem, helpsvc,
                                   lanmanserver, lanmanworkstation, Netman,
                                   Nla, RasMan, Schedule, seclogon, SENS,
                                   ShellHWDetection, TrkWks, W32Time, winmgmt,
                                   Wmi, wuauserv, WZCSVC
```

Because the File Server service coexists with many other services inside svchost, it is difficult to determine precisely how much % Processor Time is associated with File Server activity. On a machine that is used as a dedicated File Server, however, assigning all or most of the Process(svchost#*n*)\% Processor Time to this function is reasonable. Keep in mind that Kernel-mode threads that are not associated with an svchost process are used to perform many File Server service functions, so this method of processor accounting will still come up short. It is difficult to determine precisely what overall File Server processor consumption is, except, of course, if the machine is mainly a dedicated file server.

IIS Thread Pooling Architecture Internet Information Services (IIS) is another thread pooling application. This section introduces the IIS version 6.0 architecture. It also discusses the thread pool used within IIS to service conventional HTTP GET and PUT Requests. The section that follows discusses the configuration and tuning of Web services applications written using either Active Server Pages (ASP) technology or ASP.NET.

The IIS version 6.0 architecture is illustrated in Figure 6-10. It shows that IIS consists of two major processing components and two cache storage components. HTTP Method calls are passed to the IIS Kernel-mode driver, Http.sys. Http.sys maintains a Kernel-mode cache in which it stores recently referenced HTTP Response objects. On a kernel cache hit, an Http.sys kernel thread can respond immediately to an HTTP GET Request, returning an HTTP Response object that already exists without a context switch. If the Response object does not exist or the request is for an ISAPI Extension application such as an ASP (Active Server Pages) or ASP.NET page, the Request is passed to a worker process to be processed. A Worker thread from the worker process is assigned to create an HTTP Response message. Multithreaded Web application worker processes in which ASP and ASP.NET requests are processed can be configured into multiprocess *Web gardens* for even greater flexibility and scalability.

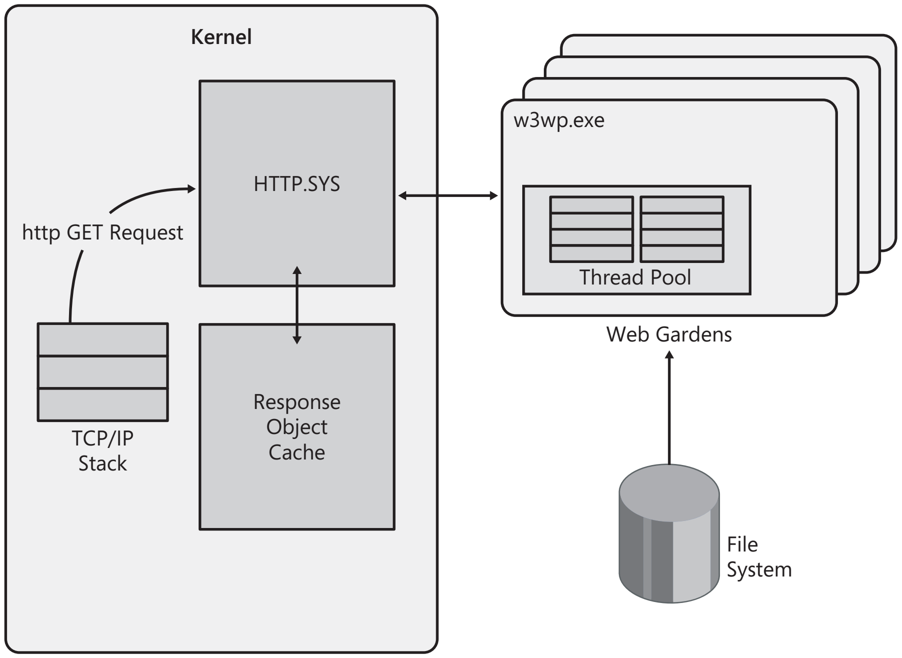


Figure 6-10 IIS consists of two major processing components and two cache storage components

For conventional Web server requests for HTML pages, JPEGs, GIFs, and other static HTML objects, the IIS 6.0 Kernel-mode components provide fast and efficient servicing. For processing Web service application requests using ASP.NET (or the older ASP technology), Web gardens provide an effective management framework in which applications can run in isolation and be controlled. Scalability options for Web applications include using both multithreading *and* multiple processes. If an application processing thread in one process hangs up, for example, the remaining worker processes in the Web garden are still available to handle incoming requests.

The Http.sys driver uses a thread pool to service incoming HTTP Requests, creating four worker threads per processor by default. If most incoming requests are for static HTML objects, a large number of Worker threads is normally not required, because Response messages can be generated so quickly. However, if your Web site needs to process a large number of CGI Requests concurrently, the size of the thread pool can be a constraint. Check the value of the Web Service\Current CGI Requests counter to determine how many CGI Requests your site is processing concurrently. You can increase the size of the IIS thread pool by setting *MaxPoolThreads* at HKLM\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters.

IIS can create additional pool threads up to the *PoolThreadLimit*, which is 2 for every MB of RAM installed. Monitor the Process(inetinfo)\Thread Count counter to determine the current number of active IIS threads. Additional counters in the Web Service object report the rate of HTTP GET and Post Requests/sec, as well other HTTP Method calls being processed. If Web Server throughput appears constrained, yet ample processor capacity exists, you might want to consider increasing the size of the IIS thread pool. Note that ISAPI Extension Requests are handled out-of-process, as discussed in more detail in the next section. Processing of ASP, ASP.NET, and other ISAPI Extension applications is not performed by the IIS internal thread pool. Table 6-4 summarizes the parameters used to control IIS thread pooling and the measurement data that is available to help with tuning decisions.

Table 6-4 IIS Web Service Thread Pooling

Control	Parameter	Where Set/Usage Notes
Max Pool Threads	*MaxPoolThreads; PoolThreadLimit*	HKLM\SYSTEM\CurrentControlSet\Services\inetinfo\parameters.
		The *PoolThreadLimit* is 2 per MB of RAM. *MaxPoolThreads* defaults to 4.
Measurement	**Object\Counter**	**Usage Notes**
Current threads	Not available	
Request rate	■ Web Service\Total Method Requests/sec ■ Web Service\Get Requests/sec ■ Web Service\Post Requests/sec ■ Web Service\CGI Requests/sec	Total Method Requests/sec is an overall activity rate. Note: ISAPI Extension Requests are handled out-of-process.
Response time	Not available	
Processor usage	Not available	

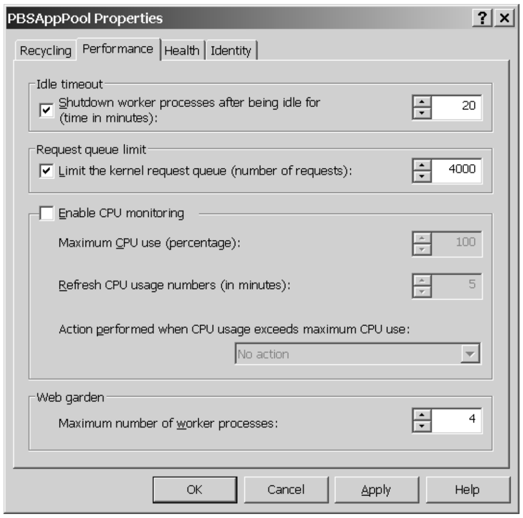
The Web Server Kernel-mode component Http.sys is likely to perform much of the work of processing conventional HTTP GET requests, and its processor consumption is not broken out separately. Only HTTP GET Requests that cannot be satisfied directly from the Kernel-mode cache are routed to a User-mode process for processing. If the Kernel-mode cache is effective, most processing of static HTML requests will be performed in Kernel mode. By comparing the rate of Web Service Cache\URL Cache Hits/sec to the overall rate of Web Service\HTTP GET Requests, you can determine how much processing is being performed in Kernel mode relative to the User-mode processes.

IIS as an application server Because support for generic thread pooling is such a valuable technique for achieving server application scalability goals, it is built into the application server run time features of IIS version 6. When IIS is used an application server, ASP and ASP.NET applications are often performing critical business functions. To ensure that ASP and ASP.NET applications scale well on multiprocessors, IIS provides a series of advanced configuration and tuning options for this environment.

More Info For more information about IIS scalability, see the *Internet Information Services (IIS) 6.0 Resource Kit* (http://www.microsoft.com/downloads/details.aspx?FamilyID=80a1b6e6-829e-49b7-8c02-333d9c148e69&DisplayLang=en), especially Chapters 6 and 7.

Unless you are running your Web server in worker process isolation mode, all ASP and ASP.NET processing is performed in one or more separate application processes known as *Application pools*. Each Application pool can be configured to use multiple worker processes, which are known as Web gardens. Each worker process in a Web garden, an instance of either the W3wp or aspnet_wp process, is a separate thread pooling application. You assign the number of worker processes for each application pool using the IIS Manager, as illustrated at the bottom of Figure 6-11. Within each worker process, the *AspProcessorThreadMax* Metabase setting determines the maximum number of Worker threads that IIS can create. The default value of *AspProcessorThreadMax* is 25 per processor.

Note The default value of *AspProcessorThreadMax* under IIS 5.0 was 10 per processor.


PBSAppPool Properties

Recycling | Performance | Health | Identity

Idle timeout

[✓] Shutdown worker processes after being idle for (time in minutes): 20

Request queue limit

[✓] Limit the kernel request queue (number of requests): 4000

[ ] Enable CPU monitoring

Maximum CPU use (percentage): 100

Refresh CPU usage numbers (in minutes): 5

Action performed when CPU usage exceeds maximum CPU use: No action

Web garden

Maximum number of worker processes: 4

Figure 6-11 Using the IIS Manager to assign the number of worker processes for an application pool

The combination of run-time settings that Web gardens provides allows you to control both the number of processes devoted to application processing and the number of Worker threads in each process. These settings provide a flexible range of configuration options to achieve your scalability goals for Web-based applications. Web gardens also provide important run-time services that will help you meet your availability and uptime requirements for important Web services applications. However, when you do not configure multiple worker processes, the *AspProcessorThreadMax* setting can serve as a constraint on throughput that you might have to monitor carefully. Adding the Active Server Pages\Requests Queued and Active Server Pages\Requests Executing counters gives you an instantaneous measure of ASP processing concurrency, which you should compare with *AspProcessorThreadMax*.

If the Request Queue backs up, relieving the *AspProcessorThreadMax* limit on the number of processing threads is likely to improve performance, absent other resource constraints like processor and memory capacity. As the IIS Administrator, you face complex configuration decisions; the Web application programs you are running might also be quite complex. Web applications might rely on processing by COM+ or .NET Framework components, where common business logic that the applications use is often consolidated for ease of maintenance. (The full range of COM+ and .NET Framework scalability features is beyond the scope of this discussion.) Web applications might also access back-end databases where a permanent store of interactive session data is maintained. COM+ server components can be configured to be accessed locally or remotely using DCOM. Similarly, database connections can be made to local machines or to remote ones. Understanding the impact of any of this additional out-of-process processing on ASP or ASP.NET Request execution time is a complicated task.

Tip If the number of Requests Queued is more than 1 or 2 per processor, and Requests Queued plus Requests Executing is at or near the value for *AspProcessorThreadMax*, consider boosting *AspProcessorThreadMax*, increasing the number of worker processes, or using some combination of both. Note that if the W3wp or aspnet_wp worker processes face virtual memory constraints, defining additional worker processes in the Web garden is usually the best course of action.

IIS will also shut down idle worker processes in a Web garden after some number of minutes of activity, as illustrated at the top of Figure 6-11. This means that the Process(w3wp)\Thread Count counter is also a useful way to measure the number of active threads.

The Active Server Pages\Requests/sec counter reports the arrival rate of requests. ASP application processing to generate dynamic responses to HTTP GET Requests is performed by scripts. Once ASP scripts are interpreted and turned into executable code, these scripts can be cached as Script Engines and reused. In contrast, ASP.NET processing is performed by compiled .NET Framework programs. Effective Script Engine caching for ASP applications can have a significant impact on processor utilization and overall execution time. Reducing execution time also means that ASP requests occupy Worker threads for less time, which reduces the number of Worker threads required. The maximum number of ASP requests that can be executing or queued is governed by the *AspRequestQueueMax* setting, which is 3000 by default. The Active Server Pages\Requests Rejected counter keeps track of Requests rejected when the *AspRequestQueueMax* limit is reached.

ASP response time measurements Active Server Pages is one of the few server applications instrumented to report response time using System Monitor counters. However, these response time measures–Active Server Pages\Request Execution Time and Active Server Pages\Request Wait Time–must be interpreted using extreme caution. In essence, both report random samples of ASP Request service time and queue time gathered once each collection interval.

Caution Active Server Pages\Request Execution Time and Active Server Pages\Request Wait Time counters represent the Execution Time and Wait Time in milliseconds of the last ASP request only. They are properly viewed as sampled values, which might or might not be representative of overall application response time.

Using Little's Law, which was discussed in Chapter 1, "Performance Monitoring Overview," you can estimate the average response time of Active Server Pages using the following formula:

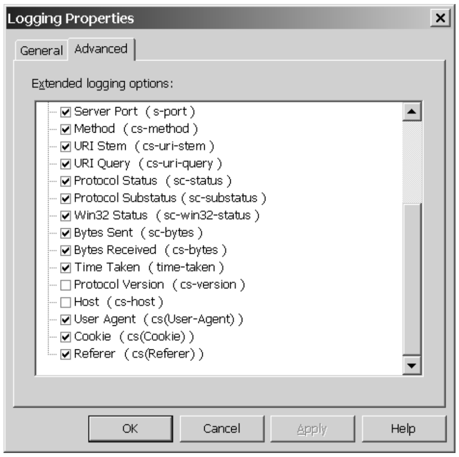
```
ASP Average Response Time =
(Active Server Pages\Requests Queued + Active Server Pages\Requests Executing)  Acti
ve Server Pages\Requests/sec
```

This calculation should be viewed as an *estimate* of the application response time. ASP Requests/sec is a continuously measured difference counter, whereas Requests Queued and Requests Executing are instantaneous values. As long as the instantaneous measurements of Requests Queued and Requests Executing are not dramatically different from the previous measurement interval, Little's Law will hold. The other validity requirement this calculation should meet is that at least 50–100 response time measurement events occur per data collection interval. In other words, for a Requests/sec rate of 1 or 2, a 1-minute measurement interval will normally provide reasonable estimates. It is always useful to compare the average ASP response time calculated in this fashion with sampled values reported for the Active Server Pages\Request Execution Time and Active Server Pages\Request Queue Time counters. Table 6-5 summarizes the parameters used to control Active Server Pages thread pooling and the measurement data available to help with tuning your ASP applications.

Note The IIS server-side response time of all HTTP Response messages is available from the IIS log. On the Web site tab of the Web Site Properties Pages, choose W3C Extended Log File Format logging, and configure the log file properties to report the Time Taken field, as illustrated in Figure 6-12. The Time Taken field in the Web log is reported in milliseconds.

Table 6-5 Active Server Pages Thread Pooling

Control	Parameter	Where Set/Usage Notes
Max Pool Threads	*AspProcessor-ThreadMax*	IIS Metabase
		The *AspProcessorThreadMax* is 25 per processor.
Max Queue	*AspRequestQueue-Max*	The *AspRequestQueueMax* is 3000 by default.
Measurement	**Object\Counter**	**Usage Notes**
Current threads	■ Process(w3p3)\Thread Count ■ Active Server Pages\Requests Queued ■ Active Server Pages\Requests Exe-cuting	Instantaneous counters.
Resource shortages	Active Server Pages\Requests Rejected	
Request rate	Active Server Pages\Requests/sec	
Requests queued	Active Server Pages\Requests Queued	
Response time	*Calculate:* Active Server Pages\Request Execution Time + Active Server Pages\Request Wait Time	Time in milliseconds for the last request completed.
Processor usage	*Calculate:* Process(w3wp)\% Processor Time Active Server Pages\Requests/sec	



Extended logging options:

- ☑ Server Port (s-port)
- ☑ Method (cs-method)
- ☑ URI Stem (cs-uri-stem)
- ☑ URI Query (cs-uri-query)
- ☑ Protocol Status (sc-status)
- ☑ Protocol Substatus (sc-substatus)
- ☑ Win32 Status (sc-win32-status)
- ☑ Bytes Sent (sc-bytes)
- ☑ Bytes Received (cs-bytes)
- ☑ Time Taken (time-taken)
- ☐ Protocol Version (cs-version)
- ☐ Host (cs-host)
- ☑ User Agent (cs(User-Agent))
- ☑ Cookie (cs(Cookie))
- ☑ Referer (cs(Referer))

Figure 6-12 Configuring Logging Properties to report the Time Taken field

ASP.NET .NET Framework applications can implement thread pooling using the *ThreadPool* class. The thread pool is created automatically the first time the .NET Framework program calls the *ThreadPool.QueueUserWorkItem* method. The common language runtime (CLR) also creates a control thread automatically that monitors all tasks that have been queued to the thread pool. The *maxWorkerThreads* and *maxIOThreads* attributes in the <processModel> node of the Machine.config configuration file set an upper limit on the number of threads in the process. The default for *maxWorkerThreads* is 20 per processor. If all active threads in the pool are continuously busy, but work requests are waiting in the queue, the CLR will create another Worker thread. However, the number of threads will never exceed the maximum value specified by the *maxWorkerThreads* attribute.

ASP.NET applications are configured for run time using Configuration files. The processModel section of an ASP.NET application Configuration file generally maps to IIS 6.0 run-time settings for Web Gardens' threading, and other performance options. These Configuration files provide even greater flexibility in setting up thread pooling applications than the IIS 6.0 controls.

More Info See the section entitled "Mapping ASP.NET Process Model Settings to IIS 6.0 Application Pool Settings" which is found at http://msdn.microsoft.com/library/en-us/cpguide/html/cpconaspnetprocessmodelsettingequivalencetoapplicationpoolsettings.asp in the *.NET Framework SDK* for more information about Configuration files.

For ASP.NET applications, an ASP.NET\Requests Current counter is an instantaneous measure of how many ASP.NET requests are in execution or waiting to execute. When ASP.NET\Requests Current exceeds the *requestQueueLimit* value defined in the processModel configuration, ASP.NET Requests are rejected. The performance counters in the .NET CLR LocksAndThreads objects provide additional statistics on application threading during run time. These statistics include the number of physical threads that are currently active and the rate of lock contention among threads within the application thread pool. For example, the Contention Rate/sec counter reports the rate at which threads in an application thread pool attempt to acquire a managed lock unsuccessfully.

To help you deal with this extra flexibility, the .NET Framework provides application-level counters for each ASP.NET application. By application, you can monitor ASP.NET Application(*instance*)\Requests Executing and ASP.NET Application(*instance*)\Requests in Application Queue, which are instantaneous measurements comparable to ASP Requests Executing and Requests Queued.

There are no comparable ASP.NET Request Execution and Request Wait Time counters that sample application response time. However, because ASP.NET Application(*instance*)\Requests/sec is available, Little's Law can again be used to estimate application response time:

```
ASP.NET Average Response Time =
(ASP.NET Application(instance)\Requests in Application Queue +
 ASP.NET Application(instance)\Requests Executing)
ASP.NET Application(instance)\Requests/sec
```

The same cautionary words made earlier about the validity of this calculation in the context of ASP apply here. This computed value should be viewed as an estimate of the application response time only. The ASP.NET Requests/sec is a continuously measured difference counter, whereas Requests in Application Queue and Requests Executing are instantaneous values. As long as the instantaneous measurements of Requests in Application Queue and Requests Executing are not dramatically different from the previous measurement interval, Little's Law will hold. Another validity requirement that you should be careful to satisfy is making sure there are at least 50–100 requests in the interval. In other words, for a Requests/sec rate of 1 or 2, a 1-minute measurement interval will normally provide reasonable response time estimates. Unlike ASP, ASP.NET does not report sampled values for request service and queue time. Again, the Time Taken field in the log provides a measure of Web application response time you can validate against. Table 6-6 summarizes the parameters used to control ASP.NET application thread pooling and the measurement data available to help with tuning your ASP.NET applications.

Table 6-6 ASP.NET Thread Pooling

Control	Parameter	Where Set/Usage Notes
Max Pool Threads	*maxWorker-Threads; maxIO-Threads*	The *processModel* section of Machine.config.
		The *maxWorkerThreads* and *maxIOThreads* parameters both default to 20 threads per processor.
Max Queue	*requestQueueLimit*	The *requestQueueLimit* is 5000 by default.
Measurement	**Object\Counter**	**Usage Notes**
Current threads	■ ASP.NET Application(*instance*)\Requests Executing ■ ASP.NET Application(*instance*)\Requests in Application Queue	Instantaneous counters.
Request rate	ASP.NET Application(*instance*)\Requests/sec	
Requests queued	ASP.NET Application(*instance*)\Requests in Application Queue	
Response time	*Estimate:* (ASP.NET Application(*instance*)\Requests in Application Queue + ASP.NET Application(*instance*)\Requests Executing) ASP.NET Application(*instance*)\Requests/sec	From Little's Law.
Processor Usage	*Calculate:* Process(w3wp)\% Processor Time ASP.NET \Requests/sec	Overall application processor usage only, but see provision for CPU Monitoring.

Guidelines for configuring thread pooling server applications Because so many server applications benefit from a thread pooling architecture, it is useful to articulate a general approach to configuring and tuning these applications on multiprocessors. Thread pooling provides the flexibility that high-performance server applications require to scale across a wide range of multiprocessing machines. Rather than use some static number of Worker threads to process client requests, thread pooling applications begin with a relatively small number of threads, and then ramp up the number of Worker threads when work requests arrive and no Worker threads are available to process them.

Ideally, the number of available Worker threads that are allocated will eventually stabilize when enough threads exist to process all client requests in a timely manner. However, you need to watch out for out-of-capacity conditions in which the application can never reach that equilibrium state. If a resource bottleneck is constraining the scalability of the thread pooling application, you need to identify that bottleneck and remove it. The second situation to watch out for is one in which the upper limit on the size of the thread pool serves as an artificial constraint on application throughput.

When there is a resource shortage, employ the bottleneck detection methods and procedures discussed earlier in this book. See Chapter 5, "Performance Troubleshooting," for identifying and relieving processor, memory, disk, and networking resource bottlenecks on servers that are experiencing performance problems. If a thread pooling application is running on a machine that is resource-constrained, application throughput will stabilize, even as the number of Worker threads increases and the pool of work items, in which pending queued requests are parked, is depleted. The measurements of utilization and queuing at the bottlenecked resource should show signs of both resource saturation and queuing delays.

Lock collisions On multiprocessors, thread pooling applications can encounter another type of resource constraint that can limit scalability. The necessity to lock critical sections of shared code and data areas to preserve their integrity on a multiprocessor can lead to an excessive number of *lock collisions* as the number of Worker threads increases. As the number of concurrently running threads increases, so does contention for critical sections and locks. Sometimes this excessive rate of lock collisions will appear as code path elongation, as illustrated in Figure 6-6 previously. (This code path elongation will show up clearly when you run the server application under a controlled load and monitor it using a profiling tool like Kernrate, which was described in Chapter 5, "Performance Troubleshooting.") For server applications that implement spin locks, an increase in lock collisions will increase the average CPU consumption per request. In other words, processor utilization will increase, while application throughput will remain the same or might even degrade.

As noted earlier, the rate of lock collisions for .NET Framework applications can be monitored directly. For SQL Server database applications, you can also monitor the counters in the SQL Server: Locks objects, including Lock Wait Time (ms), Lock Timeouts/sec, and Number of Deadlocks/sec, which are available for each database instance.

If lock collisions are constraining application throughput, boosting the number of Worker threads is likely to make the condition even worse.

Max threads There is an upper limit to the number of Worker threads the application will create. This limit is designed to prevent the thread pool from growing out of control when some resource bottleneck is constraining application throughput and client requests continue to arrive. Without this upper limit, the size of the thread pool would grow infinitely large whenever the request queue started to back up.

In the absence of any other manifest resource bottleneck, the application's max threads parameter can constrain performance. The default settings for the server applications previously discussed are all quite generous, so this situation is relatively rare. Still, it does happen in certain circumstances. Fortunately, you can learn to recognize this situation, and rectify it by setting the appropriate tuning parameter to increase the maximum number of Worker threads that can be created in the pool.

The server thread pool max threads parameter can be a constraint when the following conditions hold:

- The application is running at or near its current max threads limit.
- The number of items waiting in the work queue is greater than one per processor.
- The processors are less than 60 percent utilized.

Under these circumstances, increasing the number of Worker threads might result in improved performance and scalability. If the maximum threads limit is a constraint, increasing the maximum threads limit will result in additional Worker threads being created to handle peak loads. Application throughput should also increase when additional Worker threads are dispatched. To guard against instances in which lock collisions increase when the concurrency level increases, you should also calculate the average processor time per client request. If the processor time per request increases while throughput rates remain stable, you will want to reduce lock collisions by reverting to the earlier maximum threads setting.

If the bottleneck in the application is elsewhere—in the disks, in the database, in the network, and so on—raising the maximum threads limit should have little or no apparent effect on throughput.

When the machine is not otherwise constrained, increasing the number of threads might increase the amount of useful work that gets done, assuming there is also sufficient unused processor capacity. If, as a result of increasing the number of available Worker threads, the application's throughput increases, you are on the right track. It might seem counterintuitive, but you can then continue to increase the number of threads until the processor starts to saturate. What can happen at that point, though, is that other workloads on the same system can suffer because the processors are overloaded. That consideration forces you to throttle back the thread pooling application so that it does not overwhelm all other work on the system.

Increasing the size of the thread pool can also backfire, especially when the number of concurrently executing Worker threads is not acting as a constraint on application throughput. In a typical scenario, Worker threads run for only a short time before entering a voluntary Wait state, during which they are waiting for some event, lock, or other resource. In that case, adding more threads can increase CPU utilization without getting more useful work done. That is because more threads accessing the same data causes more lock collisions to occur. Even if there are no more collisions, having lots of threads increases the likelihood that the cached state of a running thread that is interrupted will be pushed out of processor cache by the other Worker threads before that thread ever gets to run again.

You can also have too many excess threads defined. Idle threads are not using the processor, but they still occupy resources like memory that could be in short supply. To save on the overhead of thread creation, an existing thread can be used instead if it is available. Normally, threads created during a period of heavy demand are retained for some period of time afterwards, even though they are idle. Keeping a few additional idle Worker threads alive is more efficient than having your application constantly create and destroy threads.

Concurrency levels As a capacity planner, you are faced with a bewildering set of choices when you set about configuring a multiprocessor to process your transaction workload. You might have to select both the number and the speed of the processors that you are planning to buy. You might have to choose between simple, shared memory multiprocessors and more complex, large-scale parallel ccNUMA machines. You will also have to buy enough memory and install enough disks and network bandwidth to ensure the entire configuration scales, not just the processors.

Assuming care has been taken to minimize the amount of time server applications spend in serial mode processing, the requests of individual transactions can be processed independently of each other in a highly parallel manner. Because parallel requests can be processed concurrently on parallel processing resources, a parallel processing solution is likely to be effective. Understanding the level of concurrency of your application will help you figure out what kinds of parallel processing resources to apply to it.

An analytic approach views the application as an M/M/n queuing system, as discussed in Chapter 1, "Performance Monitoring Overview." The optimal number of processing threads depends on the rate at which work arrives, the average amount of time a Worker thread spends processing a request, and the capacity of the machine to perform the work involved. (In this section, we are concerned only with CPU resources, but all the required memory, disk, and networking resources need to be considered.) Consider that Little's Law, which was also discussed in Chapter 1, provides a way to estimate the concurrency level of your application when you are not able to measure it directly:

$$Q = \lambda \times RT$$

Little's Law states that the *average* number of requests in the system is the product of the arrival rate of requests and the average response time of those requests. The average number of requests, either in service or queued for service, is a useful measurement of concurrency. This measurement also reveals the potential for parallel processing to speed up your workload. Assuming a processing thread is required for every incoming transaction request for the duration of the processing of that request, then, on average, Q processing threads are required. If the average number of requests in the system, either in service or queued for service, is two, for example, having eight processors available to process those requests probably means you have excess capacity and underutilized resources.

Of course, because the arrival rate distribution for requests is likely to be bursty, not uniform, there are bound to be times when having more than Q processing threads available is a good idea. How many additional threads are desirable? Unfortunately, Little's Law can tell you only about averages. Another complication is that your application might need to process a mix of compact transactions, which take only a few resources to process, and long running transactions, which occupy resources for much longer periods of time. Variability in the service time distribution can further complicate the problem of matching application processing threads to the available resources.

The Utilization Law, also discussed in Chapter 1, “Performance Monitoring Overview,” offers another useful perspective on your application’s level of concurrency. The Utilization Law states that the utilization of a resource is the product of the arrival rate of requests and the average service time of those requests:

$U = \lambda \times ST$

Consider a transaction request that requires 200 milliseconds of processor time (or 20 percent processor time) to complete. If five such transactions are executed per second, the processor will be 5 × 20 percent or 100 percent utilized. The Utilization Law implies an upper limit on the number of application-processing requests necessary to saturate the resource:

threads = # of processors ÷ average transaction processor utilization

Defining more threads than that would purely be for the purpose of queuing requests internally, because the saturated processors would not be capable of processing any more work. Again, you should expect that the arrival rate and service time distributions are bursty. You need to be flexible when using this calculation to set an upper limit on the number of threads that you allow the application to create.

Asymmetric partitioning Another important tuning option for system administrators is *partitioning* large-scale multiprocessors, a task that might help these multiprocessors better handle heterogeneous workloads. As discussed earlier, shared-memory multiprocessors face serious scalability hurdles. As more processing engines are added to the system, the efficiency of the overall system diminishes because of the overhead of maintaining cache coherence, lock contention, and other factors. Asymmetric partitioning of the processor means restricting threads from some applications so that these threads can run only on a subset of the available processors. Partitioning the machine can reduce lock contention and the overhead of maintaining cache coherence.

Partitioning can be a useful technique for machines with four, eight, or more processors, especially when you want to consolidate multiple workloads on a single-large machine. For both ease of administration and performance, for example, it is advantageous to consolidate Active Directory and domain controllers with machines performing file and print services so that each new redirected file system request does not require authentication from a remote Active Directory machine. Running a messaging application like Microsoft Exchange on fewer large-scale machines might result in reduced administration costs and less complex replication processing to manage. Partitioning potentially makes these mixed workloads more manageable.

Caution Many advocates of server consolidation argue that administering fewer servers lowers the cost of server administration. Hopefully, you will lower many of your administrative costs using Windows Server 2003 by reducing the number of servers that you run. There would be, for example, fewer machines requiring your attention for both performance monitoring and capacity planning.

Nevertheless, you should be aware that in the crucial matter of performance monitoring and capacity planning, when you operate fewer, larger-scale consolidated machines, each machine might require more work to configure and tune, lending greater importance to implementing the performance monitoring procedures discussed in Chapter 4, "Performance Monitoring Procedures." Running larger machines with a mixture of workloads, you must ensure that adequate service is provided to each of those workloads. Tools like the Windows System Resource Manager (WSRM), discussed in this section, make managing large heterogeneous workloads much easier.

Partitioning offers a way to carve up large-scale multiprocessors into two or more logical machines that can be managed more efficiently than an undifferentiated n-way server machine. Processor partitioning is accomplished by assigning threads from specific processes to specific processors using *hard* processor affinity. Hard processor affinity is implemented using the *SetProcessAffinityMask* Windows API call or *SetThreadAffinityMask* to operate on individual threads. The *SetProcessAffinityMask* Windows API call establishes a set of processors that a thread is eligible to run on. When hard processor affinity is set, a Ready thread will be scheduled to run only on an available processor from its processor affinity set. If all the processors in the affinity set are busy servicing higher priority work, the Ready thread remains in the Ready Queue, *even if other idle processors are available.*

Note Whereas a 64-bit Windows®–based system supports a maximum of 64 processors, a 32-bit Windows–based system supports a maximum of 32 processors. Therefore, functions such as *GetProcessAffinityMask* simulate a computer with 32 processors when called under WOW64. The affinity mask is obtained by performing a bitwise OR operation of the top 32 bits of the mask with the low 32 bits. Therefore, if a thread has affinity for processors 0, 1, and 32, WOW64 reports the affinity as 0 and 1, because processor 32 maps to processor 0. Functions that set processor affinity, such as *SetThreadAffinityMask*, restrict processors to the first 32 processors under WOW64.

The reason for restricting a process's threads to some specific set of processors is the performance improvement that can result when a thread is dispatched on a processor in which its code and data are intact in the processor caches from the last time the thread executed. Once the thread has established its working set in the processor caches, the instruction execution throughput can be expected to be several times faster. The instruction execution throughput rate for a thread subject to a cache cold start is potentially much slower than the throughput rate that can be expected when the same thread benefits from a warm start in cache. (The specific benefit of a cache warm start to an application thread is, of course, highly workload-dependent.)

Hard processor affinity is a particularly attractive option on 4-way or larger multiprocessors that are running mixed workloads. It is used to counteract some of the multiprocessor effects that normally limit multiprocessor scalability. Restricting the threads of a process to a subset of the available processors increases the likelihood that the threads are dispatched on a processor that retains cached data from that process address space. Concentrating threads on a subset of the available processors tends to improve the instruction execution rate through better cache utilization. It can also reduce serialization delays that are caused by interference from threads running concurrently on other processors.

Note The performance improvements you can achieve in theory by using processor partitioning are difficult to demonstrate in practice when you can't measure processor instruction execution throughput directly. For example, assume a process that consumed a total of 80 percent busy across all eight available processors (or an average of 10 percent busy per processor) needed only 40 percent of one processor when its threads are all concentrated on that processor. Partitioning boosts the instruction execution throughput, and the application runs twice as fast.

But, unless you are running a workload that is repeatable, this improvement is normally difficult to measure precisely. Using a repeatable benchmark workload, you should be able to observe that % Processor Time is reduced at a comparable throughput level. Alternatively, the average % Processor Time consumed per transaction should be reduced. However, repeatable workloads, like the ones associated with standard benchmarks, are usually not the kind of heterogeneous workloads that asymmetric partitioning helps the most with.

Some server applications permit you to establish hard processor affinity settings that govern their dispatching. Examples include Microsoft SQL Server, IIS, and ASP.NET. In the absence of these application-level settings, the primary tool for implementing processor partitioning at the application level is the Windows System Resource Manager (WSRM). The primary tool for implementing processor partitioning at the device driver level is the Interrupt Affinity tool from the *Windows Server 2003 Resource Ki*t.

Important The policy for setting hard processor affinity is considerably more restrictive than the policy for setting soft processor affinity scheduling, which the Windows Server 2003 operating system implements automatically. Hard processor affinity will not do much good unless you are able to achieve a relatively high concentration of processing from the selected application threads on the processors they are restricted to running on. Hard processor affinity can do considerable harm if you restrict threads from a critical workload to too few processors.

Guidelines for setting hard processor affinity Improving the instruction execution throughput of specific process threads on a multiprocessor is the rationale behind processor partitioning. Partitioning is used to counteract some of the multiprocessor scalability effects described earlier. It is an important technique for improving the cost-effectiveness of large n-way shared-memory multiprocessors, especially those running heterogeneous workloads.

Implementing a hard processor affinity policy to partition workloads on the machine has the goal of concentrating the designated CPU workload on a subset of the available processors. For this strategy to be successful, you need to understand the processor requirements of your application during both normal and peak loads. It is important that you do not restrict the application to too few processors, because that will lead to excessive queuing for the processors that the application is eligible to run on. Most importantly, partitioning requires an ongoing commitment to monitor the system to detect and respond to any changes in the workload consumption pattern.

You can gain an understanding of the processor requirements of any specific application in a straightforward manner by monitoring its associated per Process\% Processor Time counters. You should acquire some knowledge of longer-term processor usage patterns and trends by monitoring the application under consideration over several weeks at regular intervals. If processor utilization peaks at predictable hours, monitor those periods more closely to understand the characteristics of those peak periods. Consider monitoring peak periods in even greater detail, for instance, at one-minute intervals instead of five. You need to be able to configure the system so that enough processor resources are available to it for both normal and peak loads.

Deriving a *peak:average* ratio, as discussed in Chapter 1, "Performance Monitoring Overview," is a useful metric for understanding the application's processing requirements, where the peak period is the busiest one- or five-minute interval over a typical processing shift, compared to the average measured during the shift. The greater the variability in processor requirements, of course, the higher the peak:average ratio. Workloads with a peak:average ratio in excess of 2 or 3 are very inconsistent and need to be watched extremely carefully.

You also need to factor in some additional system overhead requirements beyond what is specifically consumed at the process level. No doubt, the application in question is going to need some additional system resources performed on its behalf. With server applications like SQL Server that attempt to bypass most Windows Server 2003 operating system services once they are initialized, the amount of additional system overhead they require is only another additional 10–20 percent, usually. Other server applications that rely more on various system services might require allotting an additional 20–40 percent more processor capacity.

Once you understand the processor requirements for a given workload, you can establish a partitioning scheme. To ensure the system remains very responsive to the designated workload, configure enough processors so that the projected application average load, including system overhead, would put the configured processors in the range of 20–60 percent processor busy. The rationale behind this configuration guideline is as follows. Below 20 percent utilization, the processors dedicated to this workload are underutilized and might be more effectively deployed elsewhere. Above 60 percent utilization, there might not be sufficient processor resources to handle peak loads (assuming a peak:average ratio of 1.5:1 to 2:1, for example) adequately without excessive queuing. Of course, also monitor the System\Processor Queue Length counter, and periodically check to ensure that threads from the application you are concerned with are not delayed waiting in the Ready Queue (denoted by Process(*n*)\Thread State = 1).

Finally, you need a commitment to continue monitoring that application and others running on the machine to ensure that the workload is stable and the partitioning scheme you formulated remains optimal. Whenever workload changes become apparent, you need to revisit the partitioning scheme according to the guidelines discussed earlier.

> **Tip** Implementing partitioning by restricting the processing resources that low-priority, discretionary workloads have access to often requires less analysis than setting controls that constrict critical workloads. Putting restrictions on discretionary workloads to ensure they do not draw excessive resources away from critical workloads is almost always an easier thing to do.

Application-level settings Some server applications allow you to specify hard processor affinity settings that govern their dispatching. Examples include SQL Server, ASP, and ASP.NET applications running inside IIS; and other .NET Framework applications. The hard processor affinity settings you specify at the application level are used in a *SetProcessAffinityMask* API call to specify a processor affinity mask for the threads of those applications. Each bit position in the processor affinity mask corresponds to a physical processor, with bit 0 representing processor 0, bit 1 representing processor 1, and so on. If the threads of the process can be scheduled to run on the specific processor, the corresponding bit in the processor affinity mask is set to 1. If threads are to be restricted from running on that processor, the corresponding bit in the processor affinity mask is set to 0. For example, the hexadecimal value of a processor affinity mask set to 0x0d (or a decimal value of 13) represents the bit pattern 1101. On a computer with four processors, this indicates that process threads can be scheduled on processors 0, 2, and 3, but not on processor 1. Table 6-7 lists the applications that provide a hard processor affinity tuning parameter.

Table 6-7 Hard Processor Affinity Settings

Application	Parameter	Where to Set It
.NET Framework	*cpuMask*	<processModel> element coded in the Machine.config file
	■ *Process.ProcessorAffinity* property ■ *ProcessThread.ProcessorAffinity* property ■ *ProcessThread.IdealProcessor* property	
SQL Server	■ *affinity mask* ■ *affinity64 mask*	Enterprise Manager GUI or *sp_configure* stored procedure
ASP and ASPX	■ *SMPAffinitized* ■ *SMPProcessorAffinity-Mask*	IIS Metabase: ■ /LM/W3SVC/AppPools ■ /LM/W3SVC/AppPools/DefaultAppPool ■ /LM/W3SVC/AppPools/DefaultApp-Pool/*application_pool_name*

The processor affinity mask settings available to applications that are built to run with the .NET Framework are the most extensive and most flexible. Applications running the .NET Framework can set a process's processor affinity mask using the *Process.ProcessorAffinity* property. Or a processor affinity mask can be set for an individual thread using the *ProcessThread.ProcessorAffinity* property. At the thread level, a .NET Framework application can even set the *ProcessThread.IdealProcessor* property to instruct the operating system to try to confine the thread to being scheduled on a specific subset of the available processors. Setting the *ProcessThread.IdealProcessor* property is similar to using a thread's built-in soft processor affinity because it allows the thread to run on an available idle processor if all its ideal processors are busy. Applications using the .NET Framework that need to scale on large-scale, parallel NUMA architecture machines have additional considerations, as discussed in the section entitled "ccNUMA Architectures."

SQL Server supports a processor affinity mask in both 32-bit and 64-bit addressing modes. In 32-bit mode, the processor affinity mask can reference only the first 32 processors on a 64-way multiprocessor. To set a processor affinity mask for a 64-processor configuration, you must use the 64-bit flavor of the setting. The processor affinity mask in SQL Server is designed to be used in conjunction with its priority boost option on machines dedicated to running the SQL Server application. The priority boost option sets the base priority of all SQL Server threads to run in the real-time range. With the priority boost option set, SQL Server threads are dispatched at a higher priority than almost any other Ready thread on the machine. This might suit the needs of SQL Server, but it can be very damaging (in terms of throughput) to the threads of any other application process that might need to run on that machine. By setting a processor affinity mask that confines SQL Server threads to a subset of the available processors, you can make sure that at least one processor is available to run threads from other applications.

IIS version 6 supports an *SMPProcessorAffinityMask* setting for Web garden application pools. For the maximum degree of flexibility, you can code processor affinity mask settings for each separate application pool or use a global setting that governs all application pools. The *SMPProcessorAffinityMask* settings are activated only when the *SMPAffinitized* property is set to true.

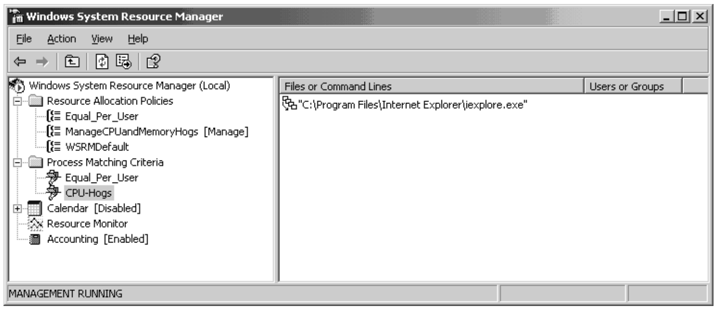
These application-level settings can be used in a variety of ways to configure large-scale, n-way multiprocessors effectively:

- You can run multiple instances of SQL Server on the same machine and, for each instance of the SQL Server process, set processor affinity masks that effectively isolate the threads from each SQL Server instance to mutually exclusive sets of eligible processors.
- You can isolate the executing Worker threads from separate application pools in IIS Web gardens to mutually exclusive sets of eligible processors.
- You can run the SQL Server database application on the same machine as your front-end IIS ASP and ASP.NET applications, and use processor affinity masks to minimize processor contention among these disparate workloads.
- You can isolate potentially long-running application processes in a Web server environment to a limited subset of the available processors to ensure that these applications don't hog processor resources to the detriment of other more response-oriented Web applications.

Partitioning using WSRM The Windows System Resource Manager (WSRM) is a management component that supports policy-based automated performance management. It is available with Windows Server 2003 Enterprise Edition and Datacenter Edition. WSRM can be used to establish and maintain an effective processor partitioning scheme for server applications that do not provide a processor affinity mask tuning option. For more information, see the *Microsoft Windows Server 2003 Deployment Kit*.

In WSRM, you build two kinds of constructs to implement policy-based automated performance management. First, you construct a managed workload definition and assign one or more processes to the workload definition you want to manage. Then you construct a resource allocation policy that will be used to control resource consumption by the designated workload. For controlling processor consumption, WSRM supports two modes of control. The first provides limits on the amount of processor capacity (or bandwidth) that managed workloads can consume, but only applies these limits when the system is under stress and the managed workload is consuming more than its designated share of the processor. The second processor control sets hard processor affinity to restrict designated workloads to a specific subset of the available processors.

An example will illustrate how these two WSRM constructs work together to implement and enforce automated resource management policies. Figure 6-13 shows the WSRM administration console for a 2-way multiprocessor machine for which WSRM policies govern the consumption of processor resources by designated workloads. In this example, a simple policy was designed to prevent specific, known processes from monopolizing the processors to the detriment of other more critical server applications.

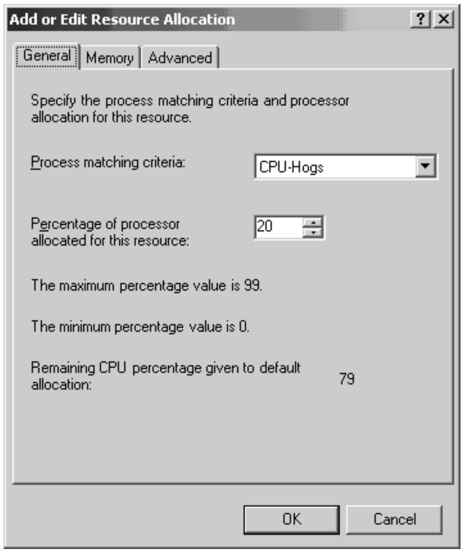


"C:\Program Files\Internet Explorer\iexplore.exe"

Figure 6-13 The WSRM administration console for a 2-way multiprocessor machine

In Figure 6-13, in the Process Matching Criteria folder, a workload designated CPU-Hogs is defined. This workload identifies the Internet Explorer application, iexplore.exe, as a process to be managed. The WSRM Administrator allows you to identify any service or application process and associate it with a workload definition.

Figure 6-13 also shows a WSRM resource allocation policy called *ManageCPUand-MemoryHogs*, which is intended to be used to manage the processes associated with the CPU-Hogs workload. Figure 6-14 shows the Add Or Edit Resource Allocation policy properties pages that are used to associate a managed workload with its resource allocation policy.



Specify the process matching criteria and processor allocation for this resource.

Process matching criteria: CPU-Hogs

Percentage of processor allocated for this resource: 20

The maximum percentage value is 99.

The minimum percentage value is 0.

Remaining CPU percentage given to default allocation: 79

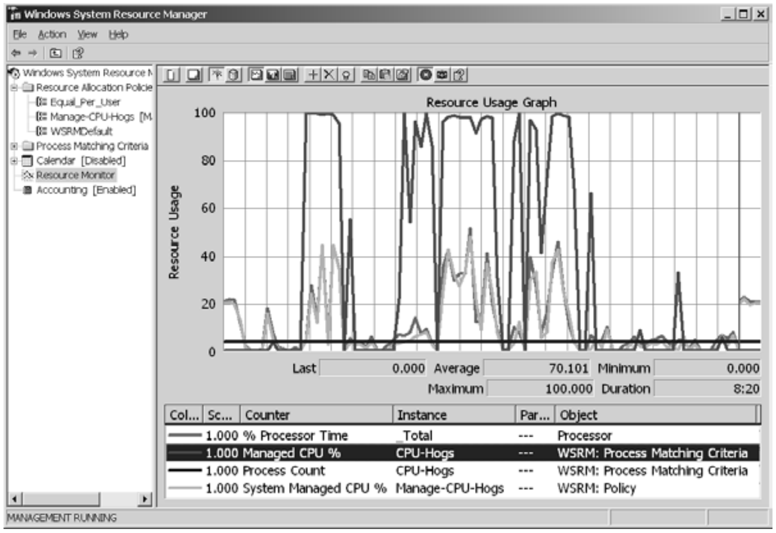
Figure 6-14 Using a properties page to associate a managed workload with its resource allocation policy

The General tab in Figure 6-14 also shows the WSRM control that assigns a target value for the amount of processor bandwidth that the designated workload will be allowed to consume. Notice that WSRM calculates a value called Remaining CPU Percentage Given To Default Allocation based on this target value. If the processor is saturated and the system is unable to provide this CPU percentage target to the remaining workloads, WSRM initiates corrective actions. It adjusts downward the dispatching priority of the threads of the processes it is assigned to manage, as it tries to force the actual processor consumption of the workload closer to the target value. At the same time, WSRM boosts the base priority of processes in the remaining default workload.

With this approach to control, the processor usage target you set for a managed workload is not a hard upper limit on how much processor bandwidth a managed workload can consume. As long as processor resources are available, WSRM will not limit the resource consumption of managed processes. WSRM intervention occurs only when there is a shortage of processor capacity *and* resource consumption by the managed workloads exceeds its target.

Tip Using the wsrmc command-line interface, you can automate the entire process of setting up, implementing, and modifying WSRM resource allocation policies. You can build WSRM procedures that are easy to migrate from one machine to the next in an application server cluster.

WSRM also supplies performance counters to help you monitor how the policies you have established are working and also evaluate the effectiveness of those policies. These performance counters are illustrated in Figure 6-15. Processor and memory consumption by any managed processes is conveniently grouped by process, by the workloads defined by process criteria rules, and by policy. By tracking overall processor utilization, as illustrated, you can easily determine whether the policies you have defined and implemented are having the desired effect.

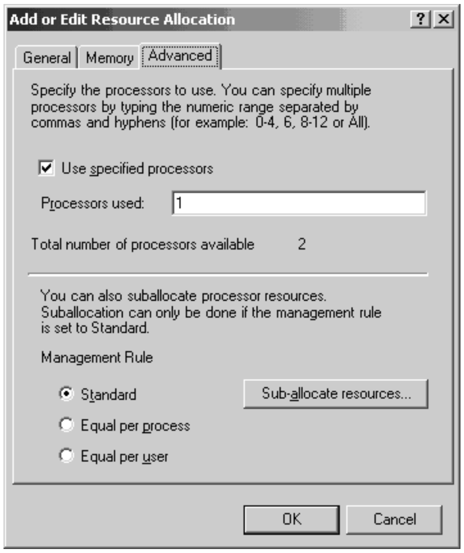


Last 0.000 Average 70.101 Minimum 0.000
Maximum 100.000 Duration 8:20

Col...	Sc...	Counter	Instance	Par...	Object
—	1.000	% Processor Time	_Total	---	Processor
—	1.000	Managed CPU %	CPU-Hogs	---	WSRM: Process Matching Criteria
—	1.000	Process Count	CPU-Hogs	---	WSRM: Process Matching Criteria
—	1.000	System Managed CPU %	Manage-CPU-Hogs	---	WSRM: Policy

Figure 6-15 WSRM performance counters

Processor partitioning using hard processor affinity with WSRM provides both a more restrictive and a more deterministic way to regulate how much processor time a managed workload consumes as compared with the manipulation of its dispatching priority. Using WSRM controls, you can assign a processor affinity mask to any service or application process that does not provide its own facilities for assigning hard processor affinity. This facility is illustrated in Figure 6-16.



Specify the processors to use. You can specify multiple processors by typing the numeric range separated by commas and hyphens (for example: 0-4, 6, 8-12 or All).

☑ Use specified processors

Processors used: 1

Total number of processors available 2

You can also suballocate processor resources. Suballocation can only be done if the management rule is set to Standard.

Management Rule

[✓ Standard]
Equal per process
Equal per user

Figure 6-16 Using WSRM controls to assign a processor affinity mask

Instead of specifying the processor affinity mask as a bitmap, as required elsewhere, the WSRM provides a convenient control that lets you designate the eligible processors by processor ID. For example, a specification of *0, 4–7* means that threads of the managed processes are eligible to run on processor 0, as well as processors 4 through 7.

Caution WSRM is not intended to manage any server applications that dynamically modify their own process scheduling priority, memory limits, or processor affinity. If you configure any of these application-tuning parameters, add them to the WSRM process exclusion list. For more information, see the section "Best Practices" in the WSRM Help.

Interrupt Affinity Filter By using the Interrupt Affinity Filter in the *Windows Server 2003 Resource Kit* tools, you can configure your machines to confine interrupt processing for selected devices to a subset of the available processors. The Interrupt-Affinity Filter tool allows you to assign a processor affinity mask to any of the Interrupt Service Routines (ISRs) that service the devices connected to your machine. Note that the DPC routine scheduled by the ISR to complete the work of interrupt processing, as discussed in Chapter 1, "Performance Monitoring Overview," is by default dispatched on the same processor where the ISR ran. The Interrupt Affinity Filter tool ensures that processing by both the ISR and (by default) its associated DPC routine is confined to specific processors on a multiprocessor.

Like any other segments of code, ISRs will execute faster when they are able to benefit from warm starts in processor cache. Confining selected ISRs so that they are processed only on a subset of the available processors increases the chances that when they do execute, their code and data areas are still resident in processor cache from the last time they executed. Setting processor affinity for ISRs is an effective optimization for almost any device that sustains a high volume of interrupts. High-speed interfaces like gigabit Ethernet cards and Fibre Channel adaptors, which sustain a high volume of interrupts, will usually benefit most. For devices that are incapable of generating hundreds or thousands of interrupts per second—excluding those implementing aggressive interrupt combining, coalescing, and polling—establishing processor affinity is usually not worth the bother.

Caution One risk of a very unbalanced, asymmetric configuration is less responsive servicing of device interrupts. Because ISRs execute at an elevated priority, you need to be very careful to avoid overloading any processor in a critical application's hard affinity set by giving it too many interrupts to process.

Establishing hard processor affinity for interrupts has comparable considerations to the ones just discussed with regard to User-mode processes. If the device interrupt processing is not concentrated sufficiently to significantly improve the chances of a cache warm start, such processing is not worth the effort. Still, even though the improvement in interrupt processing speeds might be slight, setting an affinity mask for the ISR associated with a device that could be a potential bottleneck will usually yield some improvement in performance and could be a good preventative measure for the future.

For example, assume the processor you intend to assign to device interrupt processing is running near saturation at, say, 80 percent or more busy. If, for a very active device interface, the Processor(*n*)\% Interrupt Time plus Processor(*n*)\% DPC Time—where you have attempted to concentrate interrupt processing—totals less than 3–5 percent busy for that processor, any improvement you can expect to see is likely to be marginal unless the processor cache is very large (for example, is multiple megabytes). That is because the other threads dispatched on the processor will tend to absorb most of the processor cache.

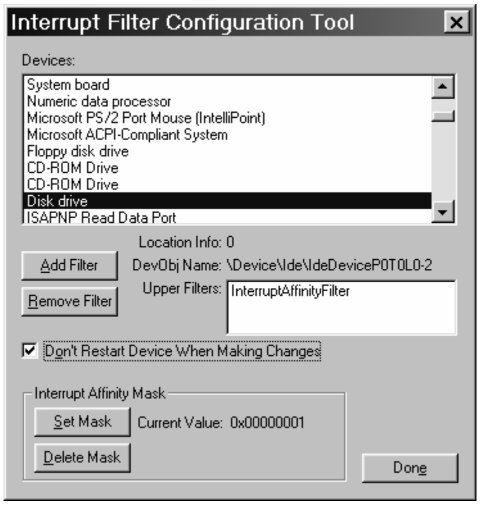
But beware of concentrating too many interrupts from different devices on a single processor. Device interrupts from an interface must be handled one at a time, making them very sensitive to minor delays in processing. Pending interrupts on an interface must wait until the ISR has cleared the previous interrupt and reset the device. Confining interrupt processing for heavily used interfaces to too few processors can create delays in device interrupt processing for the devices you are trying to help. Suppose you have a processor in which you have attempted to concentrate interrupt processing for a very active device interface. If you then observe that Processor(*n*)\% Interrupt Time combined with Processor(*n*)\% DPC Time is over 30–40 percent, you are likely to see a reduced benefit (assuming the processor is also running application code). Any improvement in ISR code execution time will likely be offset by increased interrupt pending delays.

Warning Be careful to avoid a very unbalanced, asymmetric configuration. If the sum of Processor(*n*)\% Interrupt Time and Processor(*n*)\% DPC Time is over 40 percent for a processor in an asymmetric configuration, you are likely to see a reduction in overall interrupt processing responsiveness instead of an improvement, because interrupt processing is confined to too few processors.

Confining interrupts to a subset of the available processors on a large-scale multiprocessor has one more potential benefit. If you direct interrupt processing away from processors where you have concentrated specific server applications—either to server applications that have processor affinity mask settings or by using WSRM—User-mode processes that are isolated from interrupt processing will benefit, because they are subject to fewer interruptions from ISRs dispatched at an elevated priority. By setting Interrupt Affinity Filters to limit the number of interrupts on specific processors that you have partitioned to accept critical application threads, you improve the CPU service time of those threads.

Tip Another problem caused by setting interrupt affinity addresses is that higher priority ISRs/DPCs running on the same processors as application threads can interfere with application throughput. In a database environment, for example, it is sometimes worthwhile to concentrate all device interrupts on only one or two processors and exclude those processors entirely from running application code. In this case, even if the processors handling interrupts run at greater than 80 percent utilization, there is still a clear performance benefit resulting from the clean partitioning of work.

Figure 6-17 illustrates the use of the Interrupt Affinity Filter tool. It shows a processor affinity mask that was coded for a disk drive interface. Because the processor affinity mask that is coded is 0x'1', the ISR responsible for processing disk interrupts will be dispatched only on processor 0. If that processor happens to be busy servicing a higher priority interrupt, the disk interrupt will remain pending until the previous interrupt is cleared, even if other processors are available to handle the interrupt. But more importantly, application code running on processor 0 will be interrupted for some unknown length of time until no disk interrupts are pending.


Interrupt Filter Configuration Tool

Devices:

System board
Numeric data processor
Microsoft PS/2 Port Mouse (IntelliPoint)
Microsoft ACPI-Compliant System
Floppy disk drive
CD-ROM Drive
CD-ROM Drive
Disk drive
ISAPNP Read Data Port

Location Info: 0
DevObj Name: \Device\Ide\IdeDeviceP0T0L0-2
Upper Filters: InterruptAffinityFilter

☑ Don't Restart Device When Making Changes

Interrupt Affinity Mask
Current Value: 0x00000001

Figure 6-17 Using the Interrupt Affinity Filter tool

Hyper-Threading *Hyper-Threading* is the brand name Intel uses for technology that provides the ability to load and process instructions from two separate instruction streams on a single processor core. With Hyper-Threading, a single processor engine appears to the operating system as two processors. Because the operating system treats a Hyper-Threaded processor as a multiprocessor, it will dispatch two Ready threads at a time. Hyper-Threaded processors can execute instructions from both threads simultaneously. Assuming there are two Ready threads that can be scheduled to execute in parallel, Hyper-Threading can theoretically yield performance at nearly double the instruction execution rate of a conventional single-threaded processor. Typically, however, if you see an average improvement of 30 percent or better instruction throughput, you probably should consider yourself fortunate.

Intel Hyper-Threaded processors implement *simultaneous multithreading* atop a superscalar processor core that runs the Intel IA-32 microarchitecture. As discussed earlier, superscalar machines attempt to process instructions from a single execution stream in parallel. For instruction streams that do not have a large enough degree of parallelism to take full advantage of the parallel processing resources available in a superscalar pipeline, machines resort to speculative techniques like branch prediction, out-of-order execution, and predication. Simultaneous multithreading takes a different approach by exploiting the parallelism inherent in multiple independent threads.

Note Hyper-Threading is Intel's proprietary implementation of simultaneous multithreading (SMT), a relatively recent addition to commercial processor architectures. Much of the research impetus for SMT was initiated at the University of Washington. For an overview of this recent research, see the Computer Sciences department Web site at the University of Washington at http://www.cs.washington.edu/research/smt/.

Dual logical processors The reason Hyper-Threading is often effective, according to the Intel engineers, is that the parallel processing core of the instruction execution pipeline is often idle because of pipeline stalls. The processor core is often underutilized when a single instruction stream is being executed. With simultaneous multithreading technology, when the instruction stream from one thread stalls the instruction execution pipeline, instead of idling, a Hyper-Threaded machine continues to execute instructions from the other instruction stream, boosting overall performance. The ability to execute dual instruction streams simultaneously is accomplished by supplying a second set of external interfaces to make each processor core look like a 2-way multiprocessor.

To schedule and control a second parallel instruction stream, the processor must provide an extra set of general purpose registers and control registers. Instructions from both processing threads proceed through the same instruction execution pipeline, which is largely identical to the single-threaded version of the IA-32 microarchitecture. Proponents consider this to be one of the great advantages of the simultaneous multithreading approach. It provides a significant potential for speed-up without requiring a major overhaul of the existing superscalar instruction execution pipeline. But Hyper-Threading performs better only under specific circumstances. Hyper-Threading is still an unproven technology on large-scale multiprocessors, where it appears as likely to do harm as it is to foster improvement.

A Hyper-Threaded processor core must keep track of two simultaneously executing instruction streams. This requires duplicating the circuitry that implements the external resources the operating system sees. These include the processor register set, processor status registers, and other architectural features used to establish the thread context. Meanwhile, internal processor resources associated with the instruction execution pipeline that are not visible to the operating system can be shared. Some internal resources are actually partitioned into two halves, but this is primarily a result of fairness considerations in scheduling simultaneous threads. The instruction execution pipeline proceeds to make forward progress on the two instruction streams in a fair, round-robin manner. But if one of the instruction streams stalls, instructions from the other instruction stream are used to fill the otherwise idle cycles.

Performance considerations Hyper-Threading does introduce into thread dispatching some slightly different performance considerations, compared to conventional multiprocessors. Sharing the same processor core resources among two instruction streams executing in parallel can lead to resource contention between these threads. As in conventional shared memory multiprocessors, threads executing in parallel contend for the shared memory bus. But, in a Hyper-Threaded processor, there is also bound to be internal contention for the processor caches, as well as other internal resources of the instruction pipeline, like the Register Allocation Table and the functional units that execute different types of instructions. In addition, the logical processors might face problems because of false sharing of the cache, as discussed in the technical note entitled "Avoiding False Sharing on Intel Hyper-Threading Technology Enabled Processors" posted on the Intel Web site: http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/19980.htm. If there is too much internal contention among simultaneous threads trying to access shared resources, a Hyper-Threaded processor will run those threads slower than if those threads were able to execute in a serial mode on a conventional superscalar. Synthetic instruction streams constructed to measure the performance of a processor are often prone to this condition. If they replicate a single instruction stream that is then scheduled to run simultaneously on the two logical processors, internal resource conflicts of the sort that will dampen instruction execution throughput can result.

A second performance consideration is the fairness of this scheduling scheme when one instruction stream stalls. Within the instruction execution pipeline, the round-robin scheduling scheme executes microinstructions, bouncing back and forth from one thread to another. If one instruction stream is stalled for a relatively long time, the other instruction stream can wind up monopolizing the resources of the shared instruction execution pipeline. On a Hyper-Threaded machine, the Idle loop accelerates the call to the *processr.sys* power management routine to transition an idle logical processor to a low power mode more quickly than it would on a conventional multiprocessor. The effect is to shut down the idle logical processor quickly to free up processor core resources, which can then be dedicated to the other processor that is running an active thread's instruction stream.

On a multiprocessor constructed from two or more Hyper-Threaded processor cores, there is one additional thread scheduling consideration. The Windows Server 2003 thread Scheduler is aware of which logical processors are associated with a physical processor core. The Scheduler favors dispatching a Ready thread on one of the logical processors on an idle processor core over dispatching one on an idle logical processor on a processor core that already has an active thread. The Scheduler attempts to dispatch one Ready thread per processor core before it starts doubling up and giving a Hyper-Threaded core two instruction streams to process in parallel. Soft processor affinity is also modified. Scheduling a thread on the processor core where it was last dispatched is preferred so that the two logical processors become the ideal processors.

Note To determine whether the machine is a Hyper-Threaded multiprocessor or a conventional multiprocessor, User-mode applications can make a *GetLogicalProcessorInformation* API call. This API call returns an array of SYSTEM_LOGICAL_PROCESSOR_INFORMATION structures that show the relationship of logical processors to physical processor cores.

ccNUMA architectures The Windows Server 2003 operating system provides support for NUMA architecture machines. NUMA stands for *Non-Uniform Memory Access*.

It is a popular approach to building very large multiprocessors that overcomes the traditional scalability problems associated with shared memory multiprocessors, as illustrated in Figure 6-7 previously. These scalability limitations have a variety of sources. For example, heat dissipation can be a significant factor in how many processors are packaged onto a single board. Electrical considerations are another significant factor. Bus capacity for access to shared memory might be another limitation. NUMA machines attack these scalability problems directly by providing multiple processor boards that are interconnected using additional memory buses. Almost all the larger machines that you can buy that utilize more than 8 processor chips employ the NUMA approach.

NUMA machines are constructed from nodes that usually contain either 4 or 8 processor chips, along with their dedicated caches, plus some amount of local memory and local memory bus or buses. Limiting the size of a node to just 4–8 processors makes it unlikely that the local memory bus or buses shared by the processors in a node will saturate under load. *Nodes* are modular building blocks that can be strung together using a system interconnection of some kind that provides for remote memory accesses. Because multiple nodes are interconnected, threads executing on one node can access remote memory on another node. The modular nature of NUMA machines also lends itself to a policy of upgrading a system incrementally as more and more processing resources are needed.

As a consequence of having multiple buses, memory access in NUMA machines becomes more complicated. A key feature of this architecture is that remote memory accesses take longer because they must traverse the system interconnect. The disparity in the time it takes to access remote vs. local memory is what leads to the name—memory latency is not *uniform*, or constant, depending on what specific memory location an instruction accesses. Subsets of processors each have a local memory bus used to access a fast, shared local memory unit. Remote memory access is slower—perhaps several times slower depending on the specific manufacturer. If executing threads can maintain a high rate of local memory accesses, NUMA architectures can scale cost-effectively beyond eight processors. This makes them capable of supplying considerable amounts of parallel processing resources for large computing problems. Sometimes applications have to be designed with NUMA in mind, however, to achieve the desired level of scalability. On the NUMA machines supported by Windows Server 2003, threads perform remote memory accesses that are mediated by some form of a memory controller that guarantees cache coherence between the nodes. These machines are known as *cache coherent* NUMA, or ccNUMA architectures for short. Figure 6-18 illustrates this particular approach for a 16-way multiprocessor. Because such machines maintain coherent local processor caches, the content of global memory presents a single, consistent image to executing programs. This means that *thread-safe* programs running correctly on a conventional multiprocessor will also execute correctly on a ccNUMA machine without modification.

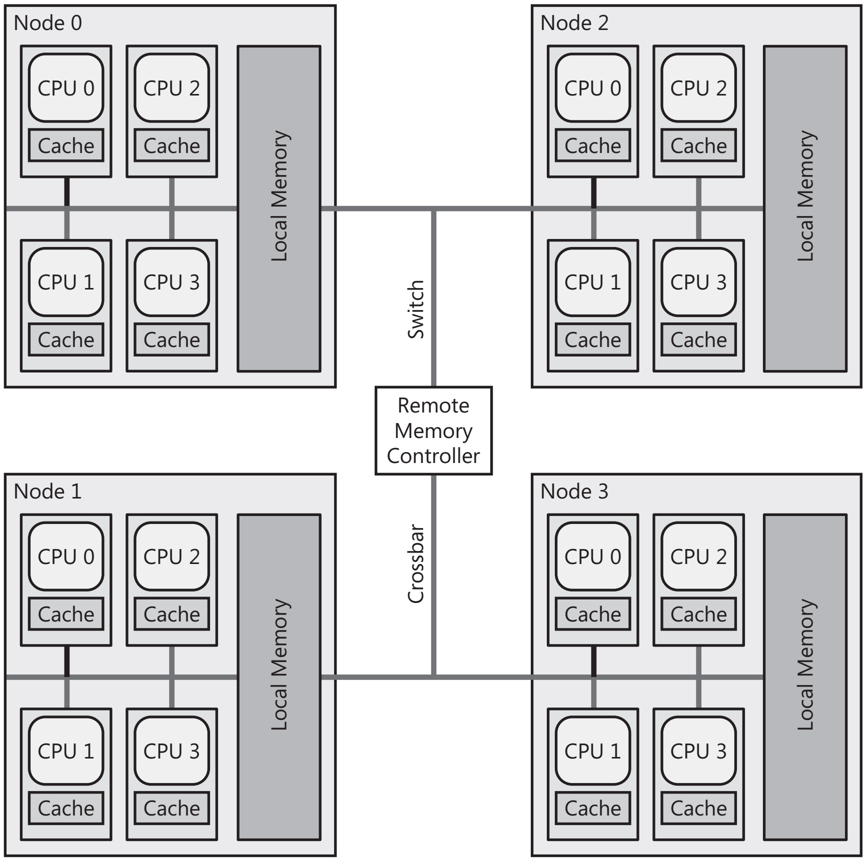


Figure 6-18 The ccNUMA architecture for a 16-way multiprocessor

The Windows Server 2003 operating system also plays a crucial role in enhancing the performance of ccNUMA machines by ensuring that the threads of a process are scheduled to run on the same nodes in which their memory is allocated. Just as threads in a symmetric multiprocessor have an affinity for an ideal processor, threads on a NUMA machine have an affinity for an *ideal node*. If a processor is idle at the thread's ideal node, the Scheduler will execute the thread there. In addition, when a running thread encounters a page fault, the virtual memory manager tries to allocate an available frame from local memory to resolve the page fault whenever possible. This is accomplished by having the operating system maintain separate memory management data structures for each node. This includes separate Paged and Nonpaged pools per node for system memory allocations.

To take full advantage of all the parallel processing resources available on a ccNUMA machine, a workload must first exhibit a high degree of parallelism. However, that is not the only important performance consideration. The fact that memory access latency is not uniform means executing threads run faster when they restrict the number of times they require access to remote memory. If all the threads of a process can execute concurrently within the same node by using just local memory, the highest levels of performance can be achieved.

Ideally, a multithreaded process running on a NUMA machine runs no more threads concurrently than there are processors available on a node, unless you want or need to harness the additional processing power at the other nodes. If additional processing capability is needed, depending on the application, you might be able to launch multiple processes that can have hard processor affinity set to run on different processor nodes. Of course, if these are .NET Framework applications, COM+ components, or ASP and ASP.NET programs operating in Web gardens, you have the flexibility to control the degree of both multithreading and multiple processes. You can set the thread pooling attributes of these applications to limit them to having no more Ready threads than there are processors in a node.

But consider a multithreaded process with more concurrent threads running than there are processors in a node. In this case, some ready threads will be scheduled to run on one or more other nodes. If these threads need to perform a good deal of remote memory access, they will run slower than the threads that have a higher concentration of local memory accesses.

These thread scheduling considerations suggest that some programs need to be sensitive to the longer latency associated with remote memory access if performance on NUMA machines is to scale effectively. Several Windows API functions are provided that programs can call to determine whether they are executing on a NUMA machine and what the topology of the machine is. For example, the *GetNumaProcessorNode* function returns the NUMA node number for a specified processor. These APIs make it possible for an application to establish what amounts to *hard node affinity*. They can determine the NUMA topology and then set a processor affinity mask to ensure that specific threads can execute only on the processors associated with a single node or subset of nodes. Using another function, *GetNumaAvailableMemoryNode*, a thread can also determine how much memory is available at the local node.

Memory Performance

This section considers several advanced memory performance topics. The first concerns a set of extended virtual addressing options that are available in 32-bit versions of Windows Server 2003. This set of extended options relieves 32-bit virtual memory address constraints that can arise on systems with 2 GB or more of RAM installed. By using the hardware's Physical Address Extension (PAE), for example, you can take advantage of 36-bit physical memory addressing available on most Intel server machines. For your applications to address additional memory outside the 2-GB limit on the size of User-mode virtual addresses, you might need to implement applications that support Address Windowing Extensions (AWE). Or you might consider using a boot option that allows you to extend the 2-GB private area available for User-mode process addresses. This means sacrificing addresses in the system range, which is not necessarily an effective trade-off. None of these interim solutions are needed once you are able to move to 64-bit systems and applications.

This section also discusses how to calculate a memory contention index that can be used to predict the onset of virtual memory shortages that lead Windows Server 2003 machines to page excessively. This memory contention index is easy to calculate and often has excellent predictive value.

Extended Virtual Addressing in 32-Bit Machines

Some server workloads can exhaust the 32-bit virtual address space associated with current versions of Windows Server 2003. Machines with 2 GB or more of RAM installed appear to be particularly vulnerable to these virtual memory constraints. When Windows Server 2003 workloads exhaust their 32-bit virtual address space, the consequences are usually catastrophic. This section discusses the signs and symptoms that indicate there is a serious virtual memory constraint.

This section also discusses the features and options that system administrators can employ to forestall running short of virtual memory. The Windows Server 2003 operating system offers many forms of relief for virtual memory constraints on 32-bit machines. These include:

- Options to change the way in which 32-bit process virtual address spaces are partitioned into private addresses and shared system addresses
- Settings that govern the size of key system memory pools
- Hardware options that permit 36-bit addressing

By selecting the right combination of options, system administrators can avoid many situations in which virtual memory constraints impact system availability and performance. Nevertheless, these virtual memory addressing constraints inevitably cause more concern as the size of RAM on these servers grows. The most effective way to deal with these constraints in the long run is to move to processors that can access 64-bit virtual addresses and run the 64-bit version Windows Server 2003.

Performance and capacity problems associated with virtual memory architectural constraints arise from a hardware limitation, namely, the number of bits associated with a virtual memory address. In the case of the 32-bit Intel-compatible processors that run Windows Server 2003, address registers are 32 bits wide, allowing for addressability of 0–4,294,967,295 bytes, which is conventionally denoted as a 4-GB range. This 4-GB range can be an architectural constraint, especially with workloads that need 2 GB or more of RAM to perform well.

Virtual memory constraints tend to appear during periods of transition from one processor architecture to another. Over the course of Intel's processor evolution, there was a period of transition from 16-bit addressing, which was a feature of the original 8086 and 8088 processors that launched the PC revolution, to the 24-bit segmented addressing mode of the 80286, to the current 32-bit flat addressing model implemented across all Intel IA-32 processors. As of this writing, the IA-32 architecture is in a state of transition. As 64-bit machines start to become more commonplace, they will definitively relieve the virtual memory constraints apparent on the 32-bit platform.

Virtual Memory Addressing Constraints

The 32-bit addresses that can be used on IA-32-compatible Intel servers are a serious architectural constraint. There are several ways in which this architectural constraint can manifest itself. One problem occurs when a User process exhausts the 2-GB range of private addresses that are available for its exclusive use. The processes most susceptible to running out of addressable private area are applications that rely on memory-resident caches to reduce the quantity of I/O operations they perform (that is, databases). Windows Server 2003 supports a boot option that allows you to specify how a 4-GB virtual address space is divided between User private area virtual addresses and shared system virtual addresses. The */3GB* boot option extends the User private area up to the 3-GB level and reduces the system range of virtual memory addresses down to as little as 1 GB.

If you specify a User private area address range that is larger than 2 GB, you must also shrink the range of system virtual addresses available by a corresponding amount. This can easily lead to a second type of virtual memory limitation, which occurs when the range of system virtual addresses available is exhausted. Because the system range of addresses is subdivided into several different pools, it is also possible to run out of virtual addresses in one of these pools long before the full range of system virtual addresses is exhausted.

In both circumstances just described, running out of virtual addresses often happens suddenly, frequently as a result of a program with a memory leak. Memory leaks are program bugs in which a process allocates virtual memory repeatedly, but then neglects to free it when finished using it. Program bugs where a process leaks memory in large quantities over a short period of time are usually pretty easy to spot. The more sinister problems arise when a program leaks memory slowly, or only under certain circumstances, so that the problem manifests itself only at erratic intervals or only after a long period of continuous execution time.

However, not all virtual memory constraints are the result of specific program flaws. *Virtual memory creep* because of slow, but inexorable, workload growth can also exhaust the virtual memory address range available. Virtual memory creep can be detected by continuous performance monitoring procedures. Such procedures allow you to intervene in advance to avoid otherwise catastrophic application and system failures. Fortunately, the same virtual memory performance monitoring tools and procedures you use to detect memory leaks, which were recommended in Chapter 3, "Measuring Server Performance," Chapter 4, "Performance Monitoring Procedures," and Chapter 5, "Performance Troubleshooting,"are also effective in diagnosing virtual memory creep.

Three counters at the process level describe how each process is allocating virtual memory. These are Process(*n*)\Virtual Bytes, Process(*n*)\Private Bytes, and Process(*n*)\Pool Paged Bytes. Process(*n*)\Virtual Bytes shows the full extent of each process's virtual address space, including shared memory segments used to mapped files and shareable image file DLLs. Any process with Private Bytes at or above 1.6 GB (without specifying */3GB*) is running up against its virtual memory constraints.

Note Because of virtual memory fragmentation, it is quite difficult for a process to increase the size of its 2-GB private area to its full size. Virtual memory allocation requests will typically fail before the private area reaches its full 2-GB limit because the remaining free blocks of the unallocated User private address space are frequently too small and fragmented to satisfy requests.

Without at least 2 GB of RAM installed, a User-mode process has difficulty exhausting its 2-GB virtual address range because virtual memory growth is constrained by the system's Commit Limit. Or, if large enough paging files are provided so that the Commit Limit is not a constraint, performance of the system is poor because of excessive paging to disk. In these circumstances, adding more RAM to the system is usually more of a priority than worrying about the potential for virtual memory to become exhausted. However, in 32-bit machines with large amounts of RAM installed, User processes running out of virtual memory are apt to be the much more serious problem. When workload growth causes critical server processes to exhaust their allotment of virtual memory, it might be possible to obtain relief using some form of extended virtual addressing. Configuration and tuning options that provide extended virtual addressing under 32-bit Windows Server 2003 are discussed in the next section.

Extended Virtual Addressing Options

The Windows Server 2003 operating system supports a number of extended virtual addressing options suitable for large machines configured with 4 GB or more of RAM. These include:

- */3GB* **boot switch** Allows the definition of process address spaces larger than 2 GB, up to a maximum of 3 GB.
- **Physical Address Extension (PAE)** Provides support for 36-bit physical addresses on Intel Xeon 32-bit processors. With PAE support enabled, Intel Xeon processors can be configured to address as much as 64 GB of RAM.
- **Address Windowing Extensions (AWE)** Permits 32-bit process address spaces access to physical addresses above their 4-GB virtual address limitations. AWE is used most frequently in conjunction with PAE.

Any combination of these extended addressing options can be deployed, depending on the circumstances. Under the right set of circumstances, one or more of these extended addressing functions can significantly enhance performance of 32-bit applications, which are still limited to using 32-bit virtual addresses. These extended addressing options relieve the pressure of the 32-bit limit on addressability in various ways discussed in more detail later in this section. But, these extended options are also subject to the addressing limitations posed by 32-bit virtual addresses. Ultimately, the 32-bit virtual addressing limit remains a barrier to performance and scalability.

Extended process private virtual addresses The */3GB* boot switch extends the per-process private address range from 2 GB to 3 GB. Windows Server 2003 permits a different partitioning of user and system addressable storage locations using the */3GB* boot switch. This extends the private User address range to 3 GB and shrinks the system area to 1 GB, as illustrated in Figure 6-19. The */3GB* switch supports an additional subparameter */userva=SizeInMB*, where *SizeinMB* can be any value between 2048 and 3072.

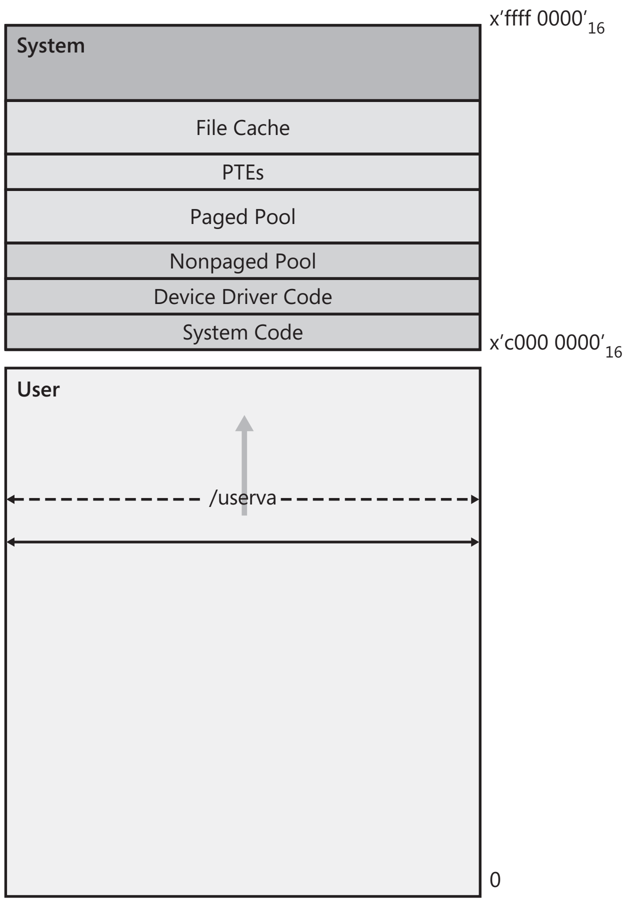


Figure 6-19 The /userva boot switch used to increase the size of the User virtual address range

Only applications compiled and linked with the *IMAGE_FILE_LARGE_ADDRESS_AWARE* switch enabled can allocate a private address space larger than 2 GB. Applications that are Large Address Aware include SQL Server, Exchange, Oracle, and SAS.

Caution Shrinking the size of the system virtual address range using the */3GB* switch can have serious drawbacks. Even though the */3GB* option allows an application to increase its private virtual addressing range, it forces the operating system to squeeze into a smaller range of addresses, as illustrated in Figure 6-19. In certain circumstances, this narrower range of system virtual addresses might not be adequate, and critical system functions can easily be constrained by their own virtual addressing limits. These concerns are discussed in more detail in the section entitled "System Virtual Memory Shortages."

PAE The Physical Address Extension (PAE) enables applications to address more than 4 GB of physical memory. It is supported by most current Intel processors running Windows Server 2003 Enterprise Edition or Datacenter Edition. Instead of 32-bit addressing, PAE extends physical addresses so that Windows-based systems can use 36-bit addresses, allowing machines to be configured with as much as 64 GB of RAM.

When PAE is enabled, the operating system builds an additional virtual memory translation layer that is used to map 32-bit virtual addresses into this 64-GB physical address range. The extra level of virtual address translation Page Tables provides access to physical memory in blocks of 4 GB, up to a maximum of 64 GB of RAM. In standard addressing mode, a 32-bit virtual address is split into three separate fields for indexing into the Page Tables used to translate virtual addresses into physical ones. In PAE mode, virtual addresses are split into four separate fields:

- A 2-bit field that directs access to four 1-GB sets of virtual address blocks
- Two 9-bit fields that refer to the Page Table Directory and the page table entry (PTE)
- A 12-bit field that corresponds to the offset within the physical page

Extended addressing is then provided by manipulating the four entries that are pointed to by the first 2-bit field so that they point to different 1-GB regions. This allows access to multiple sets of 4-GB virtual address ranges at the same time.

Server application processes running on machines with PAE enabled are still limited to using 32-bit virtual addresses. However, 32-bit server applications facing virtual memory addressing constraints can exploit PAE in two basic ways:

- They can expand *sideways* by deploying multiple application server processes. Both Microsoft SQL Server 2000 and IIS 6.0 support sideways scalability with the ability to define and run multiple application processes. Similarly, a Terminal Server machine with PAE enabled can potentially support the creation of more process address spaces than a machine limited to 4-GB physical addresses.
- They can indirectly access physical memory addresses beyond their 4-GB limit using the Address Windowing Extensions (AWE) API calls. Using AWE calls, server applications like SQL Server and Oracle can allocate database buffers in physical memory locations outside their 4-GB process virtual memory limit and manipulate them.

The PAE support provided by the operating system maps 32-bit process virtual addresses into the 36-bit physical addresses that the processor hardware uses. An application process, still limited to 32-bit virtual addresses, need not be aware that PAE is even active. When PAE is enabled, operating system functions can use only those addresses up to 16 GB. Only applications using AWE can access physical addresses above 16 GB to the 64-GB maximum. Large Memory Enabled (LME) device drivers can also directly address buffers above 4 GB using 64-bit pointers. Direct I/O for the full physical address space up to 64 GB is supported if both the devices and drivers support 64-bit addressing. For devices and drivers limited to handling 32-bit addresses, the operating system is responsible for copying buffers located in physical addresses greater than 4 GB to buffers in RAM below the 4-GB line that can be directly addressed using 32-bits.

Although expanding sideways by defining more process address spaces is a straightforward solution to virtual memory addressing constraints in selected circumstances, it is not a general-purpose solution to the problem. Not every processing task can be partitioned into subtasks that can be parceled out to multiple processes, for example. PAE brings extended physical addressing, but it adds no additional hardware-supported functions to extend interprocess communication (IPC). IPC functions in Windows-based systems rely on shared memory in the system range, which is still constrained by 32-bit addressing.

Important Most machines with PAE enabled run better *without* the */3GB* option because condensing the system virtual memory range all too frequently creates a critical shortage in system virtual addresses.

PAE is required to support machines with cache coherent Non-Uniform Memory Architecture (known as ccNUMA or sometimes NUMA, for short), but it is not enabled automatically. On AMD64-based systems running in long mode however, PAE is required, is enabled automatically, and cannot be disabled.

AWE The Address Windowing Extension (AWE) is an API that allows programs to address more physical memory locations than their 4-GB virtual addressing range would normally allow. AWE is used by applications in conjunction with PAE to extend the application's addressing range beyond 32 bits. Because process virtual addresses remain limited to 32 bits, AWE is a marginal solution with many pitfalls, limitations, and potential drawbacks.

The AWE API calls maneuver around the 32-bit address restriction by placing responsibility for virtual address translation into the hands of an application process. AWE works by defining an in-process buffering area, called the *AWE region*, that is used dynamically to map allocated physical pages to 32-bit virtual addresses. The AWE region is allocated as nonpaged physical memory within the process address space using the *AllocateUserPhysicalPages* AWE API call. *AllocateUserPhysicalPages* locks down the pages in the AWE region and returns a Page Frame array structure that is the normal mechanism the operating system uses to keep track of which physical memory pages are mapped to which process virtual address pages. (An application must have the Lock Pages in Memory user right to use this function.)

Initially, of course, no virtual addresses are mapped to the AWE region. Then, the AWE application reserves physical memory (which might or might not be in the range above 4 GB) using a call to *VirtualAlloc*, specifying the *MEM_PHYSICAL* and *MEM_RESERVE* flags. Because physical memory is being reserved, the operating system does not build page table entries (PTEs) to address these data areas. (A User-mode thread cannot directly access and use physical memory addresses. But authorized kernel threads can.) The process then requests that the physical memory acquired be mapped to the AWE region using the *MapUserPhysicalPages* function. Once physical pages are mapped to the AWE region, virtual addresses are available for User-mode threads to address. The idea in using AWE is that multiple sets of memory regions with physical memory addresses extending to 64 GB can be mapped dynamically, one at a time, into the AWE region. The application, of course, must keep track of which set of physical memory buffers is currently mapped to the AWE region and what set is required to handle the current request, and also perform virtual address unmapping and mapping as necessary to ensure addressability to the right physical memory locations. Figure 6-20 illustrates this process, which is analogous to managing overlay structures.

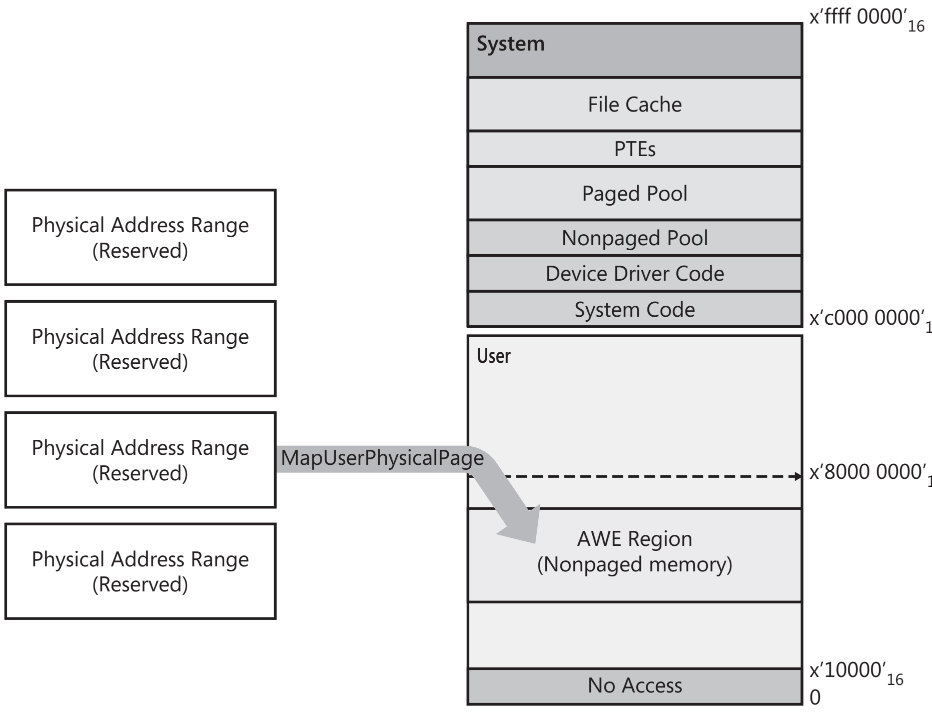

x'ffff 0000'16
System
File Cache
PTEs
Paged Pool
Nonpaged Pool
Device Driver Code
System Code
x'c000 0000'16
User
Physical Address Range (Reserved)
Physical Address Range (Reserved)
Physical Address Range (Reserved)
MapUserPhysicalPage
x'8000 0000'16
Physical Address Range (Reserved)
AWE Region (Nonpaged memory)
x'10000'16
No Access
0

Figure 6-20 Dynamically mapping virtual memory regions into the AWE region

In the example of an AWE implementation shown in Figure 6-20, the User process allocates four large blocks of physical memory that are literally outside the address space, and then uses the AWE call to *MapUserPhysicalPages* to map one physical address memory block at a time to the AWE region. In the example, the AWE region and the reserved physical memory blocks that are mapped to the AWE region are the same size, but this is not required. Applications can map multiple reserved physical memory blocks to the same AWE region, provided the AWE region address ranges they are mapped to are distinct and do not overlap.

In the example in Figure 6-20, the User process private address space extends to 3 GB. It is desirable for processes using AWE to acquire an extended private area so that they can create a large enough AWE mapping region to manage physical memory overlays effectively. Obviously, frequent unmapping and remapping of physical blocks slows down memory access considerably, mainly because of the need to broadcast Translation Lookaside Buffer (TLB) updates to other processors in a multiprocessor configuration. The AWE mapping and unmapping functions, which involve binding physical addresses to a process address space's PTEs, must be synchronized across multiple threads executing on multiprocessors. Even with all this additional overhead, AWE-enabled access to memory-resident buffers is still considerably faster than I/O to disk.

AWE limitations Besides forcing User processes to develop their own dynamic memory management routines, AWE has other limitations. For example, AWE regions and their associated reserved physical memory blocks must be allocated in pages. An AWE application can determine the page size using a call to *GetSystemInfo*. Physical memory can be mapped into only one process at a time. (Processes can still share data in non-AWE region virtual addresses.) Nor can a physical page be mapped into more than one AWE region at a time inside the same process address space. These limitations result from system virtual addressing constraints, which are significantly more serious when the */3GB* switch is in effect. Executable code (.exe files, .dll files, and so on) can be stored in an AWE region, but not executed from there. Similarly, AWE regions cannot be used as data buffers for graphics device output. Each AWE memory allocation must be also be freed as an atomic unit. It is not possible to free only part of an AWE region.

The physical pages allocated for an AWE region and associated reserved physical memory blocks are never paged out—they are locked in RAM until the application explicitly frees the entire AWE region (or exits, in which case the operating system will clean up automatically). Applications that use AWE must be careful not to acquire so much physical memory that other applications run out of memory to allocate. If too many pages in memory are locked down by AWE regions and the blocks of physical memory reserved for the AWE region overlays, contention for the RAM that remains can lead to excessive paging, or it can prevent creation of new processes or threads because of lack of resources in the system area's Nonpaged pool.

Application Support

Database applications like SQL Server, Oracle, and Exchange, which rely on memory-resident caches to reduce the amount of I/O operations they perform, are susceptible to running out of addressable private area in 32-bit Windows-based systems. These server applications all take advantage of the */3GB* boot switch for extending the process private area. Support for PAE is transparent to these server processes, allowing both SQL Server 2000 and IIS 6.0 to scale sideways. Both SQL Server and Oracle can also use AWE to gain access to additional RAM beyond their 4-GB limit on virtual addresses.

Scaling sideways SQL Server 2000 can scale sideways, allowing you to run multiple named instances of the Sqlserver process. A white paper entitled "Microsoft SQL Server 2000 Scalability Project—Server Consolidation," available at http://msdn.microsoft.com/library/en-us/dnsql2k/html/sql_asphosting.asp, documents the use of SQL Server and PAE to support multiple instances of the database process address, Sqlservr.exe, on a machine configured with 32 GB of RAM. Booting with PAE, this server consolidation project defined and ran 16 separate instances of Microsoft SQL Server. With multiple database instances defined, it was no longer necessary to use the */3GB* switch to extend the addressing capability of a single SQL Server address space.

Similarly, IIS 6.0 supports a new feature called *application processing pools*, also known as *Web gardens*. ASP and ASP.NET transactions can be assigned to run in separate application pools managed by separate copies of the W3wp.exe and aspnet_wp container processes.

Exchange Both Exchange 2000 and 2003 can take advantage of the */userva* and */3GB* switches in Boot.ini to allocate up to 3 GB of virtual memory for Exchange application use and 1 GB for the operating system. This is primarily to benefit the Store.exe process in Exchange; Store.exe is a database application that maintains the Exchange messaging data store. However, in Exchange 2000, Store.exe will not allocate much more than 1GB of RAM unless you change registry settings, because the database cache will allocate only 900 MB by default. Using the ADSI Edit tool, you can increase this limit by setting higher values for the *msExchESEParamCacheSizeMin* and *msExchESEParamCacheSizeMax* run-time parameters.

Exchange 2000 and 2003 will both run on PAE-enabled machines. However, these versions of Exchange do not make any calls to the AWE APIs to utilize virtual memory beyond its 4-GB address space.

Microsoft SQL Server Using SQL Server 2000 with AWE creates a set of special considerations. You must specifically enable the use of AWE memory by an instance of SQL Server 2000 Enterprise Edition by setting the *Awe Enabled* option using the *sp_configure* stored procedure. When an instance of SQL Server 2000 Enterprise Edition is run with *Awe Enabled* set to 1, the instance does not dynamically manage the working set of the address space. Instead, the database instance acquires nonpageable memory and holds all virtual memory acquired at startup until the SQL Server process address space is shut down.

Because the virtual memory acquired by the SQL Server process when the process is configured to use AWE is held in RAM for the entire time the process is active, the max server memory configuration setting should also be used to control how much memory is used by that instance of SQL Server.

Oracle You can enable AWE support in Oracle by setting the *AWE_WINDOW_MEMORY* registry parameter. Oracle recommends that AWE be used along with the */3GB* extended User area addressing boot switch. The *AWE_WINDOW_MEMORY* parameter controls how much of the 3-GB address space to reserve for the AWE region used to map database buffers. This parameter is specified in bytes and has a default value of 1 GB for the size of the AWE region inside the Oracle process address space.

If *AWE_WINDOW_MEMORY* is set too high, there might not be enough virtual memory left for other Oracle database processing functions—including storage for buffer headers for the database buffers, the rest of the SGA, PGAs, and stacks for all the executing program threads. As the Process(Oracle)\Virtual Bytes counter approaches 3 GB, out-of-memory errors can occur, and the Oracle process can fail. If this happens, you need to reduce *db_block_buffers* and the size of *AWE_WINDOW_MEMORY*.

Oracle recommends using the */3GB* option on machines with only 4 GB of RAM installed. With Oracle allocating 3 GB of private area virtual memory, the Windows operating system and any other applications on the system are squeezed into the remaining 1 GB. However, according to Oracle, the operating system normally does not need 1 GB of physical RAM on a dedicated Oracle machine, so there are typically several hundred MB of RAM available on the server. Enabling AWE support allows Oracle to access that unused RAM, perhaps grabbing as much as an extra 500 MB of buffers. On a machine with 4 GB of RAM, bumping up *db_block_buffers* and turning on the *AWE_WINDOW_MEMORY* setting, Oracle private virtual memory allocation might reach 3.5 GB.

SAS SAS support for PAE includes an option to place the Work library in extended memory to reduce the number of physical disk I/O requests. SAS is also enabled to take advantage of the */3GB* boot switch and use the extra private area addresses for SORT work areas.

System Virtual Memory Shortages

Operating system functions also consume RAM. The system has a working set that needs to be controlled and managed like any other process. The upper half of the 32-bit 4-GB virtual address range is earmarked for system virtual memory addresses. Using the */3GB* boot option, the system range can be limited to as little as 1 GB. On a 32-bit machine with a large amount of RAM, running out of virtual memory in the system address range is not uncommon. A multitude of important system functions—kernel threads, TCP session data, the file cache, and many other required system functions—allocate virtual memory in the system address range.

Critical system functions might be impacted when there is a shortage of 32-bit virtual addresses in the system range. For example, when the number of free System PTEs reaches zero, no function can allocate virtual memory in the system range. Unfortunately, you can sometimes run out of virtual addressing space in the Paged or Nonpaged pools before all the System PTEs are used up. The */3GB* boot option, which shrinks the range of system virtual addresses to 1 GB, sharply increases the risk of exhausting the system range of virtual addresses.

System Pools

The system virtual memory range, which is 2-GB wide, is divided into three major pools: the Nonpaged pool, the Paged pool, and the File Cache, as discussed in more detail in Chapter 5, "Performance Troubleshooting." The size of the three main system area virtual memory pools is determined initially based on the amount of RAM. There are pre-determined maximum sizes for the Nonpaged and Paged pools. However, there is no guarantee that they will reach their predetermined limits before the system runs out of virtual addresses. A substantial chunk of system virtual memory remains in reserve to be allocated on demand. How these areas held in reserve are allocated depends on which memory allocation functions overflow their original allocation targets and requisition the areas held in reserve first.

Nonpaged and Paged pool maximum extents are defined at system startup, based on the amount of installed RAM. The operating system's initial pool sizing decisions can also be influenced by a series of settings in the HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management key. (These settings are listed in Table 6-8.) Both *NonpagedPoolSize* and *PagedPoolSize* can be specified explicitly in the registry. Rather than force you to partition the system area exactly, the operating system allows you to set either the *NonpagedPoolSize* or *PagedPoolSize* to 0xffffffff for 32-bit and 0xffffffffffffffff for 64-bit Windows-based systems. (This is equivalent to setting a –1 value.) This setting instructs the operating system to allow the designated pool to grow as large as possible. Note that the Registry Editor does not allow you to assign a negative number, so you must instead set 0xffffffff on 32-bit systems and 0xffffffffffffffff on 64-bit systems. Detailed procedures for monitoring the size and composition of these pools are provided in Chapter 5, "Performance Troubleshooting."

Table 6-8 Settings for the HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management Key

Parameter	Default	Usage
NonpagedPoolSize	Defaults based on the size of RAM	Can be set explicitly. –1 extends *NonpagedPoolSize* to its maximum.
PagedPoolSize	Defaults based on the size of RAM	Can be set explicitly. –1 extends *PagedPoolSize* to its maximum.
LargeSystemCache	Defaults to 1, which favors the system working set over other process address space working sets	Can be set to 0 when server applications do not require the system file cache. Setting *LargeSystemCache* to 1 will tune Windows memory management for serving files, and setting it to 0 will tune for an application server.

Using the */3GB* or */userva* boot options, which shrink the system virtual address range in favor of a larger process private address range, substantially increases the risk of running out of system virtual memory. For example, the */3GB* boot option reduces the system virtual memory range to 1 GB and cuts the default size of the Nonpaged and Paged pools in one half for a given size RAM.

64-Bit Virtual Memory

All the capacity problems associated with virtual memory shortages discussed in this book disappear for programs running in 64-bit mode on 64-bit machines. The 64-bit architectures allow massive amounts of virtual memory to be allocated. Windows Server 2003 supports a 16-TB virtual address space for User-mode processes running in 64-bit mode. As in 32-bit mode, this 16-TB address space is divided in half, with User-mode private memory locations occupying the lower 8 TB, and the operating system occupying the upper 8 TB. Table 6-9 compares the virtual memory addressing provided in 64-bit Windows-based systems to 32-bit machines in their default configuration.

Table 6-9 Virtual Memory Addressing in 64-Bit and 32-Bit Machines

Component	64-Bit	32-Bit
Process virtual address space	16 TB	4 GB
User address space	8 TB	2 GB
System Paged pool	128 GB	470 MB*
System Nonpaged pool	128 GB	256 MB*
System cache	1 TB	1 GB
System PTEs	128 GB	660 MB
Hyperspace	8 GB	4 MB
Paging file size	512 TB	4 GB

* These are not hard limits on the size of the Paged and Nonpaged pools in Windows Server 2003, as discussed earlier in the section entitled "System Pools."

Running 64-bit systems can even help relieve the virtual memory constraints on 32-bit server applications. On 64-bit systems, all operating system functions are provided in native 64-bit code. Processes running 32-bit code use the WOW64 interface layer to communicate to operating system services. 32-bit User-mode applications that are compiled and linked with the *IMAGE_FILE_LARGE_ADDRESS_AWARE* switch enabled can allocate a private address space larger than 2 GB when running on 64-bit Windows-based systems. The */3GB* boot switch is not supported in 64-bit Windows-based systems. The larger potential 32-bit User process address space is possible because operating system functions no longer need to occupy the upper half of the total 4-GB process address space.

More Info See the Windows platform SDK documentation for details about converting your application code to run in native 64-bit mode. For advice on porting applications to 64-bit mode, see "Introduction to Developing Applications for the 64-bit Version of Windows" which is found at http://msdn.microsoft.com/library/en-us/dnnetserv/html/ws03-64-bitwindevover.asp in the SDK documentation.

Forecasting Memory Requirements

Considering the way Windows Server 2003 virtual memory management works, it should be apparent that physical memory utilization cannot be measured in the same way that processor, disk, or network line utilization, for example, is measured.

For instance, physical memory capacity is not consumed in quite the same way that processor capacity is. Physical memory is allocated on demand and occupied continuously until it is either abandoned by its original user or replaced by a more recent consumer. That means that at any time, some memory locations might be allocated but not actively used. For example, often, infrequently accessed resident pages of a process working set occupy physical memory even as these pages go unreferenced for some period of time. The fact that infrequently accessed virtual memory pages can often be trimmed and replaced by current working set pages with very little impact on performance is vivid evidence that these memory locations are not being actively utilized.

Memory allocation statistics might show that physical memory is full, but such indicators of a physical memory constraint are not foolproof. Worse, the goal of virtual memory management is to utilize all available physical memory. Consequently, virtual memory systems often report (close to) 100 percent memory occupancy on a continuous basis. Some of these physical memory locations typically are occupied by relatively infrequently referenced virtual pages, whereas others contain pages that are effectively in continuous use.

Over and above the fact that physical memory might be 100 percent allocated, the performance of virtual memory systems becomes a concern only when physical memory is over-committed and high paging rates result. Conceptually, this situation results when the size of active process working sets overflows the amount of physical RAM installed. Because physical memory is allocated on demand and 100 percent memory occupation is the rule rather than the exception, you need additional measures that capture this dynamic aspect of virtual memory management.

The root cause of a system that is paging excessively is too little physical memory to support the range of virtual addresses that active processes are routinely accessing. Instead of memory allocation statistics, we need measures of memory contention that can reveal the extent of physical memory demand generated by active virtual memory address spaces. Windows Server 2003 does not directly report on memory utilization in this sense. Instead, the available measurements merely report how much of the physical memory is currently allocated, and which processes or parts of the system working set are occupying physical memory at a particular moment. Nevertheless, you can easily derive two measures of *physical memory contention* that can be quite useful in predicting the onset of a serious paging problem or other physical memory constraint.

V:R Ratio

In addition to monitoring virtual memory allocation statistics, physical memory occupancy, and paging activity, developing predictive measures is also useful. One useful calculation is to derive a memory contention index that will correlate with the level of paging activity observed.

As noted in Chapter 5, "Performance Troubleshooting," the Available Bytes counter correlates with hard paging activity reasonably well. But monitoring Available Bytes alone cannot reliably predict the rate of hard paging activity to disk that might occur. From a capacity planning standpoint, the correlation between Available Bytes and paging rates to disk breaks down once physical memory is full.

As discussed in Chapter 1, "Performance Monitoring Overview," page trimming by the virtual memory manager is triggered by a drop in the number of Available Bytes. Page trimming attempts to replenish the pool of Available Bytes by identifying virtual memory pages that have not been referenced for a relatively long time. When page trimming is effective, older pages that are trimmed from process working sets are not needed again soon. These older pages are replaced by more active, recent pages. Trimmed pages are marked in transition and remain in RAM for an extra period of time, so very little paging to disk needs to occur.

However, if there is a critical, long-term shortage of RAM, page trimming loses effectiveness. This condition is marked by more paging operations to disk. There is little room for pages marked in transition to remain in RAM, so when recently trimmed pages are re-referenced, they often must be accessed from disk instead. In this scenario, the number of Available Bytes oscillates between the low and high threshold values that trigger page trimming. This was illustrated in several examples in Chapter 5, "Performance Troubleshooting."

Consider that the threshold-driven page trimming process works to fill up memory with recently referenced pages. Comparing a system in which the supply of RAM is ample to one in which RAM is scarce, you might observe that the average number of Available Bytes is strikingly similar, even though each system's paging rate to disk is quite dissimilar. In this type of situation, obtaining additional measures is desirable so that you have greater ability to predict the onset of a paging performance problem.

A useful working hypothesis for developing a memory contention index is that demand paging activity, in general, is caused by virtual memory address spaces contending for limited physical memory resources. Consider a memory contention index computed as the ratio of virtual memory allocated to the amount of physical memory installed, or V:R. This V:R ratio is easily computed in Windows-based systems by dividing Committed Bytes by the amount of installed RAM:

V:R = Committed Bytes ÷ sizeof(installed RAM)

One virtue of this memory contention index is that it is easily computed and has a straightforward interpretation. It measures the full range of virtual addresses created dynamically by process address spaces and compares it to the static amount of RAM configured on the system. (Note that virtual addresses of different processes that can correspond to the same physical addresses (for example, those holding .dll code pages) are not considered by this metric.)

Consider the case when V:R is 1 or less. This means that for every Committed Byte of virtual memory that processes have allocated, there is a byte on a corresponding 4-KB page somewhere in RAM. Under these circumstances, it makes sense that very little demand paging activity would occur. When V:R is near 1, Windows Server 2003 hard paging rates are usually minimal. The demand paging activity that occurs is usually confined to processes creating (and later destroying) new pages, reflected in the rate of Demand zero pages/sec that you are able to observe. Because RAM has room for just about every virtual memory page that processes allocate, you would expect to see very little hard page fault processing. There is very little contention for physical memory. Of course, even when there is little or no physical memory contention, you might still see large amounts of transition (soft) faults resulting from memory management, cache faults (depending on the type of application file processing that is occurring), and demand zero page faults (depending on the applications that are running).

Installing enough RAM to enable every process virtual memory page to remain resident in physical memory is not necessary. Some virtual memory addresses are usually inactive at any given time, so there is no need to keep infrequently accessed virtual memory pages in physical memory all the time. Some amount of hard page fault activity is certainly acceptable in most environments. However, when processes begin to allocate more virtual memory pages than can fit comfortably in RAM, the page replacement algorithms of Windows Server 2003 are forced to start to juggle the contents of RAM more—trimming inactive pages from these applications and not allowing those process working sets to grow. At this point, the number of Committed Bytes has grown larger than the amount of RAM, which is why Windows Server 2003 memory management routines need to work harder. When the number of virtual memory pages allocated by currently executing processes exceeds the amount of RAM installed, the likelihood that executing processes will access pages that are not currently resident in physical memory, causing hard pages faults to occur, increases. Naturally, when the number of hard page faults processed becomes excessive, it is time to add more RAM to the system.

Even the simple V:R ratio described here, computed as Committed Bytes/Installed RAM, can have surprising predictive value. Moreover, it is relatively easy to trend the V:R ratio as process virtual memory loads increase, allowing you to anticipate the need for a memory upgrade. For Windows servers with limited I/O bandwidth, it is recommended that you confine the amount of paging activity performed on physical disks shared with application data files to 20–40 percent or less of the I/O processing bandwidth, as discussed in Chapter 3, "Measuring Server Performance." On systems in which you maintain a V:R ratio of 1.5:1 or less, you will normally observe paging activity that is usually well within these configuration guidelines. It is recommended that you monitor the V:R ratio on a regular basis and intervene (where possible) to add RAM once the memory contention index passes 1.5:1 and before it reaches a value of 2:1. By doing so, you can ensure that paging activity to disk remains at acceptably low levels.

Unfortunately, trending V:R is not a foolproof technique for predicting paging activity. One problem is that applications like SQL Server protect their working sets from Windows Server 2003 memory management and perform their own equivalent virtual memory management. Another problem you might encounter when applying this technique is that applications performing predominantly static virtual memory allocations often allocate overly large blocks of virtual memory at initialization, instead of allocating only what they need on demand. In these cases, Committed Bytes remains static, failing to reflect the dynamic aspect of virtual memory accesses accurately. But the technique works well enough in the general case to keep you out of major trouble on most of the servers you are responsible for.

Paged Pool Contention

Another memory contention index exists that can be readily calculated. In addition to being computable from existing memory counters, this memory contention index is more likely to reflect the dynamic aspects of virtual memory demand than the simple Committed Bytes:installed RAM ratio discussed earlier. Calculate:

Paged Pool contention = Memory\Pool Paged Bytes /Memory\Pool Paged Resident Bytes

The system's pageable pool is the source for all allocations that operating system services make for virtual memory that does not need to reside permanently in RAM. Pool Paged Bytes reports the current amount of virtual memory allocated within the system's pool of pageable virtual memory. Pool Paged Resident Bytes reports the current number of pages within that pool that are resident in RAM. Note that this metric does not consider that pageable data can be shared by multiple processes (for example, shared data pages).

The Pool Paged Bytes:Pool Paged Resident Bytes ratio reports the amount of Committed pageable virtual memory allocated by operating system services, compared to the amount of physical memory those pages currently occupy. Pool Paged Bytes over and above the number of Pool Paged Resident Bytes represent committed operating system virtual memory pages currently stored on the paging file (or files). As this ratio increases, operating system services are likely to encounter increased contention for physical memory, with higher paging rates being the likely result.

One advantage of this specific memory contention index is that operating system services tend to be well-behaved users of virtual memory, which they allocate only on demand. Thus, the Pool Paged Bytes:Pool Paged Resident Bytes ratio is likely to be a good indicator of current memory contention, albeit limited to privileged operating system services. Because working storage for shared DLL library routines is also often allocated from the system's pageable pool, the index also reflects some degree of per-process virtual memory demand.

An example will illustrate how this memory contention index behaves, especially with regard to its ability to predict demand paging rates. In Figure 6-21, the Pool Paged Bytes:Pool Paged Resident Bytes memory contention index was calculated for a machine showing sharp spikes in hard paging activity. Figure 6-21 shows that this memory contention index is well correlated to the actual demand paging activity observed. The exponential trendline that is also plotted is quite representative of the performance of virtual memory systems.

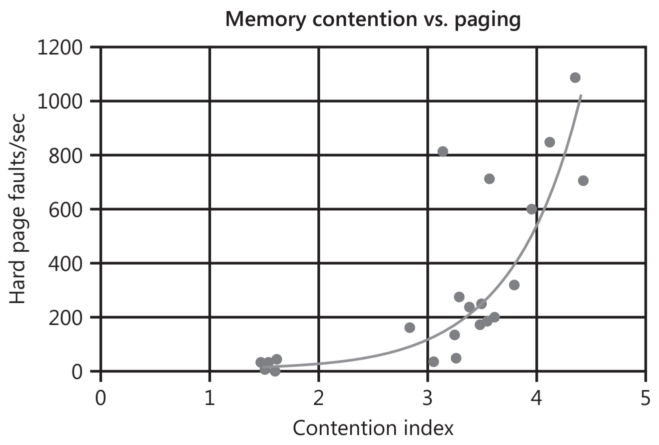


Figure 6-21 Pool Paged Bytes:Pool Paged Resident Bytes memory contention index calculator for a machine showing sharp spikes in hard paging activity

Figure 6-21 illustrates three distinct operating regions for this workload. When the Pool Paged Bytes:Pool Paged Resident Bytes ratio is approximately 1.5, the great majority of virtual memory pages allocated in this pool fit readily into available RAM. This corresponds to intervals in which very few hard pages faults are occurring. When the Pool Paged Bytes:Pool Paged Resident Bytes ratio rises to 3.0–3.5, greater memory contention and substantially higher paging rates are observed, clustered around 200 hard page faults processed/sec. Finally, at even higher Pool Paged Bytes:Pool Paged Resident Bytes ratios, the amount of paging varies drastically, reaching a maximum observed value above 1000 hard page faults/sec. This is highly suggestive of virtual memory management struggling to keep up with the demand for physical memory.

A memory utilization curve relating memory contention to paging activity can have predictive value, especially when the trendline is well correlated with the underlying data, as in the example plotted in Figure 6-21. For example, the trendline drawn in Figure 6-21 can be used to justify a memory upgrade for machines running similar workloads as the contention index begins to approach 3. Because this memory contention index is easy to trend, you can use it to forecast future virtual memory demand. For instance, if you were to project that a growing system workload will likely reach an undesirable level of paging sometime in the next 3 months, you can schedule a memory upgrade in anticipation of that event.

The System Monitor Automation Interface

Of the tools available to troubleshoot performance problems or support capacity planning, the graphical approach used in System Monitor is probably the one you will rely on more frequently than any other. You can take advantage of System Monitor's graphical capabilities while retaining the ability to script custom performance monitors. This is because System Monitor is not an executable program; instead, it is an ActiveX control embedded within the management console. This control, whose properties and methods are collectively known as the System Monitor Automation Interface, is fully scriptable; you can add the control to any application capable of hosting ActiveX objects, and then use the System Monitor properties and methods to fully configure a custom monitoring session.

Among other tasks, the System Monitor Automation Interface enables you to:

- Print performance graphs. The Performance console does not include a print function. If you need to print a performance graph, you must press ALT+PRINT SCREEN to take a snapshot of the graph, open Microsoft Paint, paste in the snapshot, and then print the graph from Paint. If the System Monitor ActiveX control is embedded in a Web page, however, you can print the graph by using the Print command in your browser.
- Create custom Performance Monitors that, upon startup, immediately begin monitoring a specified set of performance counters.
- Create custom performance monitors with enhanced functionality. For example, your performance monitor could include a script that sends e-mail to a technical support specialist. Users noticing anomalies in performance could click a button, and an e-mail message that lists the computer and user name, along with the performance counters and their values, could be sent to technical support.

This section is intended to highlight the major features of the System Monitor Automation Interface so that you can easily accomplish tasks like these.

More Info A complete reference to the System Monitor Automation Interface is available at http://msdn.microsoft.com/library/en-us/perfmon/base/system_monitor_automation_interface.asp. The complete reference manual describes the full System Monitor Automation Interface and the properties it supports, and provides the complete set of interface events.

Adding the System Monitor ActiveX Control to a Web Page

The System Monitor ActiveX control can be added to any application capable of hosting ActiveX controls. In many respects, the most logical approach is to place the control within a Web page. This enables you to use a Web browser to access your custom monitors from anywhere on the network; it also seems more intuitive to do performance monitoring from a Web interface rather than from within, say, a Microsoft Word document. Because of that, this discussion will focus on using the System Monitor ActiveX control within a Web page.

Important You cannot create a standalone script, such as a Microsoft Visual Basic Script (VBScript) file, that is able to retrieve performance data with the System Monitor methods and properties. The ActiveX control must be present in the document or application. To monitor performance from a standalone script, use the WMI performance counters. You can place the System Monitor ActiveX control inside a Microsoft Office Word document, for example, but Word documents can display only static data. For real-time displays of performance data that can be updated continuously, use a Web page.

To add the System Monitor ActiveX control to a Web page, include the code shown in Listing 6-1 within the page's <BODY> tag.

Listing 6-1 Adding the System Monitor ActiveX Control to a Web Page

```
<OBJECT
CLASSID="clsid:C4D2D8E0-D1DD-11CE-940F-008029004347" ID="MyMonitor">
</OBJECT>
```

Note In Listing 6-1, the control has been assigned an ID of *MyMonitor*. Throughout this discussion, this ID is used in the code samples. You can assign a different ID to the control when you create your own performance monitors. If you copy and use any of the code from this section, however, be sure to change all references to *MyMonitor* to the ID you assign to the control.

After the control has been added to a Web page, you can use the properties and methods of the System Monitor Automation Interface to customize both the look and feel of your new performance monitor.

Customizing the System Monitor ActiveX Control

Performance can be monitored with the Performance console, without additional scripting or coding. But the real value of the System Monitor ActiveX control is that it allows you to create a customized performance monitor. When you create a customized performance monitor, you can perform such tasks as:

- Distribute the monitor to other users, enabling them to monitor performance without having to understand how to set up and configure System Monitor
- Predefine the performance counters to be measured

- Provide a clean and simple user interface
- Prevent users from making changes to the monitor, which helps ensure accurate and consistent performance measurement

To help you customize your performance monitors, the System Monitor Automation Interface allows you to specify such attributes as:

- How users will view the data
- What values will be displayed to the users
- How often values will be measured
- What the System Monitor user interface will look like

An example of the HTML code for a Web page that includes a fully configured System Monitor control is shown in Listing 6-3. Within the code, you will see lines similar to those shown in Listing 6-2.

Listing 6-2 Sample Commands for Configuring the System Monitor Control

```
Sub Window_onLoad()
    MyMonitor.Counters.Add "\Memory\Available Bytes"
    MyMonitor.ReadOnly = True
    MyMonitor.UpdateInterval = “5"
End Sub
```

The lines of code shown in Listing 6-2 (which must be included within a <SCRIPT> tag when embedding the control within a Web page) are used to configure the System Monitor control. In this section, all the code samples are presented under the assumption that you will be using them within a <SCRIPT> tag on a Web page. Although the code for configuring System Monitor can be placed in any function or procedure on your Web page, it is common to include this code as part of the *Window_onLoad()* procedure. The code included in this procedure is automatically executed each time the Web page is loaded. By placing your System Monitor code in this procedure, you can be assured that users will have a fully functioning and configured System Monitor each time they load the page.

Listing 6-3 Complete HTML Code for a Custom System Monitor

```
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Window_onLoad()
    MyMonitor.Counters.Add "\Memory\Available Bytes"
    MyMonitor.ReadOnly = True
    MyMonitor.UpdateInterval = "5"
    MyMonitor.BackColor = vbBlack
    MyMonitor.ForeColor = vbWhite
End Sub
```

```
-->
</SCRIPT>
<OBJECT CLASSID="clsid:C4D2D8E0-D1DD-11CE-940F-008029004347"
ID="MyMonitor" WIDTH="100%" height="100%">
</OBJECT>
</BODY>
```

Configuring the System Monitor ActiveX Control Display Type

Like System Monitor, the ActiveX control can display data using three different views:

Graph Charts performance over time. This allows you to see how performance has fluctuated over a given time interval. The Graph view is less useful, however, for counters that aggregate performance over time. For example, many of the server performance counters measure total errors (Errors Logon, Errors Granted Access, Errors Access Permissions). These cumulative statistics will never decrease; as a result, you might be more interested in seeing the current value rather than a steadily increasing array of values.

Unless otherwise specified, data is displayed as a line graph.

Histogram Displays a single value for each counter in bar chart format. The Histogram view allows you to quickly determine the current value for any counter, but does not allow you to see how those values have fluctuated over time.

Report Shows the current value for each counter being monitored. In Report view, numerical values are displayed without any accompanying graphics.

You can configure the display type for your System Monitor control as part of your script. For example, to display data in report form, use this command:

```
MyMonitor.DisplayType = 3
```

Valid System Monitor display values are shown in Table 6-10.

Table 6-10 System Monitor Display Values

DisplayType	Description
1	Displays data as a histogram
2	Displays data as a line graph
3	Displays data in report form

Note Switching between display types will not result in a loss of data. For example, if you are viewing data as a line graph, you will see multiple data points for each counter. If you switch to Report view, you will see only the latest data point for each counter. Upon switching back to the line graph view, however, all your previously measured counter values will again be visible.

When you choose the Histogram or the Report display types (the views that display a single value for each counter), you can also choose whether the value displayed is the current value, the average value, the minimum value, or the maximum value. You can do this by setting the *ReportValueType* property to one of the valid *ReportValueTypes* or one of the VBScript constants shown in Table 6-11. For example, either of the following commands can be used to display only the minimum value for a counter:

```
MyMonitor.ReportValueTypeType = SysmonMinimum
MyMonitor.ReportValueTypeType = 3
```

Table 6-11 System Monitor Report Value Types

ReportValue-Type	VBScript Constant	Description
0	*SysmonDefault-Value*	The value displayed depends on the data source being used. If you are monitoring real-time performance data, the current value for the counter will be displayed. If you are displaying data from an existing performance log, the average (mean) value for the counter will be displayed.
1	*Sysmon-CurrentValue*	Reports the last measured value for a counter. If you are retrieving information from a performance log, this will display the value for the last data bucket in the log file.
2	*SysmonAverage*	Average (mean) value for the counter for the display interval is calculated and displayed.
3	*SysmonMinimum*	The smallest measured value for the counter for the display interval is displayed. This is useful when monitoring an item such as available bytes of memory. In a case like that, you might be less interested in monitoring normal fluctuations in memory use than you are in ensuring that available memory does not fall below a certain level.
4	*SysmonMaximum*	The largest-measured value for the counter for the display interval is displayed. This is useful when monitoring an item such as the amount of time a disk drive is in use. In a case like that, you might be less interested in monitoring normal fluctuations in disk use than you are in ensuring that disk usage does not exceed a certain level.

Note that the System Monitor *ReportValueType* constants apply to both Report and Histogram displays.

Configuring the System Monitor ActiveX Control Sampling Rate

When setting the display type and the report value type for your custom performance monitor, you might also want to configure the sampling rate. By default, the System Monitor ActiveX control collects a new performance sample every 15 seconds. To specify the number of seconds between samples, set the *UpdateInterval* property to an integer that specifies the number of seconds between data collection intervals. For example, this command sets the sampling rate to once every 60 seconds:

```
MyMonitor.UpdateInterval = 60
```

Manually Retrieving Performance Data

The sampling rate determines how often the System Monitor control automatically collects a new set of performance data. By changing the sampling rate, you can control when System Monitor will automatically collect the next set of measurement data.

In addition to changing the sampling rate, however, you can set the *ManualUpdate* property and then use the *CollectSample* method to retrieve performance data manually every time you click a button. First, create a button on your Web page. Then, in the button's Event handler, call the *CollectSample* method each time the button is pressed. Listing 6-4 shows an example of a procedure that might be called when an Update Now button is pressed.

Listing 6-4 Calling the *CollectSample* Method

```
Sub Window_onLoad()
…
    MyMonitor.ManualUpdate = True
End Sub
.
.
.
Sub UpdateSamples()
    MyMonitor.CollectSample
End Sub
```

Note that when the *ManualUpdate* property is True, the value specified in the *UpdateInterval* property is ignored.

Of course, if the System Monitor control can automatically collect data, you might wonder why you would want to manually collect data. There are two primary reasons:

- Within some applications, the automated data collection process is not fully reliable. For example, it is recommended that you use manual data collection if you embed the System Monitor control in a Microsoft Office document.
- When you manually collect data, you can take advantage of the *OnSampleCollected* method.

The *OnSampleCollected* method (if present) is automatically called each time data is collected using the *CollectSamples* method. Note that your *OnSampleCollected* routine is not called when samples are gathered automatically. After the data has been collected, you can include *OnSampleCollected* code to examine the data that was retrieved, and then take some sort of action. For example, the code shown in Listing 6-5 checks the value of *MyMonitor.Counters(1)*, and alerts the user when the value is less than 4 MB. (The ability to reference individual performance counters is discussed later in the "Configuring System Monitor ActiveX Control Performance Counters" section.)

Listing 6-5 Using the Collect Sample Method

```
Sub MyMonitor_OnSampleCollected()
    If MyMonitor.Counters(1).Value < 4000000 Then
        Wscript.Echo "Available memory is less than four megabytes."
    End If
End Sub
```

Note The *OnSampleCollected* method is invoked only for manual data collection using the *CollectSamples* method.

Other event methods available in the System Monitor Automation Interface are listed in Table 6-12. By using these event methods, you can create scripts that do such things as perform an action each time a user selects one of the counters in the graph legend, or each time a user adds or deletes a counter.

Table 6-12 System Monitor Event Methods

Event Method	Description
OnSampleCollected	Occurs whenever samples are collected using the *CollectSamples* method.
OnCounterAdded	Occurs whenever a new counter is added to the System Monitor control.
OnCounterDeleted	Occurs whenever an existing counter is deleted from the System Monitor control.
OnCounterSelected	Occurs whenever a counter is selected in the System Monitor control. Counters are selected by clicking the counter name in the legend.
OnDblClick	Occurs whenever a counter is double-clicked in Graph, Histogram, or Report view.

For example, the script shown in Listing 6-6 displays a message box each time a counter has been deleted.

Listing 6-6 Using the *OnCounterDeleted* Method

```
Sub MyMonitor_OnCounterDeleted()
    Wscript.Echo "A counter has been deleted."
End Sub
```

The *OnCounterAdded*, *OnCounterDeleted*, and *OnCounterSelected* methods all return the index value that identifies the affected counter from the *Counters* collection of *CounterItems*. (For more information about *CounterItems* and the *Counters.Add* method, see "Adding Performance Counters to the System Monitor ActiveX Control" later in this chapter.)

Configuring the System Monitor ActiveX Control's Appearance

When you create a custom performance monitor using the System Monitor ActiveX control, you can decide in advance which portions of the user interface will be available to your users. For example, you can configure any of the user interface items shown in Table 6-13.

Table 6-13 System Monitor User Interface Properties

Property	Value	Description
ShowLegend	True/False	On-screen legend that matches the lines in the graph with the name of the associated performance counters. For example, the legend might tell you that the red line represents \Memory\Available Bytes.
ShowHorizontalGrid	True/False	Displays horizontal gridlines in the Histogram and line Graph views.
ShowVerticalGrid	True/False	Displays vertical gridlines in the Histogram and line Graph views.
YAxisLabel	Character string	Retrieves or sets the label of the vertical y-axis of the Histogram and line Graph views.
ShowToolBar	True/False	Displays buttons that allow the user to add and delete counters, change the display type, and so on. Hiding the toolbar will not prevent users from making these changes; users can still right-click the graph and make these changes by using the context menu.

Table 6-13 System Monitor User Interface Properties

Property	Value	Description
ShowValueBar	True/False	Displays the last, minimum, maximum, and average values for the currently selected counter. Counters are selected whenever a user clicks the counter name in the legend or double-clicks the counter in the Chart, Histogram, or Report view.
GraphTitle	Any valid character string	Title displayed for the histogram, or the line graph.

For example, the lines of code shown in Listing 6-7 configure System Monitor to display the legend, hide the toolbar, hide the value bar, and set the name of the graph to "Disk Drive Performance Monitor."

Listing 6-7 Configuring the System Monitor User Interface

```
Sub Window_OnLoad()
    MyMonitor.ShowLegend = True
    MyMonitor.ShowToolbar = False
    MyMonitor.ShowValueBar = False
    MyMonitor.GraphTitle = "Disk Drive Performance Monitor"
End Sub
```

Configuring the System Monitor ActiveX Control Color Schemes

You can also specify the colors used when performance data is displayed. The color properties you can configure are described in Table 6-14.

Table 6-14 System Monitor Color Properties

Property	Description
BackColor	Color of the graph, chart, or report background (the area where data is displayed). If not specified, *BackColor* color defaults to the background color of the container.
BackColorCtl	Color of the area surrounding the graph, chart, or report (the area where the value labels, axis labels, and so on are displayed). If not specified, *BackColorCtl* defaults to the system button face color.
ForeColor	Color of the font used throughout the display. If not specified, *ForeColor* defaults to the system color defined for button text.
GridColor	Color of the gridlines (if shown) used in Graph or Chart view. If not specified, *GridColor* is dark gray.
TimeBarColor	Color of the vertical bar that moves across the display, indicating the passage of time. If not specified, *TimeBarColor* is red.

To set the color for a System Monitor component, you can use the Visual Basic color code, the RGB color code, or the hexadecimal color code. For example, Listing 6-8 shows three ways to set the font color to blue.

Listing 6-8 Three Options for Setting the System Monitor Font Color

```
MyMonitor.ForeColor = vbBlue
MyMonitor.ForeColor = RGB(0,0,255)
MyMonitor.ForeColor = &hFF0000
```

Color codes for some of the more commonly used colors are shown Table 6-15.

Table 6-15 System Monitor Color Codes

Color	VB Color Code	RGB Value	Hexadecimal Value
Black	*vbBlack*	0,0,0	&h00
Blue	*vbBlue*	0,0,255	&hFF0000
Cyan	*vbCyan*	0,255,255	&hFFFF00
Green	*vbGreen*	0,255,0	&hFF00
Magenta	*vbMagenta*	255,0,255	&hFF00FF
Red	*vbRed*	255,0,0	&hFF
White	*vbWhite*	255,255,255	&hFFFFFF
Yellow	*vbYellow*	255,255,0	&hFFFF

Note You can determine RGB values by using Microsoft Paint. In Paint, double-click any color in the color palette. In the Edit Colors dialog box, click Define Custom Colors. When you click any color in the dialog box, the RGB values are displayed in the boxes labeled Red, Green, and Blue.

You can also reference any of the standard Visual Basic color constants like *vbBack-groundWindow* or *vbWindowsText*, or their .NET Framework equivalents.

Configuring the System Monitor ActiveX Control Font Styles

You can control font size, font style, font color, and many other font properties when you create a custom performance monitor. The only limitation is that all the fonts in your System Monitor ActiveX control must have the same characteristics; for example, you cannot use Arial for the data labels and Times Roman for the graph labels. (Both the System Monitor control and the System Monitor in the Performance console have the same limitation.)

Table 6-16 lists some of the font properties that you can configure for your System Monitor control. (For a complete list, refer to the *FontObject* property in Visual Basic Help.) When configuring fonts, remember that the System Monitor font color is configured using the *ForeColor* property rather than the *Font* property.

Table 6-16 System Monitor Font Properties

Property	Value
Font	Set to the desired font name
Bold	True or False
Italic	True or False
Size	Set to the desired point size (from 1 through 2048)

For example, the code shown in Listing 6-9 sets the font for your System Monitor control to 12-point red Trebuchet bold.

Listing 6-9 Setting System Monitor Font Properties

```
Sub Window_onLoad()
    MyMonitor.Font.Size = 12
    MyMonitor.Forecolor = vbRed
    MyMonitor.Font = "Trebuchet"
    MyMonitor.Font.Bold = True
End Sub
```

If you do not set the font properties, the default font for the application hosting the System Monitor control is used.

Adding Performance Counters to the System Monitor ActiveX Control

Of course, even the most aesthetically pleasing System Monitor control is of little value unless it actually monitors performance. Before you can monitor performance with your custom control, you must use the *Counters.Add* method to specify the performance counters to be gathered.

The *Counters.Add* method can be called from anywhere at any time. For example, you could have a button on your page that, when clicked, causes counters to be added to the System Monitor control. Monitoring would then start as soon as a counter is added to the control.

In many cases, however, you will want to begin monitoring performance immediately, without user intervention. To do that, include the *Counters.Add* method within a *Window_onLoad* script. (As noted earlier, the *Window_onLoad* script, if present, is automatically run each time a Web page is loaded.) For example, the script shown in Listing 6-10 adds several Memory counters as part of the *Window_onLoad* script. As a result, each time this Web page is loaded, System Monitor immediately begins monitoring these Memory counters.

Listing 6-10 Adding Multiple Performance Counters

```
Sub Window_onLoad()
    MyMonitor.Counters.Add "\Memory\Available Bytes"
    MyMonitor.Counters.Add "\Memory\Pages/sec"
    MyMonitor.Counters.Add "\Memory\Cache Bytes"
End Sub
```

Configuring System Monitor ActiveX Control Performance Counters

You can use the *Counters* collection to track each counter added to your performance monitor. Counters are assigned an index number based on the order in which they are added to the control. For example, suppose you add three performance counters by using the lines of code shown in Listing 6-16.

Listing 6-11 Adding New Performance Counters

```
Sub Window_onLoad()
    MyMonitor.Counters.Add "\Memory\Available Bytes"
    MyMonitor.Counters.Add "\Memory\Pages/sec"
    MyMonitor.Counters.Add "\Memory\Cache Bytes"
End Sub
```

In this example, note the following:

- The \Memory\Available Bytes counter—the first counter added to the graph—is designated *MyMonitor.Counters(1)*.
- The \Memory\Pages/sec counter is designated *MyMonitor.Counters(2)*.
- The \Memory\Cache Bytes is designated *MyMonitor.Counters(3)*.

With these index numbers assigned, you can use the *Counters* collection to obtain information about a specific counter, and to set the properties for a specific counter. For example, this command displays the value for the \Memory\Available Bytes counter in a message box:

```
Wscript.Echo MyMonitor.Counters(1).Value
```

To display the fully qualified path name for the second counter in the *Counters* collection, use this command:

```
Wscript.Echo MyMonitor.Counters(2).Path
```

The individual counter properties available to you are shown in Table 6-17.

Table 6-17 System Monitor Performance Counter Properties

Property	Description
Color	Sets the color used when displaying the counter in either line Graph or Histogram view.
LineStyle	Sets the style of the line used when displaying the counter in either line Graph or Histogram view. Valid line styles include: 0—Solid line 1—Dashed line 2—Dotted line 3—Dotted line with alternating short and long segments 4—Dotted line with alternating dashes and double dots
Path	Retrieves the path name of the counter.
ScaleFactor	Sets or retrieves the scale factor used when graphing the counter values.
Value	Retrieves the current value of a counter.
Width	Sets the width of the line used in the line Graph and Histogram views. Widths can range from 1 pixel through 9 pixels.

The lines of code shown in Listing 6-12 display the first counter in the collection as a red dotted line, with a thickness of 2 pixels.

Listing 6-12 Configuring Performance Counter Properties

```
Sub Window_onLoad()
    MyMonitor.Counters(1).Color = vbRed
    MyMonitor.Counters(1).LineStyle = 2
    MyMonitor.Counters(1).Width = 2
End Sub
```

The *Counters.Add* method also returns a reference to the *CounterItem* that was just added. You can, if you choose, save this *CounterItem* in a local variable.

Removing Performance Counters from the System Monitor ActiveX Control

Performance counters can be removed from the System Monitor control by calling the *Counters.Remove* method and referencing the item number within the *Counters* collection. This can be done by including a button on your Web page that, when clicked, deletes a particular counter. For example, the script shown in Listing 6-13 removes the \Memory\Pages/sec counter from *MyMonitor*.

Listing 6-13 Removing a Performance Counter Using the Counter Index

```
Sub RemovePagesPerSec()
    MyMonitor.Counters.Remove(2)
End Sub
```

To remove all the counters in a single operation, use the *Reset* method, as shown in Listing 6-14.

Listing 6-14 Removing All Performance Counters Using the *Reset* Method

```
Sub RemoveAllCounters()
    MyMonitor.Reset
End Sub
```

Using Counter Paths to Track Individual Performance Counters

When you remove performance counters, the *Counters* collection index number for the remaining counters might change as well. For example, suppose you add three counters to the performance monitor in this example, as shown in Listing 6-15.

Listing 6-15 Adding New Performance Counters

```
Sub Window_onLoad()
    MyMonitor.Counters.Add "\Memory\Available Bytes"
    MyMonitor.Counters.Add "\Memory\Pages/sec"
    MyMonitor.Counters.Add "\Memory\Cache Bytes"
End Sub
```

Referencing individual counters by their counter numbers (as shown earlier in this chapter in "Configuring System Monitor ActiveX Control Performance Counters") works well, provided counters are neither added nor deleted during a monitoring session. However, if you delete a counter, the index numbers will be decremented (because there are now only two counters to be assigned numbers), and individual counters might be reassigned new index numbers. For example, suppose you remove the first counter. In that case:

- \Memory\Pages/sec will be reassigned index number 1.
- \Memory\Cache Bytes will be reassigned to index number 2.
- There will no longer be a counter assigned to index number 3.

As a result, any code you have that explicitly refers to *MyMonitor.Counters(3)* will fail.

One way to work around this problem is to cycle through all the counters in the current collection, and see whether any match the appropriate counter path. For example, the code shown in Listing 6-16 cycles through the counters, looking for one with a path of \Memory\Available Bytes. If the counter is found, a message box is displayed.

Note *MyMonitor.Counters.Count* retrieves the total number of counters currently assigned to the System Monitor control.

Listing 6-16 Identifying Performance Counters by Path

```
Sub CheckForAvailableBytes()
    For i = 1 to MyMonitor.Counters.Count
        If MyMonitor.Counters(i).Path = "\Memory\Available Bytes" then
            Wscript.Echo "\Memory\Available Bytes is in the current collection."
        End if
    Next
End Sub
```

Creating a Web Page for Monitoring Performance

Figure 6-22 shows an example of a Web page hosting the System Monitor control. This custom performance monitor has the following properties:

- The only performance counter being monitored is \Memory\Available MBytes (*MyMonitor.Counters.Add* "\Memory\Available MBytes").
- The control has been designed to fill the entire display (height equals 100% and width equals 100%).
- The color of the data display area (*BackColor*) has been set to white (*vbWhite*).
- The font color (*ForeColor*) has been set to black (*vbBlack*).
- The sampling rate (*UpdateInterval*) has been set to 5 seconds.
- The y-axis maximum has been set to 800.
- The default width of the line graph has been set to 4 pixels.
- Horizontal and vertical gridlines have been enabled.
- A title has been added to the graph.
- The control's properties cannot be changed interactively (*MyMonitor.ReadOnly* = *True*).

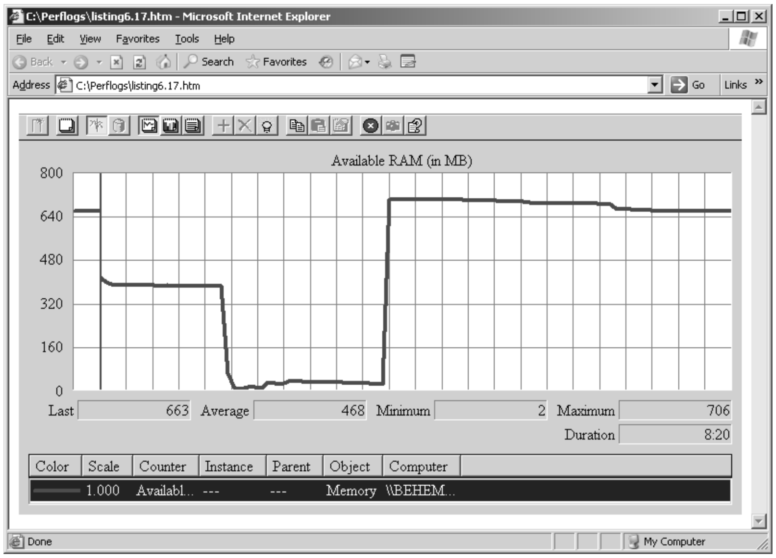

C:\Perflogs\listing6.17.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Perflogs\listing6.17.htm
Go Links
Available RAM (in MB)
800
640
480
320
160
0
Last 663 Average 468 Minimum 2 Maximum 706
Duration 8:20
Color Scale Counter Instance Parent Object Computer
1.000 Availabl... --- --- Memory \\BEHEM...
Done
My Computer

Figure 6-22 A Web page hosting the System Monitor control

To create the example performance monitor page, copy the code shown in Listing 6-17 into Notepad, and save it as a Web page using the .htm file extension.

Listing 6-17 Complete HTML Code for a Custom System Monitor

```
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Window_onLoad()
    MyMonitor.Counters.Add "\Memory\Available MBytes"
    MyMonitor.GraphTitle = "Available RAM (in MB)"
    MyMonitor.Counters(1).Width = 4
    MyMonitor.ReadOnly = True
    MyMonitor.UpdateInterval = "5"
    MyMonitor.BackColor = vbWhite
    MyMonitor.ForeColor = vbBlack
    MyMonitor.MaximumScale = 800
    MyMonitor.ShowHorizontalGrid = True
    MyMonitor.ShowVerticalGrid = True
End Sub
-->
</SCRIPT>
<OBJECT CLASSID="clsid:C4D2D8E0-D1DD-11CE-940F-008029004347" ID="MyMonitor"
WIDTH="100%" height="100%">
</OBJECT>
</BODY>
```

Even though this is a simple example, the code can easily be modified to create a more extensive and more sophisticated monitoring tool. For example, you can:

- **Modify the properties of the System Monitor control** To do this, change any of the lines listed as part of the *Window_onLoad* script. For example, to give users the right to modify the properties of the control, set the *ReadOnly* property to False, like this:

  ```
  MyMonitor.ReadOnly = False
  ```

  To configure properties not specified in our example, add those items to the *Window_onLoad* script. For example, this command sets the font for the graph to Arial:

  ```
  MyMonitor.Font = "Arial"
  ```

- **Add additional counters** To add more counters, add additional instances of the *Counters.Add* method. For example, this line can be inserted to add the \Memory\Committed Bytes counter to the control:

  ```
  MyMonitor.Counters.Add "\Memory\Committed Bytes"
  ```

Drag-and-Drop Support

Scripting the System Monitor control provides an easy way to develop and use preformatted performance reports. Once you have created one or more HTML script files to invoke the System Monitor control with a full range of settings, you can use drag-and-drop support to deploy these performance monitoring reports quickly and easily.

For example, Listing 6-18 shows a System Monitor control HTML script file that creates a Memory Allocation report. This Memory Allocation report not only includes the main performance counters that characterize virtual and physical memory allocation, it also formats them to scale on a single Chart view.

Listing 6-18 HTML Code for a System Monitor Memory Allocation Report

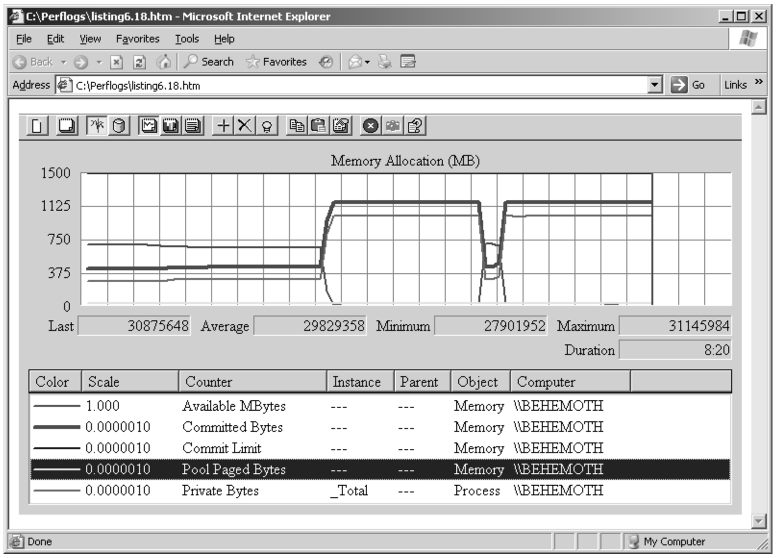
```
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Window_onLoad()
    MyMonitor.Counters.Add "\Memory\Available MBytes"
    MyMonitor.Counters.Add "\Memory\Committed Bytes"
    MyMonitor.Counters.Add "\Memory\Commit Limit"
    MyMonitor.Counters.Add "\Memory\Pool Paged Bytes"
```

```
    MyMonitor.Counters.Add "\Process(_Total)\Private Bytes"
    MyMonitor.GraphTitle = "Memory Allocation (MB)"
    MyMonitor.MaximumScale = 1500
    MyMonitor.Counters(2).ScaleFactor = -6
    MyMonitor.Counters(3).ScaleFactor = -6
    MyMonitor.Counters(4).ScaleFactor = -6
    MyMonitor.Counters(5).ScaleFactor = -6
    MyMonitor.Counters(1).Width = 2
    MyMonitor.Counters(2).Width = 2
    MyMonitor.Counters(3).Width = 4
    MyMonitor.Counters(4).Width = 2
    MyMonitor.Counters(5).Width = 2
    MyMonitor.ReadOnly = False
    MyMonitor.UpdateInterval = "5"
    MyMonitor.BackColor = vbWhite
    MyMonitor.ForeColor = vbBlack
    MyMonitor.ShowHorizontalGrid = True
    MyMonitor.ShowVerticalGrid = True

End Sub
-->
</SCRIPT>
<OBJECT CLASSID="clsid:C4D2D8E0-D1DD-11CE-940F-008029004347" ID="MyMonitor"
WIDTH="100%" height="100%">
</OBJECT>
</BODY>
```

In this example, Committed Bytes, Available Mbytes, Pool Paged Bytes, and Process (_Total)\Private Bytes are charted alongside the Commit Limit. All the memory allocation metrics are charted against a y-axis scale that is set to 1500 MB. The memory allocation counters that are reported in bytes have *ScaleFactor* settings to divide the bytes counts by 1,000,000. Note that the *ScaleFactor* settings correspond to powers of 10. The actual counter value is multiplied by this scaling factor to calculate the value to be displayed. The valid range of the *ScaleFactor* parameter corresponds to *PDH_MIN_SCALE* (–7) and *PDH_MAX_SCALE (+7)* in the *PDH.library* calls. A value of zero will set the scale to 1 so that the actual value is returned.

In this example, `MyMonitor.Counters(2).ScaleFactor = -6` sets the scale factor for Committed Bytes to 10^{-6}, which results in dividing the actual value of the Committed Bytes counter by 1,000,000 prior to displaying it.

Figure 6-23 illustrates the display that results when you drag this HTML file from Explorer to an Internet Explorer Web page.


C:\Perflogs\listing6.18.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Perflogs\listing6.18.htm
Go Links
Memory Allocation (MB)
1500
1125
750
375
0
Done
My Computer

Last	Average	Minimum	Maximum
30875648	29829358	27901952	31145984

Duration 8:20

Color	Scale	Counter	Instance	Parent	Object	Computer
—	1.000	Available MBytes	---	---	Memory	\\BEHEMOTH
—	0.0000010	Committed Bytes	---	---	Memory	\\BEHEMOTH
—	0.0000010	Commit Limit	---	---	Memory	\\BEHEMOTH
—	0.0000010	Pool Paged Bytes	---	---	Memory	\\BEHEMOTH
—	0.0000010	Private Bytes	_Total	---	Process	\\BEHEMOTH

Figure 6-23 Display of an HTML file on an Internet Explorer Web page

You can also drag your preformatted System Monitor report onto the System Monitor control hosted within the Microsoft Management Console.

Glossary

A

access control entry (ACE). An entry in an object's discretionary access control list (DACL) that grants permissions to a user or group. An ACE is also an entry in an object's system access control list (SACL) that specifies the security events to be audited for a user or group. See also access control list (ACL); discretionary access control list (DACL); object; security descriptor; system access control list (SACL).

access control list (ACL). A list of security protections that apply to an entire object, a set of the object's properties, or an individual property of an object. There are two types of access control lists: discretionary and system. See also access control entry (ACE); discretionary access control list (DACL); object; security descriptor; system access control list (SACL).

ACE. See definition for access control entry (ACE).

ACL. See definition for access control list (ACL).

Active Directory. The Windows-based directory service. Active Directory stores information about objects on a network and makes this information available to users and network administrators. Active Directory gives network users access to permitted resources anywhere on the network by using a single logon process. It provides network administrators with an intuitive, hierarchical view of the network and a single point of administration for all network objects. See also directory service; object.

Active Directory Users and Computers. An administrative tool used by an administrator to perform day-to-day Active Directory administration tasks. The tasks that can be performed with this tool include creating, deleting, modifying, moving, and setting permissions on objects stored in the directory. Examples of objects in Active Directory are organizational units, users, contacts, groups, computers, printers, and shared file objects. See also Active Directory; object.

active volume. The volume from which the computer starts up. The active volume must be a simple volume on a dynamic disk. You cannot mark an existing dynamic volume as the active volume, but you can upgrade a basic disk containing the active partition to a dynamic disk. After the disk is upgraded to dynamic, the partition becomes a simple volume that is active. See also basic disk; dynamic disk; dynamic volume; simple volume.

ActiveX. A set of technologies that allows software components to interact with one another in a networked environment, regardless of the language in which the components were created. See also ActiveX component.

ActiveX component. A reusable software component that can be used to incorporate ActiveX technology. See also ActiveX.

address (A) resource record. A resource record (RR) used to map a DNS domain name to a host Internet Protocol version 4 (IPv4) address on the network. See also IP address; resource record (RR).

ADSL. See definition for Asymmetric Digital Subscriber Line (ADSL).

affinity mask. A value that contains bits for each processor on the system, defining which processors a process or thread can use.

allocate. To mark media for use by an application. Media in the available state can be allocated.

allocated state. A state that indicates media are in use and assigned to application media pools.

allocation unit. The smallest amount of disk space that can be allocated to hold a file. All file systems used by Windows organize hard disks based on allocation units. The smaller the allocation unit size, the more efficiently a disk stores information. If you do not specify an allocation unit size when formatting the disk, Windows picks default sizes based on the size of the volume. These default sizes are selected to reduce the amount of space that is lost and the amount of fragmentation on the volume. Also called a *cluster*. See also file system; volume.

AppleTalk. See definition for AppleTalk Protocol.

AppleTalk Protocol. The set of network protocols on which AppleTalk network architecture is based. The AppleTalk Protocol is installed with Services for Macintosh to help users access resources on a network. See also resource.

application media pool. In Removable Storage, one of two classes of media pools: system and application. The application media pool is a data repository that determines which media can be accessed by which applications and that sets the policies for that media. Applications create application media pools. See also Removable Storage.

application programming interface (API). A set of routines that an application uses to request and carry out lower-level services performed by a computer's operating system. These routines usually carry out maintenance tasks such as managing files and displaying information.

Asymmetric Digital Subscriber Line (ADSL). A high-bandwidth digital transmission technology that uses existing phone lines and also allows voice transmissions over the same lines. Most of the traffic is transmitted downstream to the user, generally at rates of 512 Kbps to about 6 Mbps.

attribute. For files, information that indicates whether a file is read-only, hidden, ready for archiving (backing up), compressed, or encrypted, and whether the file contents should be indexed for fast file searching.

In Active Directory, a property of an object. For each object class, the schema defines which attributes an instance of the class must have and which additional attributes it might have. See also Active Directory; object.

authoritative. Describes a DNS server that hosts a primary or secondary copy of a DNS zone. See also DNS server; resource record (RR).

available state. A state in which media can be allocated for use by applications.

averaging counter. A type of counter that measures a value over time and displays the average of the last two measurements over some other factor (for example, PhysicalDisk\Avg. Disk Bytes/Transfer).

B

bad block A disk sector that can no longer be used for data storage, usually because of media damage or imperfections. Also known as *bad sector*.

bad sector A disk sector that can no longer be used for data storage, usually because of media damage or imperfections. Also known as *bad block*.

bar code A machine-readable label that identifies objects, such as physical media.

baseline A range of measurements derived from performance monitoring that represents acceptable performance under typical operating conditions.

basic disk A physical disk that can be accessed by MS-DOS and all Windows-based operating systems. Basic disks can contain up to four primary partitions, or three primary partitions and an extended partition with multiple logical drives. If you want to create partitions that span multiple disks, you must first convert the basic disk to a dynamic disk by using Disk Management or the Diskpart.exe command-line tool. See also dynamic disk.

basic input/output system (BIOS) On x86-based computers, the set of essential software routines that test hardware at startup, starts the operating system, and supports the transfer of data among hardware devices. The BIOS is stored in read-only memory (ROM) so that it can be executed when you turn on the computer. Although critical to performance, the BIOS is usually invisible to computer users.

bindery A database in Novell NetWare 3.*x* that contains organizational and security information about users and groups.

BIOS See definition for basic input/output system (BIOS).

BIOS parameter block (BPB) A series of fields containing data on disk size, geometry variables, and the physical parameters of the volume. The BPB is located within the boot sector.

boot partition The partition that contains the Windows operating system and its support files. The boot partition can be, but does not have to be, the same as the system partition. See also partition.

boot sector A critical disk structure for starting your computer, located at sector 1 of each volume or floppy disk. It contains executable code and data that is required by the code, including information used by the file system to access the volume. The boot sector is created when you format the volume.

bottleneck A condition, usually involving a hardware resource, that causes a computer to perform poorly.

bound trap In programming, a problem in which a set of conditions exceeds a permitted range of values, causing the microprocessor to stop what it is doing and handle the situation in a separate routine.

C

C2 level of security U.S. government security level that designates a system that has controls capable of enforcing access limitations on an individual basis. In a C2 system, the owner of a system resource has the right to decide who can access it, and the operating system can detect when data is accessed and by whom.

cache A special memory subsystem in which frequently used data values are duplicated for quick access. See also cache file.

cache file A file used by DNS servers and clients to store responses to DNS queries. For Windows DNS servers, the cache file is named Cache.dns by default. See also authoritative; cache; DNS server.

caching The process of temporarily storing recently accessed information in a special memory subsystem for quicker access. See also cache; caching resolver.

caching resolver A program that extracts information from DNS servers in response to client requests. See also cache; cache file; caching; DNS server.

capture buffer The maximum size of the capture file. When the capture file reaches the maximum size, the oldest frames are removed to make room for newer frames (FIFO queue).

change journal A feature that tracks changes to NTFS volumes, including additions, deletions, and modifications. The change journal exists on the volume as a sparse file. See also NTFS file system; volume.

changer The robotic element of an online library unit.

checkpoint In a server cluster node's registry, a snapshot of the Cluster subkey or an application subkey. The checkpoint is written to the quorum disk when certain events take place, such as a node failure.

child object An object that resides in another object. A child object implies relation. For example, a file is a child object that resides in a folder, which is the parent object. See also object; parent object.

client request A service request from a client computer to a server computer or a cluster of server computers.

Client Service for NetWare A service that allows clients to make direct connections to resources on computers running NetWare 2.*x*, 3.*x*, 4.*x*, or 5.*x* server software by using the Internetwork Packet Exchange (IPX) protocol only. This service is included with Windows XP Professional and the Microsoft Windows Server 2003 family. See also Internetwork Packet Exchange (IPX).

cluster database A database containing configuration data for all cluster objects. The cluster database is synchronized across all cluster nodes. See also node.

Cluster service The essential software component that controls all aspects of server cluster operation and manages the cluster database. Each node in a server cluster runs one instance of the Cluster service. See also node; server cluster.

COM See definition for Component Object Model (COM).

complementary metal-oxide semiconductor (CMOS) The battery-packed memory that stores information, such as disk types and amount of memory, used to start the computer.

completed state A state that indicates that media can no longer be used for write operations.

Component Object Model (COM) An object-based programming model designed to promote software interoperability; it allows two or more applications or components to easily cooperate with one another, even if they were written by different vendors, at different times, in different programming languages, or if they are running on different computers running different operating systems. OLE technology and ActiveX are both built on top of COM. See also ActiveX.

console tree The left pane in Microsoft Management Console (MMC) that displays the items contained in the console. The items in the console tree and their hierarchical organization determine the capabilities of a console. See also Microsoft Management Console (MMC).

container object An object that can logically contain other objects. For example, a folder is a container object. See also Active Directory; noncontainer object; object.

context switch An event that occurs when the kernel switches the processor from one thread to another, for example, when an I/O operation causes a thread to be blocked and the operating system selects another thread to run on the processor.

convergence The process of stabilizing a system after changes occur in the network. For dynamic routing, if a route becomes unavailable, routers send update messages throughout the network, reestablishing information about preferred routes.

For Network Load Balancing, a process by which hosts exchange messages to determine a new, consistent state of the cluster and to elect the default host. During convergence, a new load distribution is determined for hosts that share the handling of network traffic for specific Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) ports. See also Network Load Balancing.

cyclic redundancy check (CRC) A procedure used in checking for errors in data transmission. CRC error checking uses a complex calculation to generate a number based on the data transmitted. The sending device performs the calculation before transmission and sends its result to the receiving device. The receiving device repeats the same calculation after transmission. If both devices obtain the same result, it is assumed that the transmission was error-free. The procedure is known as a redundancy check because each transmission includes not only data but extra (redundant) error-checking values. Communications protocols such as XMODEM and Kermit use cyclical redundancy checking.

D

deallocate To return media to the available state after they have been used by an application.

decommissioned state A state indicating that media have reached their allocation maximum.

deferred procedure call (DPC) A kernel-defined control object type that represents a procedure that is to be called later. A DPC runs at DISPATCH_LEVEL IRQL. A DPC can be used when a timer event occurs or when an ISR needs to perform more work but should do so at a lower interrupt request level than the one at which an ISR executes. In an SMP environment, a DPC might run immediately on a processor other than the current one, or might run after another interrupt has run on the current processor.

desired free space An amount of free space specified in Remote Storage that should be maintained on a volume at all times during normal use. See also Remote Storage; volume.

device fonts See definition for printer fonts.

DFS See definition for Distributed File System (DFS).

DHCP See definition for Dynamic Host Configuration Protocol (DHCP).

differential data Saved copies of changed data that can be applied to an original volume to generate a volume shadow copy. See also volume; volume shadow copy.

digital video disc (DVD) A type of optical disc storage technology. A digital video disc (DVD) looks like a CD-ROM disc, but it can store greater amounts of data. DVDs are often used to store full-length movies and other multimedia content that requires large amounts of storage space. See also DVD decoder; DVD drive.

direct memory access (DMA) Memory access that does not involve the microprocessor. DMA is frequently used for data transfer directly between memory and a peripheral device such as a disk drive.

directory An information source that contains information about users, computer files, or other objects. In a file system, a directory stores information about files. In a distributed computing environment (such as a Windows domain), the directory stores information about objects such as printers, fax servers, applications, databases, and other users. See also object.

directory service Both the directory information source and the service that makes the information available and usable. A directory service enables the user to find an object when given any one of its attributes. See also Active Directory; attribute; directory; object.

disconnected placeholder A placeholder whose file contents have been removed from Remote Storage. A disconnected placeholder could have been restored from backup after the space in Remote Storage was reclaimed, or the data within Remote Storage is physically unavailable (for example, because of a media failure). See also Remote Storage.

discretionary access control list (DACL) The part of an object's security descriptor that grants or denies specific users and groups permission to access the object. Only the owner of an object can change permissions granted or denied in a DACL; thus, access to the object is at the owner's discretion. See also access control entry (ACE); object; security descriptor; system access control list (SACL).

disk bottleneck A condition that occurs when disk performance is reduced to the extent that overall system performance is affected.

disk quota The maximum amount of disk space available to a user.

Distributed File System (DFS) A service that allows system administrators to organize distributed network shares into a logical namespace, enabling users to access files without specifying their physical location and providing load sharing across network shares.

DNS server A server that maintains information about a portion of the DNS database, and that responds to and resolves DNS queries.

domain controller In an Active Directory forest, a server that contains a writable copy of the Active Directory database, participates in Active Directory replication, and controls access to network resources. Administrators can manage user accounts, network access, shared resources, site topology, and other directory objects from any domain controller in the forest. See also Active Directory; directory.

domain local group A security or distribution group that can contain universal groups, global groups, other domain local groups from its own domain, and accounts from any domain in the forest. Domain local security groups can be granted rights and permissions on resources that reside only in the same domain where the domain local group is located. See also global group.

dots per inch (DPI) The standard used to measure screen and printer resolution, expressed as the number of dots that a device can display or print per linear inch. The greater the number of dots per inch, the better the resolution.

downloadable fonts A set of characters stored on disk and sent (downloaded) to a printer's memory when needed for printing a document. Downloadable fonts are most commonly used with laser printers and other page printers, although many dot-matrix printers can accept some of them. Also called *soft fonts*. See also font; font cartridge.

DVD decoder A hardware or software component that allows a digital video disc (DVD) drive to display movies on your computer screen. See also digital video disc (DVD); DVD drive; hardware decoder.

DVD drive A disk storage device that uses digital video disc (DVD) technology. A DVD drive reads both CD-ROM and DVDs; however, you must have a DVD decoder to display DVD movies on your computer screen. See also digital video disc (DVD); DVD decoder.

dynamic disk A physical disk that provides features that basic disks do not, such as support for volumes that span multiple disks. Dynamic disks use a hidden database to track information about dynamic volumes on the disk and other dynamic disks in the computer. You convert basic disks to dynamic by using the Disk Management snap-in or the DiskPart command-line tool. When you convert a basic disk to dynamic, all existing basic volumes become dynamic volumes. See also active volume; basic disk; dynamic volume; partition; volume.

Dynamic Host Configuration Protocol (DHCP) A TCP/IP service protocol that offers dynamic leased configuration of host IP addresses and distributes other configuration parameters to eligible network clients. DHCP provides safe, reliable, and simple TCP/IP network configuration, prevents address conflicts, and helps conserve the use of client IP addresses on the network.

DHCP uses a client/server model where the DHCP server maintains centralized management of IP addresses that are used on the network. DHCP-supporting clients can then request and obtain lease of an IP address from a DHCP server as part of their network boot process.

See also IP address; Transmission Control Protocol/Internet Protocol (TCP/IP).

dynamic priority The priority value to which a thread's base priority is adjusted to optimize scheduling.

dynamic volume A volume that resides on a dynamic disk. Windows supports five types of dynamic volumes: simple, spanned, striped, mirrored, and RAID-5. A dynamic volume is formatted by using a file system, such as file allocation table (FAT) or NTFS, and has a drive letter assigned to it. See also basic disk; dynamic disk; mirrored volume; RAID-5 volume; simple volume; spanned volume; striped volume; volume.

dynamic-link library (DLL) An operating system feature that allows executable routines (generally serving a specific function or set of functions) to be stored separately as files with .dll extensions. These routines are loaded only when needed by the program that calls them.

E

EFS See definition for Encrypting File System (EFS).

EIDE See definition for enhanced integrated device electronics (EIDE).

(EISA) See definition for Extended Industry Standard Architecture (EISA).

Encrypting File System (EFS) A feature in this version of Windows that enables users to encrypt files and folders on an NTFS volume disk to keep them safe from access by intruders. See also NTFS file system.

enhanced integrated device electronics (EIDE) An extension of the IDE standard, EIDE is a hardware interface standard for disk drive designs that houses control circuits in the drives themselves. It allows for standardized interfaces to the system bus, while providing for advanced features, such as burst data transfers and direct data access.

event logging The process of recording an audit entry in the audit trail whenever certain events occur, such as services starting and stopping, or users logging on and off and accessing resources.

expire interval For DNS, the number of seconds that DNS servers operating as secondary masters for a zone will use to determine whether zone data should be expired when the zone is not refreshed and renewed. See also DNS server.

Extended Industry Standard Architecture (EISA) A 32-bit bus standard introduced in 1988 by a consortium of nine computer-industry companies. EISA maintains compatibility with the earlier Industry Standard Architecture (ISA) but provides additional features.

F

failback The process of moving resources, either individually or in a group, back to their preferred node after the node has failed and come back online. See also failback policy; node; resource.

failback policy Parameters that an administrator can set using Cluster Administrator that affect failback operations. See also failback.

failover In server clusters, the process of taking resource groups offline on one node and bringing them online on another node. When failover occurs, all resources within a resource group fail over in a predefined order; resources that depend on other resources are taken offline before, and are brought back online after, the resources on which they depend. See also failover policy; node; possible owner; server cluster.

failover policy Parameters that an administrator can set using Cluster Administrator that affect failover operations. See also failover.

fault tolerance The ability of computer hardware or software to ensure data integrity when hardware failures occur. Fault-tolerant features appear in many server operating systems and include mirrored volumes, RAID-5 volumes, and server clusters. See also mirrored volume; RAID-5 volume.

FIFO First in, first out.

file allocation table (FAT) A file system used by MS-DOS and other Windows operating systems to organize and manage files. The file allocation table is a data structure that Windows creates when you format a volume by using FAT or FAT32 file systems. Windows stores information about each file in the file allocation table so that it can retrieve the file later. See also file system; NTFS file system.

File Share resource A file share accessible by a network path that is supported as a cluster resource by a Resource DLL.

file system In an operating system, the overall structure in which files are named, stored, and organized. NTFS, FAT, and FAT32 are types of file systems. See also NTFS file system.

File Transfer Protocol (FTP) A member of the TCP/IP suite of protocols, used to copy files between two computers on the Internet. Both computers must support their respective FTP roles: one must be an FTP client and the other an FTP server. See also Transmission Control Protocol/Internet Protocol (TCP/IP).

font A graphic design applied to a collection of numbers, symbols, and characters. A font describes a certain typeface, along with other qualities such as size, spacing, and pitch.

font cartridge A plug-in unit available for some printers that contains fonts in several styles and sizes. As with downloadable fonts, printers using font cartridges can produce characters in sizes and styles other than those created by the fonts built into it. See also downloadable fonts; font.

foreground boost A mechanism that increases the priority of a foreground application.

FTP See definition for File Transfer Protocol (FTP).

G

global group A security or distribution group that can contain users, groups, and computers from its own domain as members. Global security groups can be granted rights and permissions for resources in any domain in the forest. See also group; local group.

globally unique identifier (GUID) A 16-byte value generated from the unique identifier on a device, the current date and time, and a sequence number. A GUID is used to identify a particular device or component.

graphical user interface (GUI) A display format, like that of Windows, that represents a program's functions with graphic images such as buttons and icons. GUIs enable a user to perform operations and make choices by pointing and clicking with a mouse.

group A collection of users, computers, contacts, and other groups. Groups can be used as security or as e-mail distribution collections. Distribution groups are used only for e-mail. Security groups are used both to grant access to resources and as e-mail distribution lists. See also global group; local group.

Group Policy The infrastructure within Active Directory directory service that enables directory-based change and configuration management of user and computer settings, including security and user data. You use Group Policy to define configurations for groups of users and computers. With Group Policy, you can specify policy settings for registry-based policies, security, software installation, scripts, folder redirection, remote installation services, and Internet Explorer maintenance. The Group Policy settings that you create are contained in a Group Policy object (GPO). By associating a GPO with selected Active Directory system containers—sites, domains, and organizational units—you can apply the GPO's policy settings to the users and computers in those Active Directory containers. To create an individual GPO, use the Group Policy Object Editor. To manage Group Policy objects across an enterprise, you can use the Group Policy Management console. See also Active Directory.

H

HAL See definition for hardware abstraction layer (HAL).

hard affinity A mechanism by which a thread can run only on a set of processors.

hardware abstraction layer (HAL) A thin layer of software provided by the hardware manufacturer that hides, or abstracts, hardware differences from higher layers of the operating system. By means of the filter provided by the HAL, different types of hardware look alike to the rest of the operating system. This enables the operating system to be portable from one hardware platform to another. The HAL also provides routines that enable a single device driver to support the same device on all platforms.

Hardware Compatibility List (HCL) A hardware list that Microsoft compiled for specific products, including Microsoft Windows 2000 and earlier versions of Windows. The list for a specific product, such as Windows 2000, includes the hardware devices and computer systems that are compatible with that version of the product. For products in the Windows Server 2003 family, you can find the equivalent information on the Windows Catalog Web site.

hardware decoder A type of digital video disc (DVD) decoder that allows a DVD drive to display movies on your computer screen. A hardware decoder uses both software and hardware to display movies. See also DVD decoder; DVD drive.

hardware malfunction message A character-based, full-screen error message displayed on a blue background. It indicates that the microprocessor detected a hardware error condition from which the system cannot recover.

HCL See definition for Hardware Compatibility List (HCL).

heartbeat A message that is sent at regular intervals by one computer on a Network Load Balancing cluster or server cluster to another computer within the cluster to detect communication failures. See also Network Load Balancing; server cluster.

heartbeat thread A thread initiated by the Microsoft Windows NT Virtual DOS Machine (NTVDM) process that interrupts every 55 milliseconds to simulate a timer interrupt.

high performance file system (HPFS) The file system designed for the OS/2 version 1.2 operating system.

HTML See definition for Hypertext Markup Language (HTML).

HTTP See definition for Hypertext Transfer Protocol (HTTP).

Hypertext Markup Language (HTML) A simple markup language used to create hypertext documents that are portable from one platform to another. HTML files are simple ASCII text files with codes embedded (indicated by markup tags) to denote formatting and hypertext links.

Hypertext Transfer Protocol (HTTP) The protocol used to transfer information on the World Wide Web. An HTTP address, which is one kind of Uniform

Resource Locator (URL), takes the following form: *http://www.microsoft.com*. See also Uniform Resource Locator (URL).

I

ideal processor A processor associated with a thread containing a default value assigned by the system, or specified by the program developer in the application code. The scheduler favors running a thread on the ideal processor that is assigned to the thread as part of the soft affinity algorithm.

IIS Server Instance resource A server-instance designation used with Microsoft Internet Information Services (IIS) that supports the WWW and FTP services. IIS server instances are supported as cluster resources by a Resource DLL. IIS Server Instance resources can have dependencies on IP Address resources, Network Name resources, and Physical Disk resources. Access information for server instances does not fail over. See also failover; Internet Information Services (IIS).

import media pool A logical collection of data-storage media that has not been cataloged by Removable Storage. Media in an import media pool is cataloged as soon as possible so that they can be used by an application. See also Removable Storage.

imported state A state that indicates media whose label types are recognized by Removable Storage, but whose label IDs are not cataloged by Removable Storage. See also media states; Removable Storage.

inaccessible state A state that indicates that a side of a multicartridge drive is in a drive, but is not in the accessible state. See also media states.

incompatible state A state that indicates that media are not compatible with the library in which they were classified. This media should be immediately ejected from the library. See also library; media states.

independent software vendors (ISVs) A third-party software developer; an individual or an organization that independently creates computer software.

initialize In Disk Management, the process of detecting a disk or volume and assigning it a status (for example, healthy) and a type (for example, dynamic). See also basic disk; dynamic disk; dynamic volume.

input/output (I/O) port A channel through which data is transferred between a device and the microprocessor. The port appears to the microprocessor as one or more memory addresses that it can use to send or receive data. See also memory address; port.

instantaneous counter A type of counter that displays the most recent measurement taken by the Performance console.

Internet Information Services (IIS) Software services that support Web site creation, configuration, and management, along with other Internet functions. Internet Information Services includes Network News Transfer Protocol (NNTP), File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP). See also File Transfer Protocol (FTP); Network News Transfer Protocol (NNTP); Simple Mail Transfer Protocol (SMTP).

Internetwork Packet Exchange (IPX) A network protocol native to NetWare that controls addressing and routing of packets within and between local area networks (LANs). IPX does not guarantee that a message will be complete (no lost packets). See also Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX); local area network (LAN).

Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX) Transport protocols used in Novell NetWare networks, which together correspond to the combination of TCP and IP in the TCP/IP protocol suite. Windows implements IPX through NWLink. See also Internetwork Packet Exchange (IPX); NWLink IPX/SPX/NetBIOS Compatible Transport Protocol (NWLink); Transmission Control Protocol/Internet Protocol (TCP/IP).

interprocess interrupt A high IRQ level interrupt that can send an interrupt from one processor to another, allowing processors to communicate. See also interrupt request (IRQ).

interrupt avoidance A feature of device adapters that allows a processor to continue processing interrupts without new interrupts being queued until all pending interrupts are complete.

interrupt moderation A feature of device adapters that allows a processor to process interrupts more efficiently by grouping several interrupts to a single hardware interrupt.

interrupt request (IRQ) A signal sent by a device to get the attention of the processor when the device is ready to accept or send information. Each device sends its interrupt requests over a specific hardware line. Each device must be assigned a unique IRQ number.

IP address For Internet Protocol version 4 (IPv4), a 32-bit address used to identify an interface on a node on an IPv4 internetwork. Each interface on the IP internetwork must be assigned a unique IPv4 address, which is made up of the network ID, plus a unique host ID. This address is typically represented with the decimal value of each octet separated by a period (for example, 192.168.7.27). You can configure the IP address statically or dynamically by using Dynamic Host Configuration Protocol (DHCP).

For Internet Protocol version 6 (IPv6), an identifier that is assigned at the IPv6 layer to an interface or set of interfaces and that can be used as the source or destination of IPv6 packets.

See also Dynamic Host Configuration Protocol (DHCP); node.

IPX/SPX See definition for Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX).

J

job object A system-level structure that allows processes to be grouped together and managed as a single unit.

jukebox See definition for library.

K

kernel The core of layered architecture that manages the most basic operations of the operating system and the computer's processor. The kernel schedules for the processor different blocks of executing code, called threads, to keep the processor as busy as possible, and coordinates multiple processors to optimize performance. The kernel also synchronizes activities among Executive-level subcomponents, such as I/O Manager and Process Manager, and handles hardware exceptions and other hardware-dependent functions. The kernel works closely with the hardware abstraction layer.

L

LAN manager replication The default file replication service in Windows NT.

library A data-storage system, usually managed by Removable Storage. A library consists of removable media (such as tapes or discs) and a hardware device that can read from or write to the media. There are two major types of libraries: robotic libraries (automated multiple-media, multidrive devices) and stand-alone drive libraries (manually operated, single-drive devices). A robotic library is also called a *jukebox* or *changer*. See also Removable Storage.

library request A request for a library or stand-alone drive to perform a task. This request can be issued by an application or by Removable Storage. See also library; Removable Storage.

Line Printer Daemon (LPD) A service on a print server that receives print jobs from Line Printer Remote (LPR) tools that are running on client computers. See also Line Printer Remote (LPR); print job; print server.

Line Printer Remote (LPR) A connectivity tool that runs on client computers and that is used to print files to a computer running a Line Printer Daemon (LPD) server. See also Line Printer Daemon (LPD).

load balancing A technique used by Windows Clustering to scale the performance of a server-based program (such as a Web server) by distributing its client requests across multiple servers within the cluster. Each host can specify the load percentage that it will handle, or the load can be equally distributed across all the hosts. If a host fails, Windows Clustering dynamically redistributes the load among the remaining hosts.

local area network (LAN) A communications network connecting a group of computers, printers, and other devices located within a relatively limited area (for example, a building). A LAN enables any connected device to interact with any other on the network. See also NetBIOS Extended User Interface (NetBEUI); virtual local area network (VLAN).

local group A security group that can be granted rights and permissions only on resources that are on the computer the group was created on. Local groups can have any user accounts that are local to the computer as members, as well as users, groups, and computers from a domain to which the computer belongs. See also global group.

local printer A printer that is directly connected to one of the ports on a computer. See also port.

local storage For the Windows Server 2003 family, NTFS file system disk volumes used as primary data storage. Such disk volumes can be managed by Remote

Storage by copying infrequently accessed files to remote (secondary) storage. See also NTFS file system; Remote Storage; volume.

LocalTalk The Apple networking hardware built into every Macintosh computer. LocalTalk includes the cables and connector boxes that connect components and network devices that are part of the AppleTalk network system. Formerly known as *AppleTalk Personal Network*.

logical printer The software interface between the operating system and the printer in Windows. Whereas a printer is the device that does the actual printing, a logical printer is its software interface on the print server. This software interface determines how a print job is processed and how it is routed to its destination (to a local or network port, to a file, or to a remote print share). When you print a document, it is spooled (or stored) on the logical printer before it is sent to the printer itself. See also spooling.

M

master boot record (MBR) The first sector on a hard disk, which begins the process of starting the computer. The MBR contains the partition table for the disk and a small amount of executable code called the *master boot code*.

master file table (MFT) An NTFS system file on NTFS-formatted volumes that contains information about each file and folder on the volume. The MFT is the first file on an NTFS volume. See also file allocation table (FAT); NTFS file system; volume.

MBR See definition for master boot record (MBR).

media access control (MAC) A sublayer of the IEEE 802 specifications that defines network access methods and framing.

media label library A dynamic-link library (DLL) that can interpret the format of a media label written by a Removable Storage application. See also dynamic-link library (DLL); Removable Storage.

media states A status designation for media managed by Removable Storage. Media states include Idle, In Use, Mounted, Loaded, and Unloaded. See also Removable Storage.

memory address A portion of computer memory that can be allocated to a device or used by a program or the operating system. Devices are usually allocated a range of memory addresses.

memory leak A condition that occurs when an application allocates memory for use but does not free allocated memory when finished.

metadata Data about data. For example, the title, subject, author, and size of a file constitute the file's metadata.

Microsoft Management Console (MMC) A framework for hosting administrative tools called *snap-ins*. A console might contain tools, folders, or other containers; World Wide Web pages; and other administrative items. These items are displayed in the left pane of the console, called a *console tree*. A console has one or more windows that can provide views of the console tree. The main MMC window provides commands and tools for authoring consoles. The authoring features of MMC and the console tree itself might be hidden when a console is in User mode. See also console tree; snap-in.

migrate In file management, to move files or programs from an older file format or protocol to a more current format or protocol. For example, WINS database entries can be migrated from static WINS database entries to dynamically registered DHCP entries.

In Active Directory, to move Active Directory accounts, resources, and their associated security objects from one domain to another.

In Windows NT, to change the domain controller operating system from Windows NT to an operating system with Active Directory, such as Windows 2000 or Windows Server 2003. A migration from Windows NT can include in-place domain upgrades, domain restructuring, or both.

In Remote Storage, to copy an object from local storage to remote storage.

See also Active Directory; Dynamic Host Configuration Protocol (DHCP); Remote Storage.

migration See definition for migrate.

minimum TTL In DNS, a default Time to Live (TTL) value that is set in seconds and used with all resource records in a zone. This value is set in the start of authority (SOA) resource record for each zone. By default, the DNS server includes this value in query responses. It is used to inform recipients about how long they can store and use resource records, which are provided in the query answer, before they must expire the stored records data. When TTL values are set for individual resource records, those values override the minimum TTL. See also DNS server; resource record (RR); Time to Live (TTL).

mirror One of the two volumes that make up a mirrored volume. Each mirror of a mirrored volume resides on a different disk. If one mirror becomes unavailable (because of a disk failure, for example), Windows can use the remaining mirror to gain access to the volume's data. See also fault tolerance; mirrored volume; volume.

mirror set A fault-tolerant partition created with Windows NT 4.0 or earlier that duplicates data on two physical disks. Microsoft Windows XP and the Windows Server 2003 family do not support mirror sets. In the Windows Server 2003 family, you must create mirrored volumes on dynamic disks. See also dynamic disk; mirrored volume.

mirrored volume A fault-tolerant volume that duplicates data on two physical disks. A mirrored volume provides data redundancy by using two identical volumes, which are called *mirrors*, to duplicate the information contained on the volume. A mirror is always located on a different disk. If one of the physical disks fails, the data on the failed disk becomes unavailable, but the system continues to operate in the mirror on the remaining disk. You can create mirrored volumes only on dynamic disks on computers running the Windows 2000 Server or Windows Server 2003 families of operating systems. You cannot extend mirrored volumes. See also dynamic disk; dynamic volume; fault tolerance; RAID-5 volume; volume.

MMC See definition for Microsoft Management Console (MMC).

mounted drive A drive attached to an empty folder on an NTFS volume. Mounted drives function the same as any other drive, but are assigned a label or name instead of a drive letter. The mounted drive's name is resolved to a full file system path instead of just a drive letter. Members of the Administrators group can use Disk Management to create mounted drives or reassign drive letters. See also NTFS file system; volume.

N

name resolution service A service, such as that provided by WINS or DNS, that allows friendly names to be resolved to an address, or to other specially defined resource data used to locate network resources of various types and purposes.

NetBEUI See definition for NetBIOS Extended User Interface (NetBEUI).

NetBIOS Extended User Interface (NetBEUI) A network protocol native to Microsoft Networking. It is usually used in small, department-size local area networks (LANs) of 1 to 200 clients. NetBEUI can use Token Ring source routing as its only method of routing. NetBEUI is the Microsoft implementation of the NetBIOS standard. See also local area network (LAN); Token Ring.

NetBIOS over TCP/IP (NetBT) A feature that provides the NetBIOS programming interface over the TCP/IP protocol. It is used for monitoring routed servers that use NetBIOS name resolution.

NetWare Core Protocol The file-sharing protocol that governs communications about resources (such as the disk and printer), bindery, and Novell Directory Services (NDS) operations between server and client computers on a Novell NetWare network. See also bindery; Internetwork Packet Exchange (IPX).

network administrator A person responsible for planning, configuring, and managing the day-to-day operation of the network. Also called a *system administrator*.

network data stream The total amount of data transferred over a network at any given time.

Network Load Balancing A Windows network component that uses a distributed algorithm to load-balance Internet Protocol (IP) traffic across a number of hosts, helping to enhance the scalability and availability of mission-critical, IP-based services, such as Terminal Services, Web services, virtual private networking, and streaming media. It also provides high availability by detecting host failures and automatically redistributing traffic to the surviving hosts.

Network News Transfer Protocol (NNTP) A member of the TCP/IP suite of protocols used to distribute network news messages to NNTP servers and clients (newsreaders) on the Internet. NNTP is designed so that news articles are stored on a server in a central database, thus enabling a user to select specific items to read. See also Transmission Control Protocol/Internet Protocol (TCP/IP).

NNTP See definition for Network News Transfer Protocol (NNTP).

node For tree structures, a location on the tree that can have links to one or more items below it. For local area networks (LANs), a device that is connected to the network and is capable of communicating with other network devices. For server clusters, a computer system that is an active or inactive member of a cluster. See also local area network (LAN); server cluster.

noncontainer object An object that cannot logically contain other objects. For example, a file is a noncontainer object. See also container object; object.

NTFS file system An advanced file system that provides performance, security, reliability, and advanced features that are not found in any version of file allocation table (FAT). For example, NTFS guarantees volume consistency by using standard transaction logging and recovery techniques. If a system fails, NTFS uses its log file and checkpoint information to restore the consistency of the file system. NTFS also provides advanced features, such as file and folder permissions, encryption, disk quotas, and compression. See also file allocation table (FAT); file system.

NWLink See definition for NWLink IPX/SPX/NetBIOS Compatible Transport Protocol (NWLink).

NWLink IPX/SPX/NetBIOS Compatible Transport Protocol (NWLink) The Microsoft implementation of the Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX) protocol used on NetWare networks. NWLink allows connectivity between Windows-based computers and NetWare networks running IPX/SPX. NWLink also provides network basic input/output system (NetBIOS) functionality and the Routing Information Protocol (RIP). See also Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX).

O

object An entity, such as a file, a folder, a shared folder, a printer, or an Active Directory object, that is described by a distinct, named set of attributes. For example, the attributes of a File object include its name, location, and size; the attributes of an Active Directory User object might include the user's first name, last name, and e-mail address.

For OLE and ActiveX, an object can also be any piece of information that can be linked to, or embedded into, another object.

See also Active Directory; attribute; child object; parent object.

object linking and embedding (OLE) A method for sharing information among applications. Linking an object, such as a graphic, from one document to another inserts a reference to the object into the second document. Any changes you make in the object in the first document will also be made in the second document. Embedding an object inserts a copy of an object from one document into another document. Changes you make in the object in the first document will not be updated in the second unless the embedded object is explicitly updated.

offline media Media, such as a tape or optical disc, that are not currently accessible by a computer and that must be inserted into a drive to be accessed.

offset When defining a pattern match within a filter using Network Monitor, the number of bytes from the beginning of the frame where the pattern occurs in a frame.

on-media identifier (OMID) A label that is electronically recorded on each medium in a Removable Storage system. Removable Storage uses on-media identifiers to track media in the Removable Storage database. See also Removable Storage.

online library See definition for library.

operator request A message that asks a user to perform a specific task. Operator requests can be issued by Removable Storage or by a program that is aware of Removable Storage, such as Backup. See also Removable Storage.

option types Client configuration parameters that a DHCP server can assign when offering an IP address lease to a client. Typically, these option types are enabled and configured for each scope. Most options are predefined through RFC 2132, but DHCP Manager can be used to define and add custom option types if needed.

original equipment manufacturer (OEM) A company that typically purchases computer components from other manufacturers, uses the components to build a personal computer, preinstalls Windows onto that computer, and then sells the computer to the public.

orphan A member of a mirrored volume or a RAID-5 volume that has failed because of a severe cause, such as a loss of power or a complete hard-disk head failure. When this happens, the fault-tolerant driver determines that it can no longer use the orphaned member and directs all new reads and writes to the remaining members of the fault-tolerant volume. See also fault tolerance; mirrored volume; RAID-5 volume.

overclocking Setting a microprocessor to run at speeds above the rated specification.

P

page-description language (PDL) A computer language that describes the arrangement of text and graphics on a printed page. See also PostScript; Printer Control Language (PCL); Printer Job Language (PJL).

paper source The location (such as Upper Paper Tray or Envelope Feeder) of the paper at the printer.

parent object An object in which another object resides. For example, a folder is a parent object in which a file, or child object, resides. An object can be both a parent and a child object. For example, a subfolder that contains files is both the child of the parent folder and the parent folder of the files. See also child object; object.

parity A calculated value that is used to reconstruct data after a failure. RAID-5 volumes stripe data and parity intermittently across a set of disks. When a disk fails, some server operating systems use the parity information together with the data on good disks to recreate the data on the failed disk. See also fault tolerance; RAID-5 volume; striped volume.

parity bit In asynchronous communications, an extra bit used to check for errors in groups of data bits transferred within or between computer systems. In modem-to-modem communications, a parity bit is often used to check the accuracy with which each character is transmitted. See also parity.

partition A portion of a physical disk that functions as though it were a physically separate disk. After you create a partition, you must format it and assign it a drive letter before you can store data on it.

On basic disks, partitions are known as *basic volumes*, which include primary partitions and logical drives. On dynamic disks, partitions are known as *dynamic volumes*, which include simple, striped, spanned, mirrored, and RAID-5 volumes.

See also basic disk; dynamic volume.

pattern match In Network Monitor, specific pattern of ASCII or hexadecimal data. A pattern match can be used in setting a filter or capture trigger.

paused A state that applies to a node in a cluster. The node is a fully active member in the cluster but cannot accept new resource groups. (For example, a resource group cannot fail over or fail back to a paused node.) You can administer and maintain a paused node. See also failback; failover; node.

performance counter In System Monitor, a data item that is associated with a performance object. For each counter selected, System Monitor presents a value corresponding to a particular aspect of the performance that is defined for the performance object. See also performance object.

Performance Monitor A Windows NT administrative tool that monitors performance on local or remote computers. Performance Monitor was replaced by the Performance console in Windows 2000.

performance object In System Monitor, a logical collection of counters that is associated with a resource or service that can be monitored. See also performance counter.

peripheral component interconnect (PCI) A specification introduced by Intel Corporation that defines a local bus system that allows up to 10 PCI-compliant expansion cards to be installed in the computer.

physical media A storage object that data can be written to, such as a disk or magnetic tape. A physical medium is referenced by its physical media ID (PMID).

placeholder A Remote Storage identifier for an NTFS volume.

Plug and Play A set of specifications developed by Intel Corporation that enables a computer to detect and configure a device automatically and install the appropriate device drivers. See also universal serial bus (USB).

port A connection point on your computer where you can connect devices that pass data into and out of a computer. For example, a printer is typically connected to a parallel port (also called an

LPT port), and a modem is typically connected to a serial port (also called a COM port). See also universal serial bus (USB).

port monitor A device that controls the computer port that provides connectivity to a local or remote print device.

Portable Operating System Interface for UNIX (POSIX) An Institute of Electrical and Electronics Engineers (IEEE) standard that defines a set of operating-system services. Programs that adhere to the POSIX standard can be easily ported from one system to another. POSIX was based on UNIX system services, but it was created in a way that allows it to be implemented by other operating systems.

POSIX See definition for Portable Operating System Interface for UNIX (POSIX).

possible owner A node in a cluster that can run a specific resource. By default, all nodes appear as possible owners, so the resource can run on any node. In most cases, it is appropriate to use this default setting. If you want the resource to be able to fail over, at least two nodes must be designated as possible owners. See also failover; node; resource.

POST See definition for power-on self test (POST).

PostScript A page-description language (PDL), developed by Adobe Systems, for printing on laser printers. PostScript offers flexible font capability and high-quality graphics. It is the standard for desktop publishing because it is supported by imagesetters, the high-resolution printers used by printing services for commercial typesetting. See also page-description language (PDL); PostScript printer; Printer Control Language (PCL); Printer Job Language (PJL).

PostScript printer A printer that uses the PostScript page-description language (PDL) to create text and graphics on the output medium, such as paper or overhead transparency. Examples of PostScript printers include the Apple LaserWriter, the NEC LC-890, and the QMS PS-810. See also page-description language (PDL); PostScript; virtual printer memory.

power-on self test (POST) A set of routines stored in read-only memory (ROM) that tests various system components such as RAM, the disk drives, and the keyboard, to see whether they are properly connected and operating. If problems are found, these routines alert the user with a series of beeps or a message, often accompanied by a diagnostic numeric value. If the POST is successful, it passes control to the bootstrap loader.

premigrated file A file that has been copied to Remote Storage in preparation for truncation but remains on the managed volume. When it is truncated, it becomes a placeholder for the file. See also Remote Storage.

print device A hardware device used for printing, commonly called a *printer*.

print job The source code that contains both the data to be printed and the commands for print. Print jobs are classified into data types based on what modifications, if any, the spooler must make to the job for it to print correctly. See also print spooler.

print processor The component that, working in conjunction with the printer driver, receives and alters print jobs, as necessary, according to their data type to ensure that the jobs print correctly. See also print job; printer driver.

print server A computer that is dedicated to managing the printers on a network. The print server can be any computer on the network.

Print Server for Macintosh A service that enables Macintosh clients to send and spool documents to printers attached to a computer running Windows NT Server; Windows 2000 Server; or an operating system in the Windows Server 2003 family, excluding 64-bit editions; and that enables clients to send documents to printers anywhere on an AppleTalk network. Also known as *MacPrint*.

print server service A service that receives print jobs from remote print clients. Different services are provided for different clients.

Print Services for UNIX A print server service for UNIX clients.

print spooler Software that accepts a document sent to a printer and then stores it on disk or in memory until the printer is ready for it. See also spooling.

Printer Control Language (PCL) The page-description language (PDL) developed by Hewlett-Packard for their laser and inkjet printers. Because of the widespread use of laser printers, this command language has become a standard in many printers. See also page-description language (PDL); PostScript; Printer Job Language (PJL).

printer driver A program designed to allow other programs to work with a particular printer without concerning themselves with the specifics of the printer's hardware and internal language. By using printer drivers that handle the subtleties of each printer, programs can communicate properly with a variety of printers.

printer fonts Fonts residing in or intended for a printer. A printer font, usually located in the printer's read-only memory (ROM), can be internal, downloaded, or on a font cartridge. See also device fonts; downloadable fonts; font; font cartridge.

Printer Job Language (PJL) The printer command language developed by Hewlett Packard that provides printer control at the print-job level. Using PJL commands, you can change default printer settings such as number of copies to print. PJL commands also permit switching printer languages between print jobs without action by the user. If bi-directional communication is supported, a PJL-compatible printer can send information such as printer model and job status to the print server. See also page-description language (PDL); PostScript; Printer Control Language (PCL).

printer permissions Permissions that specify the type of access that a user or group has to a printer. The printer permissions are Print, Manage Printers, and Manage Documents.

printers folder The folder in Control Panel that contains the Add Printer Wizard and icons for all the printers installed on your computer.

priority inversion The mechanism that allows low-priority threads to run and complete execution rather than being preempted and locking up a resource such as an I/O device.

pruning A process that removes unavailable printers from Active Directory. A program running on the domain controller periodically checks for orphaned printers (printers that are offline or powered down) and deletes the printer objects of the printers it cannot find. See also Active Directory; domain controller.

pull partner A WINS component that requests replication of updated WINS database entries from its push partner. See also push partner.

push partner A WINS component that notifies its pull partner when updated WINS database entries are available for replication. See also pull partner.

Q

quantum Also known as a *time slice*, the maximum amount of time a thread can run before the system checks for another ready thread of the same priority to run.

queue A list of programs or tasks waiting for execution. In Windows printing terminology, a queue refers to a group of documents waiting to be printed. In NetWare and OS/2 environments, queues are the primary software interface between the application and print device; users submit documents to a queue. With Windows, however, the printer is that interface; the document is sent to a printer, not a queue. See also transactional message.

R

RAID See definition for Redundant Array of Independent Disks (RAID).

RAID-5 volume A fault-tolerant volume with data and parity striped intermittently across three or more physical disks. Parity is a calculated value that is used to reconstruct data after a failure. If a portion of a physical disk fails, Windows recreates the data that was on the failed portion from the remaining data and parity. You can create RAID-5 volumes only on dynamic disks on computers running the Windows 2000 Server or Windows Server 2003 families of operating systems. You cannot mirror or extend RAID-5 volumes. In Windows NT 4.0, a RAID-5 volume was known as a *striped set with parity*. See also dynamic disk; dynamic volume; fault tolerance; parity; volume.

RAM See definition for random access memory (RAM).

random access memory (RAM) Memory that can be read from or written to by a computer or other devices. Information stored in RAM is lost when the computer is turned off. See also virtual memory.

recall An operation that retrieves the removed, unnamed data attribute from remote storage and places it on the managed volume. The placeholder is replaced on the managed volume with a copy of the file from remote storage. Upon completion of the recall, the file becomes a premigrated file.

Redundant Array of Independent Disks (RAID) A method used to standardize and categorize fault-tolerant disk systems. RAID levels provide various mixes of performance, reliability, and cost. Some servers provide three of the RAID levels: Level 0 (striping), Level 1 (mirroring), and Level 5 (RAID-5). See also fault tolerance; RAID-5 volume.

registry A database repository for information about a computer's configuration. The registry contains information that Windows continually references during operation, such as:

- Profiles for each user
- The programs installed on the computer and the types of documents that each can create
- Property settings for folders and program icons
- What hardware exists on the system
- Which ports are being used
- The registry is organized hierarchically as a tree, and it is made up of keys and their subkeys, hives, and entries.

registry key An identifier for a record or group of records in the registry. See also registry.

remote procedure call (RPC) A message-passing facility that allows a distributed application to call services that are available on various computers on a network. Used during remote administration of computers.

Remote Storage A data management service used to migrate infrequently accessed files from local storage to remote storage. Migrated files are recalled transparently when the user opens the file. See also local storage; validation.

Removable Storage A service used for managing removable media (such as tapes and discs) and storage devices (libraries). Removable Storage allows applications to access and share the same media resources. See also library.

reparse points NTFS file system objects that have a definable attribute containing user-controlled data and that are used to extend functionality in the input/output (I/O) subsystem. See also attribute; NTFS file system; object.

Request for Comments (RFC) An official document of the Internet Engineering Task Force (IETF) that specifies the details for protocols included in the TCP/IP family. See also Transmission Control Protocol/Internet Protocol (TCP/IP).

reserved state A state that indicates that the second side of a two-sided medium is available only to the application that has already allocated the first side.

resolver DNS client programs used to look up DNS name information. Resolvers can be either a small *stub* (a limited set of programming routines that provide basic query functionality) or larger programs that provide additional lookup DNS client functions, such as caching. See also caching; caching resolver.

resource Generally, any part of a computer system or network, such as a disk drive, a printer, or memory, that can be allotted to a running program or a process.

For Device Manager, any of four system components that control how the devices on a computer work. These four system resources are interrupt request (IRQ) lines, direct memory access (DMA) channels, input/output (I/O) ports, and memory addresses.

For server clusters, a physical or logical entity that is capable of being managed by a cluster, brought online and taken offline, and moved between nodes. A resource can be owned by only a single node at any point in time.

See also direct memory access (DMA); input/output (I/O) port; memory address; node; server cluster.

resource group In a server cluster, a defined collection of resources. Resources that are dependent on each other are typically placed within the same resource group. See also node; resource; server cluster.

resource record (RR) A standard DNS database structure containing information used to process DNS queries. For example, an address (A) resource record contains an IP address corresponding to a host name. Most of the basic resource record types are defined in RFC 1035, but additional RR types have been defined in other RFCs and approved for use with DNS. See also Request for Comments (RFC).

response time The amount of time required to do work from start to finish. In a client/server environment, this is typically measured on the client side.

RFC See definition for Request for Comments (RFC).

robotic library A library consisting of media, a robotic media changer, and a drive that accesses media for read and write operations. See also library.

S

scaling The process of adding processors to a system to achieve higher throughput.

sector A 512-byte unit of physical storage on a hard disk. Windows file systems allocate storage in clusters, where a cluster is one or more contiguous sectors. See also file system.

security descriptor A data structure that contains security information associated with a protected object. Security descriptors include information about who owns the object, who can access it and in what way, and what types of access are audited. See also discretionary access control list (DACL); group; object; system access control list (SACL).

security ID (SID) A data structure of variable length that identifies user, group, and computer accounts. Every account on a network is issued a unique SID when the account is first created. Internal processes in Windows refer to an account's SID rather than the account's user or group name.

server cluster A group of computers, known as *nodes*, working together as a single system to ensure that mission-critical applications and resources remain available to clients. A server cluster presents the appearance of a single server to a client. See also node.

Server Message Block (SMB) A file-sharing protocol designed to allow networked computers to transparently access files that reside on remote systems over a variety of networks. The SMB protocol defines a series of commands that pass information between computers. SMB uses four message types: session control, file, printer, and message.

shared printer A printer that receives input from more than one computer. For example, a printer attached to another computer on the network can be shared so that it is available for you to use. Also called a *network printer*.

Simple Mail Transfer Protocol (SMTP) A member of the TCP/IP suite of protocols that governs the exchange of electronic mail between message transfer agents. See also Transmission Control Protocol/Internet Protocol (TCP/IP).

Simple Network Management Protocol (SNMP) A network protocol used to manage TCP/IP networks. In Windows, the SNMP service is used to provide status information about a host on a TCP/IP network. See also Transmission Control Protocol/Internet Protocol (TCP/IP).

simple volume A dynamic volume made up of disk space from a single dynamic disk. A simple volume can consist of a single region on a disk or multiple regions of the same disk that are linked together. If the simple volume is not a system volume or boot volume, you can extend it within the same disk or onto additional disks. If you extend a simple volume across multiple disks, it becomes a spanned volume. You can create simple volumes only on dynamic disks. Simple volumes are not fault tolerant, but you can mirror them to create mirrored volumes on computers running the Windows 2000 Server or Windows Server 2003 families of operating systems. See also dynamic disk; dynamic volume; fault tolerance; mirrored volume; spanned volume; volume.

small computer system interface (SCSI) A standard high-speed parallel interface defined by the American National Standards Institute (ANSI). A SCSI interface is used for connecting microcomputers to peripheral devices, such as hard disks and printers, and to other computers and local area networks (LANs). See also local area network (LAN).

SMTP See definition for Simple Mail Transfer Protocol (SMTP).

snap-in A type of tool that you can add to a console supported by Microsoft Management Console (MMC). A standalone snap-in can be added by itself; an extension snap-in can be added only to extend the function of another snap-in. See also Microsoft Management Console (MMC).

SNMP See definition for Simple Network Management Protocol (SNMP).

soft affinity A mechanism designed to optimize performance in a multiprocessor environment. Soft affinity favors scheduling threads on the processor in which they recently ran or on the ideal processor for the thread. With soft affinity, the efficiency of the processor cache is higher, because threads often run on the processor on which they previously ran. Soft affinity does not restrict a thread to run on a given processor.

software trap In programming, an event that occurs when a microprocessor detects a problem with executing an instruction, which causes it to stop.

spanned volume A dynamic volume consisting of disk space on more than one physical disk. You can increase the size of a spanned volume by extending it onto additional dynamic disks. You can create spanned volumes only on dynamic disks. Spanned volumes are not fault tolerant and cannot be mirrored. See also dynamic disk; dynamic volume; fault tolerance; mirrored volume; simple volume; volume.

spooling A process on a server in which print documents are stored on a disk until a printer is ready to process them. A spooler accepts each document from each client, stores it, and then sends it to a printer when the printer is ready. See also print spooler.

Standard TCP/IP Port Monitor A port monitor that connects a print server running Windows 2000, Windows XP, or Windows Server 2003 to network printers that use the TCP/IP protocol. It replaces LPRMON for TCP/IP printers connected directly to the network through a network adapter. See also port monitor; print server; Transmission Control Protocol/Internet Protocol (TCP/IP).

Stop error A serious error that affects the operating system and that could place data at risk. The operating system generates an obvious message, a screen with the Stop error, rather than continuing on and possibly corrupting data. Also called a *fatal system error*.

Stop message A character-based, full-screen error message displayed on a blue background. A Stop message indicates that the Windows kernel detected a condition from which it cannot recover. Each message is uniquely identified by a Stop error code (a hexadecimal number) and a string indicating the error's symbolic name. Stop messages are usually followed by up to four additional hexadecimal numbers, enclosed in parentheses, which identify developer-defined error parameters. A driver or device might be identified as the cause of the error. A series of troubleshooting tips are also displayed, along with an indication that, if the system was configured to do so, a memory dump file was saved for later use by a kernel debugger. See also Stop error.

storage pool A unit of storage administered by Removable Storage and composed of homogenous storage media. A storage pool is a self-contained storage area with homogenous characteristics (for example, random access, sequential access, read/write, and write-once).

stream A sequence of bits, bytes, or other small structurally uniform units.

striped volume A dynamic volume that stores data in stripes on two or more physical disks. Data in a striped volume is allocated alternately and evenly (in stripes) across the disks. Striped volumes offer the best performance of all the volumes that are available in Windows, but they do not provide fault tolerance. If a disk in a striped volume fails, the data in the entire volume is lost. You can create striped volumes only on dynamic disks. Striped volumes cannot be mirrored or extended. See also dynamic disk; dynamic volume; fault tolerance; mirrored volume; volume.

subnet mask A 32-bit value that enables the recipient of Internet Protocol version 4 (IPv4) packets to distinguish the network ID and host ID portions of the IPv4 address. Typically, subnet masks use the format 255.*x.x.x*. IPv6 uses network prefix notations rather than subnet masks. See also IP address.

symmetric interrupt distribution A mechanism for distributing interrupts across available processors.

system access control list (SACL) The part of an object's security descriptor that specifies which events are to be audited per user or group. Examples of auditing events are file access, logon attempts, and system shutdowns. See also discretionary access control list (DACL); object; security descriptor.

System Monitor A tool that supports detailed monitoring of the use of operating system resources. System Monitor is hosted, along with Performance Logs and Alerts, in the Performance console. The functionality of System Monitor is based on Windows NT Performance Monitor, not Windows 98 System Monitor.

systemroot The path and folder name where the Windows system files are located. Typically, this is C:\Windows, although you can designate a different drive or folder when you install Windows. You can use the value %systemroot% to replace the actual location of the folder that contains the Windows system files. To identify your systemroot folder, click Start, click Run, type **%systemroot%**, and then click OK.

T

T1 A communication line with a data transmission rate of 1.544 megabits per second (Mbps). A T1 line is also known as a *DS-1 line*.

TCP/IP See definition for Transmission Control Protocol/Internet Protocol (TCP/IP).

thread A type of object within a process that runs program instructions. Using multiple threads allows concurrent operations within a process and enables one process to run different parts of its program on different processors simultaneously. A thread has its own set of registers, its own kernel stack, a thread environment block, and a user stack in the address space of its process. See also kernel.

thread state A numeric value indicating the execution state of the thread. Numbered 0 through 5, the states seen most often are 1 for ready, 2 for running, and 5 for waiting. See also thread.

Time to Live (TTL) For Internet Protocol (IP), a field in the IP header of an IP packet that indicates the maximum number of links over which the packet can travel before being discarded by a router.

For DNS, TTL values are used in resource records within a zone to determine how long requesting clients should cache and use this information when it appears in a query response answered by a DNS server for the zone.

See also DNS server; resource record (RR); Transmission Control Protocol/Internet Protocol (TCP/IP).

Token Ring The Institute of Electrical and Electronics Engineers (IEEE) 802.5 standard that uses a token-passing technique for media access control (MAC). Token Ring supports media of both shielded and unshielded twisted pair wiring for data rates of 4 megabits per second (Mbps) and 16 megabits per second. See also media access control (MAC).

total instance A unique instance that contains the performance counters that represent the sum of all active instances of an object. See also object; performance counter.

track A thin concentric band that stores data on a hard disk. A hard disk contains multiple platters, and each platter contains many tracks. Each track is divided into units of storage called *sectors*. Track numbers start at 0 and progress in order, with track 0 at the outer track of a hard disk. See also sector.

transactional message For Message Queuing, a message that can be sent and received only from within a transaction. This type of message returns to its prior state when a transaction is terminated abruptly. A transactional message is removed from a queue only when the transaction is committed; otherwise, it remains in the queue and can be subsequently read during another transaction. See also queue.

Transmission Control Protocol/Internet Protocol (TCP/IP) A set of networking protocols widely used on the Internet that provides communications across interconnected networks of computers with diverse hardware architectures and various operating systems. TCP/IP includes standards for how computers communicate and conventions for connecting networks and routing traffic.

TrueType fonts Fonts that are scalable and sometimes generated as bitmaps or soft fonts, depending on the capabilities of your printer. TrueType fonts are device-independent fonts that are stored as outlines. They can be sized to any height, and they can be printed exactly as they appear on the screen. See also font.

truncate To remove a premigrated file in Remote Storage and replace it with a Remote Storage identifier or placeholder, thus reclaiming space on the local volume. See also premigrated file; Remote Storage.

TTL See definition for Time to Live (TTL).

U

UCS See definition for Universal Character Set (UCS).

UNC See definition for Universal Naming Convention (UNC).

Unicode A character encoding standard developed by the Unicode Consortium that represents almost all the written languages of the world. The Unicode character repertoire has multiple representation forms, including UTF-8, UTF-16, and UTF-32. Most Windows interfaces use the UTF-16 form. See also Universal Character Set (UCS).

Uniform Resource Locator (URL) An address that uniquely identifies a location on the Internet. A URL for a World Wide Web site is preceded by *http://*, as in the fictitious URL *http://www.example*

.microsoft.com. A URL can contain more detail, such as the name of a page of hypertext, usually identified by the file name extension .html or .htm.

uninterruptible power supply (UPS) A device that connects a computer and a power source to ensure that electrical flow is not interrupted. UPS devices use batteries to keep the computer running for a period of time after a power failure. UPS devices usually provide protection against power surges and brownouts as well.

Universal Character Set (UCS) An international standard character set reference that is part of the Unicode standard. The most widely held existing version of the UCS standard is UCS-2, which specifies 16-bit character values currently accepted and recognized for use to encode most of the world's languages. See also Unicode.

Universal Naming Convention (UNC) A convention for naming files and other resources beginning with two backslashes (\), indicating that the resource exists on a network computer. UNC names conform to the \\*servername**sharename* syntax, where *servername* is the server's name and *sharename* is the name of the shared resource. The UNC name of a directory or file can also include the directory path after the share name, by using the following syntax: \\*servername**sharename**directory**filename*.

universal serial bus (USB) An external bus that supports Plug and Play installation. Using USB, you can connect and disconnect devices without shutting down or restarting your computer. You can use a single USB port to connect up to 127 peripheral devices, including speakers, telephones, CD-ROM drives, joysticks, tape drives, keyboards, scanners, and cameras. A USB port is usually located on the back of your computer near the serial port or parallel port. See also Plug and Play; port.

unnamed data attribute The default data stream of an NTFS file, sometimes referred to as *$DATA*.

unprepared state In Removable Storage, a state that indicates a side of a medium is not claimed, used, or available for use by any application. The side is available for use after Removable Storage writes a free label on the medium. See also Removable Storage.

unrecognized media pool A repository of blank media and media that are not recognized by Removable Storage.

unrecognized state A state that indicates that the label types and label IDs of a medium are not recognized by Removable Storage.

UPS See definition for uninterruptible power supply (UPS).

USB See definition for universal serial bus (USB).

V

validation The process of comparing files on local volumes with their associated data in secondary storage by Remote Storage. Volumes that are validated ensure that the correct data is recalled from remote storage when a user attempts to open the file from a local volume. See also Remote Storage; volume.

value bar The area of the System Monitor graph or histogram display that shows last, average, minimum, and maximum statistics for the selected counter.

virtual local area network (VLAN) A logical grouping of hosts on one or more local area networks (LANs) that allows communication to occur between hosts as if they were on the same physical LAN. See also local area network (LAN).

virtual memory Temporary storage used by a computer to run programs that need more memory than the computer has. For example, programs could have access to 4 gigabytes (GB) of virtual memory on a computer's hard drive, even if the computer has only 32 megabytes (MB) of random access memory (RAM). The program data that does not currently fit in the computer's memory is saved into paging files. See also random access memory (RAM); Virtual Memory Size; virtual printer memory.

Virtual Memory Size In Task Manager, the amount of virtual memory, or address space, committed to a process. See also virtual memory.

virtual printer memory In a PostScript printer, a part of memory that stores font information. The memory in PostScript printers is divided into two areas: banded memory and virtual memory. The banded memory contains graphics and page-layout information needed to print your documents. The virtual memory contains any font information that is sent to your printer either when you print a document or when you download fonts. See also PostScript printer; virtual memory.

VoIP (Voice over Internet Protocol) A method for sending voice over a local area network (LAN), a wide area network (WAN), or the Internet using TCP/IP packets. See also local area network (LAN); Transmission Control Protocol/Internet Protocol (TCP/IP); wide area network (WAN).

volume An area of storage on a hard disk. A volume is formatted by using a file system, such as file allocation table (FAT) or NTFS, and has a drive letter assigned to it. You can view the contents of a volume by clicking its icon in Windows Explorer or in My Computer. A single hard disk can have multiple volumes, and volumes can also span multiple disks. See also file allocation table (FAT); NTFS file system; simple volume; spanned volume.

volume decommission A process that occurs when a managed volume is no longer accessible. The data in remote storage is no longer associated with a placeholder or a premigrated file. This space is available for space reclamation.

volume set A volume that consists of disk space on one or more physical disks. A volume set is created by using basic disks and is supported only in Windows NT 4.0 or earlier. Volume sets were replaced by spanned volumes, which use dynamic disks. See also basic disk; dynamic disk; partition; spanned volume; volume.

volume shadow copy A volume that represents a duplicate of the original volume taken at the time the copy began. See also differential data; volume.

W

WAN See definition for wide area network (WAN).

wide area network (WAN) A communications network that connects geographically separated locations and uses long-distance links of third-party telecommunications vendors. See also local area network (LAN).

WINS database The database used to register and resolve computer names to IP addresses on Windows-based networks. The contents of this database are replicated at regular intervals throughout the network.

working set For a process, the amount of physical memory assigned to the process by the operating system.

Index

Symbols and Numerics

% Disk Time counter, 288
% Idle Time counter, 282
% Interrupt Time counter, 253
% Privileged Time counter, 254
% Processor Time counter, 244, 248
/3GB boot switch, 611
32-bit extended virtual addressing
  /3GB boot switch, 611
  Address Window Extension (AWE), 611, 615–617
  application support, 617–619
  Exchange, 618
  extended process private virtual addresses, 611–613
  options, 611–617
  Oracle, 619
  overview, 608
  Physical Address Extension (PAE), 611, 613
  SAS, 619
  SQL Server, 618
  system pools, 620
  system virtual memory shortages, 620
  virtual memory addressing constraints, 609
32-bit virtual memory addressing limits
  investigating pool usage, 461–464
  overview, 456
  system pools, 458–461
  system virtual memory, 457
64-bit architecture, 543–545
64-bit virtual memory, 621

A

absolute exceptions, 238
access rights, 394
Active Directory, trace providers, 221
Active Server Pages (ASP), 578–579
Active Server Pages\Request Execution Time counter, 305
Active Server Pages\Request Queue Time counter, 306
Active Server Pages\Requests Executing counter, 307
Active Server Pages\Requests Queued counter, 308
ActiveX control
  adding performance counters, 639
  adding to Web pages, 629
  CollectSample method, 634
  configuring, sample commands, 631
  configuring appearance, 636
  configuring color schemes, 637
  configuring display types, 632
  configuring font styles, 638
  configuring performance counters, 640
  configuring sampling rates, 634
  configuring user interface, 637
  creating Web pages for monitoring performance, 643
  customizing, 630
  drag-and-drop support, 645
  event methods, 635
  manually retrieving performance data, 634
  Memory Allocation report, 645
  OnCounterDeleted method, 636
  OnSampleCollected method, 635
  removing performance counters, 641
  tracking performance counters using counter paths, 642
adaptive retransmission
  fast retransmit, 534
  Karn's algorithm, 534
  overview, 532
  retransmission timeout (RTO), 533, 534
  timestamps option, 534
Address Window Extension (AWE)
  extended virtual addressing, 611, 615–617
  limitations, 617
  memory-resident disk caches, 277
Adlb.exe, 132
administrative controls, 128
advanced performance topics
  memory performance. *See* memory and paging
  overview, 537
  processor performance. *See* processors
  System Monitor Automation Interface. *See* System Monitor Automation Interface
affinity
  asymmetric partitioning, 591
  hard processor, 591
  processors, 60, 558
alerts
  applications, 337
  automated counter log processing, 362
  configuring, 226–229
  configuring notifications, 229–233
  daily server monitoring procedures, 330–337
  effective use, 330–337
  logging events to application logs, 229
  overview, 225
  procedures, 335
  running programs, 231
  scheduling, 226–227
  sending network messages, 230
  starting performance data logs, 230
  thresholds, 228
  triggering counter logs automatically, 332–335
anticipatory paging, 73
application logs, 229
application servers, 576–578
applications
  Active Server Pages\Request Execution Time counter, 305
  Active Server Pages\Request Queue Time counter, 306
  Active Server Pages\Requests Executing counter, 307
  Active Server Pages\Requests Queued counter, 308
  alerts, 337
  ASP.NET\Request Execution Time counter, 305
  ASP.NET\Request Queue Time counter, 306

applications, *continued*
  ASP.NET\Requests Executing counter, 307
  ASP.NET\Requests Queued counter, 308
  Available Threads counter, 301
  Elapsed Time counter, 242
  File Server service, 299
  ideal vs. actual scalability, 9
  maintaining server applications, 297
  monitoring using Task Manager, 156
  Process(n)\Elapsed Time counter, 242
  Queue Length counter, 302
  Server Work Queues(n)\Available Threads counter, 301
  Server Work Queues(n)\Queue Length counter, 302
  Server\Work Item Shortages counter, 299
  symmetric partitioning, 595
  symmetric portioning, 593
  thread pooling, 298, 583
  troubleshooting performance problems, 6
  Web-based applications, 303–309
  Work Item Shortages counter, 299
architecture
  64-bit, 543–545
  ccNUMA, 604–607
  disks, 92–95
  IIS thread pooling, 573–576
  Internet, 100
  microarchitecture, 542
  performance monitoring, 207–211
  system. *See* system architecture
arrival rate distribution, 30
ASP (Active Server Pages), 578–579
ASP.NET
  Little's Law, 309
  ProcessorThreadMax property, 307
  thread pooling, 581–583
ASP.NET\Request Execution Time counter, 305
ASP.NET\Request Queue Time counter, 306
ASP.NET\Requests Executing counter, 307
ASP.NET\Requests Queued counter, 308
asymmetric multiprocessing, 250
asymmetric partitioning
  affinity, 591
  application-level settings, 593–595
  hard processor affinity, 591
  Interrupt-Affinity Filter (Intfiltr.exe), 599–601
  overview, 588–591
  Windows System Resource Manager (WSRM), 595–599
automated counter log processing
  alert-generated counter log management, 362
  archiving reports, 362
  backup file cleanup, 360
  daily counter log management, 360
  defining log configurations for multiple servers, 364
  monthly reports and archiving, 362–364
  overview, 354
  reports, 362
  sample post-processing script, 355–358
  scheduled monthly reports and archiving, 362–364
  script initialization, 358
  summarized counter log creation, 360
automated performance monitoring
  counter logs, 169–177
  overview, 168
  Performance Logs and Alerts. *See* Performance Logs and Alerts
Available Bytes counter, 260–262
Available Bytes pool
  Least Recently Used (LRU) page replacement policy, 81–82
  page fault resolution, 66
Available Threads counter, 301
Avg. Disk Queue Length counter, 286
Avg. Disk secs/transfer counter, 281
AWE. *See* Address Window Extension (AWE)

B

back-off and retry algorithm, 109
balanced systems, 22
bandwidth, 16, 103
base priorities, 57
baseline data, troubleshooting, 402
baseline measurements, 321
bottlenecks
  analysis, 25–27, 402–404
  balanced systems, 22
  current performance levels, 403
  decomposition, 404
  fair scheduling, 24
  identifying, 237
  Little's Law, 35
  managing queues for optimal performance, 23
  memory bottleneck primary indicators, 436
  overview, 22
  preventing, 35
  priority queuing with preemptive scheduling, 24
  processors, 246
  queue length, 403, 407–409
  resource utilization, 403, 407–409
  scalability, 10–12
  serving shorter requests first, 25
  starvation, 24
  troubleshooting baseline data, 402
  troubleshooting overview, 402
  troubleshooting processors, 246
  unfair scheduling, 24
byte sequence numbers, 120
Bytes Total/sec counter, 293

C

Cache Bytes counter, 263
cache coherence, 555
cache coherent NUMA (ccNUMA) architectures, 604–607
cache memory, 485–487, 543
cache miss, 277
caching controllers, 485–487
capacity planning
  forecasting techniques, 383–388
  historical data for, 346–354
  networks, 515–516
  organizing data, 380–383
  overview, 316, 379
ccNUMA (cache coherent NUMA) architectures, 604–607
central processing unit (CPU). *See* processors
Checkrepl.vbs, 132
Clearmem.exe, 132
clocks, 47
clustered paging I/O, 73
collection queries, 182
CollectSample method, 634

collisions, 108
Commit Limit
  overview, 69
  virtual memory shortages, 452
Committed Bytes counter, 269
committed pages
  anticipatory paging, 73
  clustered paging I/O, 73
  Commit Limit, 69
  multiple paging files, 72
  overview, 69
  paging file extensions, 70–72
  prefetching, 73
concurrency levels, 586
congestion windows, 123, 530–531
connections, 120
Connections Established counter, 295
Consume.exe, 132
context switches
  counters, 250
  overview, 44
Context Switches/sec counter, 251
counter logs
  access rights, 394
  accessing files in SQL Server, 375–377
  analyzing, 176
  automated processing. *See* automated counter log processing
  background performance monitoring, 314
  capacity planning, 316
  capacity planning databases, 348
  configuring, 172
  creating performance logs using Logman, 178
  daily. *See* daily counter logs
  database schemas, 369–373
  disabled performance counters, 398
  file formats, 170
  file management, 171
  identifying runaway processes, 416–430
  local logging with local log files, 390
  local logging with remote log files, 391
  logging local counters, 388–392
  management reporting, 315
  merging using Relog.exe, 193
  missing performance counters, 396–399
  monitoring remote servers in real time, 392–395
  overview, 169, 314
  Perflib error messages, 396–398
  Performance console for remote performance analysis, 394
  permissions, 394
  properties, 175
  remote logging with local log files, 391
  remote logging with remote log files, 392
  restoring corrupt performance counters, 400
  rights, 394
  scenarios, 388–395
  scheduling collection periods, 171
  settings files, 348
  summarizing daily counter logs, 338–342
  summarizing log files using Relog.exe, 196–198
  System Monitor, 176
  triggering automatically, 332–335
  troubleshooting collection problems, 395–400
  working with, 171
  writing Typeperf.exe output, 205
counter settings files
  adding application-specific counters, 323
  daily management reporting, 340–342
  File and Print Server, 328–330
  gathering error indicators, 328–330
  listing, 322
  sample files, 324–327
  using, 322–330
CounterData table
  overview, 370
  retrieving counter log measurement data, 377–379
CounterDetails table, 370–371
counters
  Active Server Pages\Request Execution Time, 305
  Active Server Pages\Request Queue Time, 306
  Active Server Pages\Requests Executing, 307
  Active Server Pages\Requests Queued, 308
  adding to System Monitor ActiveX control, 639
  ASP.NET\Request Execution Time, 305
  ASP.NET\Request Queue Time, 306
  ASP.NET\Requests Executing, 307
  ASP.NET\Requests Queued, 308
  Available Bytes, 260–262
  Available Threads, 301
  Avg. Disk Queue Length, 286
  Avg. Disk secs/transfer, 281
  Bytes Total/sec, 293
  Cache Bytes, 263
  Committed Bytes, 269
  configuring System Monitor ActiveX control, 640
  Connections Established, 295
  context switches, 250
  Context Switches/sec, 251
  Current Disk Queue Length, 284
  Datagrams/sec, 294
  disabling, 210
  disk performance troubleshooting, 464
  disk response time decomposition, 490
  % Disk Time, 288
  Disk Transfers/sec, 283
  disk workload, 489
  effective use, 237
  Elapsed Time, 242
  Free Megabytes, 291
  Free System Page Table Entries, 274
  % Idle Time, 282
  % Interrupt Time, 253
  IPvn\Datagrams/sec, 294
  Logical Disk(n)\Free Megabytes, 291
  memory and paging overview, 256
  memory and paging troubleshooting, 431–433
  Memory\Available Bytes, 260–262
  Memory\Cache Bytes, 263
  Memory\Committed Bytes, 269
  Memory\Free System Page Table Entries, 274
  Memory\Nonpaged Pool Bytes, 273
  Memory\Page Faults/sec, 267
  Memory\Paged Pool Bytes, 273
  Memory\Pages/sec, 258

counters, *continued*
- Memory\Pool Nonpaged Bytes, 264
- Memory\System Cache Resident Bytes, 277
- Memory\Transition Faults/sec, 266
- monitoring multiple using Typeperf, 204
- Network Interface(n)\Bytes Total/sec, 293
- network troubleshooting, 509–514
- Nonpaged Pool Bytes, 273
- obtaining lists of, using Typeperf queries, 201
- overview, 140, 235
- Page Faults/sec, 267
- Paged Pool Bytes, 273
- Paged Pool Failures, 275
- Pages/sec, 258
- paths, 140–143
- Physical Disk(n)\% Disk Time, 288
- Physical Disk(n)\% Idle Time, 282
- Physical Disk(n)\Avg. Disk Queue Length, 286
- Physical Disk(n)\Avg. Disk secs/transfer, 281
- Physical Disk(n)\Current Disk Queue Length, 284
- Physical Disk(n)\Disk Transfers/sec, 283
- Physical Disk(n)\Split IO/sec, 290
- Pool Nonpaged Bytes, 264
- primary indicators, 235
- Private Bytes, 271
- % Privileged Time, 254
- Process(instancename)\% Processor Time, 248
- Process(n)\Elapsed Time, 242
- Process(n)\Private Bytes, 271
- Process(*)\Working Set, 262
- process-level CPU consumption, 248
- Processor Queue Length, 246
- % Processor Time, 244, 248
- processor utilization by processor, 250
- Processor(_Total)\% Interrupt Time, 253
- Processor(_Total)\% Privileged Time, 254
- Processor(_Total)\% Processor Time, 244
- processors, measuring utilization, 243
- processors, normalizing utilization measures, 245
- processors, utilization overview, 242
- Queue Length, 302
- remote monitoring, 210
- removing from System Monitor ActiveX control, 641
- Requests Executing, 307
- Requests Queued, 308
- retrieving from remote computers, 203
- secondary indicators, 235
- Segments Received/sec, 295
- selection logic, 420–423
- Server Work Queues(n)\Available Threads, 301
- Server Work Queues(n)\Queue Length, 302
- Server\Paged Pool Failures, 275
- Server\Work Item Shortages, 299
- settings files. *See* counter settings files
- Split IO/sec, 290
- System Cache Resident Bytes, 277
- System Up Time, 241
- System\Context Switches/sec, 251
- System\Processor Queue Length, 246
- System\System Up Time, 241
- TCPvn\Connections Established, 295
- TCPvn\Segments Received/sec, 295
- Terminal services\Total sessions, 310
- text string files, 208
- Total sessions, 310
- tracking using paths, 642
- Transition Faults/sec, 266
- troubleshooting processor bottlenecks, 246
- types, 143–144
- Work Item Shortages, 299
- Working Set, 262

CPU (central processing unit). *See* processors
crisis mode interventions, 8
Current Disk Queue Length counter, 284
Custreasonedit.exe, 132

D

daily counter logs
- automated counter log processing, 360
- baseline measurements, 321
- constructing file names, 321
- counter settings files, 322–330
- establishing daily performance logging, 317
- executing Log Manager (Logman.exe), 321
- identifying source machines, 321
- logging local counters to local disks, 319
- logging local counters to network shares, 320
- overview, 317
- starting logging data, 319
- Windows Time Service, 320

daily management reporting
- consolidating performance data from multiple servers, 339–342
- counter settings files, 340–342
- overview, 338
- sample management reports, 342–345
- summarizing daily counter logs, 338–342

Daily Processor Utilization report, 342
daily server monitoring procedures
- alerts, 330–337
- automated counter log processing. *See* automated counter log processing
- baseline measurements, 321
- consolidating performance data from multiple servers, 339–342
- constructing file names, 321
- counter settings files, 322–330
- daily counter logs overview, 317
- daily management reporting, 338–345
- establishing daily performance logging, 317
- executing Log Manager (Logman.exe), 321
- historical data for, 346–354
- identifying source machines, 321
- logging local counters to local disks, 319
- logging local counters to network shares, 320
- overview, 316
- sample management reports, 342–345

starting logging data, 319
summarizing daily counter logs, 338–342
triggering counter logs automatically, 332–335
Windows Time Service, 320
Datagrams/sec counter, 294
decomposition
bottleneck analysis, 404
processor troubleshooting, 409–413
deferred procedure call (DPC), 56
definitions
bandwidth, 16
performance monitoring, 13–22
queue time, 20
queuing system, 14–16
response time, 20
service time, 19
throughput, 16
utilization, 17–19
Depends.exe, 131
derived disk measurements
overview, 286
Physical Disk(n)\% Disk Time counter, 288
Physical Disk(n)\Avg. Disk Queue Length counter, 286
Devcon.exe, 131
DH.exe, 132
diagnosing. *See* troubleshooting
diagnostic tools. *See* tools
Diruse.exe, 131
Disk Defragmenter
analysis reports, 501–503
analyzing volumes before defragmenting them, 501
disk performance improvements, 504
overview, 499
using effectively, 500
Disk Free Space report, 344
disk mirroring, 480
disk striping, 480
% Disk Time counter, 288
Disk Transfers/sec counter, 283
disk troubleshooting
caching controllers, 485–487
configuration strategies, 495–499
counters for decomposing disk response time, 490
counters for disk workload, 489
diagnosing performance problems, 489–494
Disk Defragmenter, 499–504
disk I/O workload characteristics, 472
establishing disk drive performance baseline, 473–477
file-level access patterns, 505–509
files causing most disk I/Os, 505
formulas for calculating disk measurements, 465
formulas for calculating disk service time, 490
overview, 464
performance counters to log, 464
performance expectations, 466
performance-oriented disk configuration options, 496–497
RAID, 479–487
storage controllers, 477–479
tuning strategies, 495–499
virtual disks, 477–479, 487
disks
architecture, 92–95
Avg. Disk Queue Length counter, 286
Avg. Disk secs/transfer counter, 281
Current Disk Queue Length counter, 284
derived measurements, 286–289
Disk Free Space report, 344
% Disk Time counter, 288
Disk Transfers/sec counter, 283
excessive paging, 435
Free Megabytes counter, 291
% Idle Time counter, 282
Logical Disk(n)\Free Megabytes counter, 291
logical disks, 280
monitoring operations overview, 279
performance expectations, 91, 466
performance measurements, 95–97
performance reporting, 344
Physical Disk(n)\% Disk Time counter, 288
Physical Disk(n)\% Idle Time counter, 282
Physical Disk(n)\Avg. Disk Queue Length counter, 286
Physical Disk(n)\Avg. Disk secs/transfer counter, 281
Physical Disk(n)\Current Disk Queue Length counter, 284
Physical Disk(n)\Disk Transfers/sec counter, 283
Physical Disk(n)\Split IO/sec counter, 290
physical disks, 280
RAID, 479–487
space usage measurements, 291–292
split I/Os, 290
Split IO/sec counter, 290
troubleshooting. *See* disk troubleshooting
virtual disks, 477–479, 487
dispatching priorities
dynamic priority adjustments, 58–60
overview, 56–58
DisplayToID table, 372–373
DPC (deferred procedure call), 56
DPC mode, 411
dual logical processors, 602
dynamic priority adjustments, 58–60
dynamic range (1–15), 57
dynamic volumes, 484

E

Elapsed Time counter, 242
Empty.exe, 132
Ethernet
back-off and retry, 109
collision detection, 108
exponential back-off and retry, 109
maximum transmission unit (MTU), 108
overview, 106
performance monitoring, 110–111
ETW. *See* Event Tracing for Windows (ETW)
event methods, 635
event traces
event time stamps, 224
overview, 127, 212
Performance Logs and Alerts, 213
Event Tracing for Windows (ETW)
overview, 127, 212
page faults per process, 445
Performance Logs and Alerts, 213
events, 229
exception-reporting thresholds, 239

excessive paging
  available memory, 439–440
  disk contention, 435
  memory allocation, 441–442
  memory bottleneck primary indicators, 435
  memory reporting tools, 443–448
  overview, 435
  page faults, 435
  page faults per process, 445–448
Exchange Server 2000, 618
Exctrlst.exe. *See* Extensible Counter List (Exctrlst.exe)
Executive
  I/O Manager, 89–91, 95–97
  overview, 40, 41
exponential back-off and retry algorithm, 109
extended process private virtual addresses, 611–613
extended virtual addressing in 32-bit machines
  /3GB boot switch, 611
  Address Window Extension (AWE), 611, 615–617
  application support, 617–619
  Exchange, 618
  extended process private virtual addresses, 611–613
  options, 611–617
  Oracle, 619
  overview, 608
  Physical Address Extension (PAE), 611, 613
  SAS, 619
  SQL Server, 618
  system pools, 620
  system virtual memory shortages, 620
  virtual memory addressing constraints, 609
Extensible Counter List (Exctrlst.exe)
  overview, 131
  Performance Data Helper (PDH) processing, 209

F

fair scheduling
  bottlenecks, 24
  round robin, 24, 58
false sharing, 556–558
fast retransmit, 534
File and Print Server
  counter log settings files, 348
  counter settings files, 328–330
File I/O report, 505
File Server service
  overview, 299
  Server Work Queues(n)\Available Threads counter, 301
  Server Work Queues(n)\Queue Length counter, 302
  Server\Work Item Shortages counter, 299
  thread pooling, 570–573
file-level access patterns, 505–509
forecasting techniques
  latent demand, 387
  linear regression, 384–385
  nonlinear regression, 385–387
  overview, 383
Free Megabytes counter, 291
Free System Page Table Entries counter, 274
Freedisk.exe, 129

H

HAL (hardware abstraction layer), 41
hard affinity, 61
hard disks. *See* disks
hardware
  disks. *See* disks
  processors. *See* processors
  troubleshooting performance problems, 6
hardware abstraction layer (HAL), 41
Health_chk.cmd, 131
historical data for capacity planning
  accumulating historical data, 349
  capacity planning databases, 348
  counter log settings files, 348
  creating historical summary logs, 347–349
  creating SQL Server PDB, 350–353
  editing daily counter logs, 347
  overview, 346
  populating repositories, 353–354
  SQL Server, reasons to use, 347
  summarizing daily counter logs, 347
historical summary logs, 347–349
Hyper-Threading, 602–604

I

ICMP (Internet Control Message Protocol), 117
ideal processors, 61
idle loops, 559
Idle thread, 51
% Idle Time counter, 282
instruction execution throughput
  64-bit architecture, 543–545
  caches, 543
  microarchitecture, 542
  Out-of-Order execution, 542
  overview, 539
  pipelining, 541
  predication, 542
  processor hardware basics, 539–540
  processor performance hardware mechanisms, 541–543
  simultaneous multithreading (SMT), 543
  speculation, 542
  superscalar execution, 542
Internet, architecture, 100
Internet Control Message Protocol (ICMP), 117
Internet Information Services (IIS)
  application servers, 576–578
  critical measurements, 303
  thread pooling, 573–578
  trace providers, 221
  Web-based applications, 303
% Interrupt Time counter, 253
Interrupt-Affinity Filter (Intfiltr.exe)
  asymmetric partitioning, 599–601
  multiprocessors, 599–601
  overview, 132
interrupts
  priority, 55
  processing, 48
Intfiltr.exe. *See* Interrupt-Affinity Filter (Intfiltr.exe)
invalid PTEs
  overview, 80–81
  paging file number (PFN), 80
involuntary waits, 49
I/O Manager
  disk performance measurements, 95–97
  monitoring disk operations overview, 279
  overview, 89–91

IP routing
  Internet Control Message Protocol (ICMP), 117
  overview, 111
  packet headers, 117
  router performance, 115
  routing, 112
  routing tables, 113
IPvn\Datagrams/sec counter, 294

K

Karn's algorithm, 534
Kernel Debugger
  investigating pool usage, 462
  memory allocation, 444–445
  system pools, 459
Kernel mode, 39
Kernrate.exe
  example report on processor usage, 424–430
  identifying processor use, 423–430
  overview, 132, 417
  parameters, 424

L

LAN. *See* local area network (LAN)
latency, 105–106
latent demand, 387
lazy writes, 277
Least Recently Used (LRU), 81–84
linear regression, 384–385
Little's Law
  ASP.NET counters, 309
  bottleneck prevention, 35
  overview, 33
  performance measurements, 35
  response time revisited, 34
load-testing tools. *See* tools
local area network (LAN)
  network capacity planning, 515–516
  Network Monitor packet traces, 518–522
  performance overview, 514
  SPA system network diagnosis, 516–517
  TCP congestion control, 522–524
  tuning, 522–526
local logging
  local log files, 390
  remote log files, 391
lock collisions, 584
lock contention, 552
Lodctr.exe, 129
Log Manager (Logman.exe)
  Active Directory, 221
  adding versioning information to log file names, 185
  automating data logging, 317
  command parameters, 318
  command syntax, 179, 218
  configuring output files, 184
  configuring sampling intervals, 188
  creating collection queries, 182
  creating new log files automatically, 187
  creating performance logs, 178
  deleting data collections, 189
  establishing daily performance logging, 317
  event time stamps, 224
  executing, 321
  file size limit checking, 187, 224
  formatting output files, 186
  interactive sessions, 218
  Internet Information Services (IIS), 221
  logging local counters to local disks, 319
  logging local counters to network shares, 320
  managing data collections using Windows Script Host (WSH), 190
  monitoring remote computers, 183
  overview, 129, 179
  parameters for event tracing, 222
  parsing command parameters, 318
  scheduling data collections, 188
  settings files, 182
  specifying maximum sizes for output files, 186
  starting data collections, 189
  starting logging data, 319
  stopping data collections, 189
  trace logs, 217–224
  trace providers, 219–223
  updating data collections, 189
  Windows Time Service, 320
Logical Disk(n)\Free Megabytes counter, 291
logical disks
  derived disk measurements, 286–289
  Free Megabytes counter, 291
  overview, 280
  performance expectations, 466
  space usage measurements, 291–292
Logman.exe. *See* Log Manager (Logman.exe)
logs
  adding versioning information to log file names, 185
  application, 229
  counter. *See* counter logs
  creating performance, using Logman, 178
  data formats of Typeperf.exe logs, 206
  file formats, 420–423
  filtering log files using Relog.exe, 194
  historical summary, 347–349
  managing performance, 191
  starting performance data, 230
  summarizing log files using Relog.exe, 196–200
  trace. *See* trace logs
LRU (Least Recently Used), 81–84

M

management by exception, 238
management reporting, daily. *See* daily management reporting
maximum transmission unit (MTU), 108
measuring performance. *See* counters; performance monitoring
Memory Allocation report, 645
memory and paging. *See also* physical memory; virtual memory
  cache memory, 485–487
  clustered paging I/O, 73
  committed pages, 69–74
  extended virtual addressing in 32-bit machines. *See* extended virtual addressing in 32-bit machines
  forecasting memory requirements, 622–628
  Memory Allocation report, 645
  memory leaks, 256
  memory status bits, 79
  monitoring rates overview, 256
  overview, 61
  page fault resolution, 65–69
  page replacement, 81–85

memory and paging, *continued*
  Page Table entries (PTEs), 78–81
  Page Tables, 63–65
  paging files, 70–72
  performance, advanced topics overview, 608
  process virtual address spaces, 74–77
  process working set management, 85–89
  real memory. *See* physical memory
  troubleshooting. *See* memory and paging troubleshooting
  virtual addressing, 62
memory and paging troubleshooting
  32-bit virtual memory addressing limits, 456–464
  counters to evaluate when troubleshooting performance, 431–433
  evaluating specific problems, 434
  excessive paging. *See* excessive paging
  memory bottleneck primary indicators, 435
  memory reporting tools, 443–448
  overview, 430
  virtual memory shortages, 448–456
memory contention index, 624–626
memory leaks
  overview, 256
  virtual memory leaks, 451
Memory Manager
  available bytes, 260
  transition faults, 266
Memory Pool Monitor (Poolmon.exe)
  investigating pool usage, 463
  overview, 131
Memory\Available Bytes counter, 260–262
Memory\Cache Bytes counter, 263
Memory\Committed Bytes counter, 269
Memory\Free System Page Table Entries counter, 274
Memory\Nonpaged Pool Bytes counter, 273
Memory\Page Faults/sec counter, 267
Memory\Paged Pool Bytes counter, 273
Memory\Pages/sec counter, 258
Memory\Pool Nonpaged Bytes counter, 264
memory-resident disk caches, 276–279
Memory\System Cache Resident Bytes counter, 277
Memory\Transition Faults/sec counter, 266
Memsnap.exe, 131
Memtriage.exe, 132
microarchitecture, 542
Microsoft Event Tracing for Windows. *See* Event Tracing for Windows (ETW)
Microsoft SQL Server. *See* SQL Server
Microsoft Visual Basic Scripting Edition. *See* VBScript
monitoring performance. *See* performance monitoring
monitoring tools. *See* tools
Msinfo32.exe, 130
MTU (maximum transmission unit), 108
multipathing, 472
multiprocessing, 250
multiprocessors
  asymmetric partitioning. *See* asymmetric partitioning
  cache coherence, 555
  ccNUMA architectures, 604–607
  configuration strategies overview, 563
  false sharing, 556–558
  Hyper-Threading, 602–604
  idle loop, 559
  Interrupt-Affinity Filter (Intfiltr.exe), 599–601
  lock contention, 552
  minimizing serialization delays, 564
  nonuniform memory access (NUMA), 604–607
  optimizing, 558–559
  overview, 548
  partitioning using WSRM, 595–599
  queued spin locks, 559
  scalability, 551–562
  shared memory multiprocessors, 549–551
  single system images, 550
  snooping, 555
  spin locks, 554
  symmetric multiprocessing, 551
  thread pooling. *See* thread pooling
  TPC performance benchmarks, 561
  tuning strategies overview, 563
  workload parallelism, 552
multithreading
  overview, 45
  simultaneous multithreading (SMT), 543, 602

N

Netcap.exe, 131
Network Interface Traffic report, 345
Network Interface(n)\Bytes Total/sec counter, 293
network interfaces
  bandwidth, 103
  Bytes Total/sec counter, 293
  Ethernet, 106–111
  IP routing, 111–118
  latency, 105–106
  measurements, 293
  Network Interface(n)\Bytes Total/sec counter, 293
  overview, 98
  packets, 98
  protocol stacks, 100–103
  Transmission Control Protocol (TCP), 118–123
Network Monitor
  overview, 130, 233
  packet traces, 518–522
network troubleshooting
  adaptive retransmission, 532–535
  congestion windows, 530–531
  counters to evaluate, 511–514
  counters to log, 509–511
  effective capacity of WAN links, 529
  LAN performance overview, 514
  LAN tuning, 522–526
  network capacity planning, 515–516
  Network Monitor packet traces, 518–522
  overview, 509
  round trip time (RTT), 527–531
  SPA system network diagnosis, 516–517
  TCP congestion control, 522–524
  WAN performance overview, 526

networks
capacity planning, 515–516
capturing network packets, 519–522
error conditions, 297
managing traffic, 292–297
monitoring using Task Manager, 163–167
Network Interface Traffic report, 345
troubleshooting. *See* network troubleshooting
nonlinear regression, 385–387
Nonpaged Pool Bytes counter, 273
Nonpaged pools, 272
nonuniform memory access (NUMA), 604–607
normalizing processor utilization measures, 245
NUMA (nonuniform memory access), 604–607

O
objects
multiple instances, 134–135
overview, 134
types, 135–140
OnCounterDeleted method, 636
OnSampleCollected method, 635
operating systems
64-bit platform support, 544
clocks, 47
components, 40–42
Executive, 41
functions, 42
hardware abstraction layer (HAL), 41
Kernel mode, 39
performance-related tools, 129
Privileged state, 39
Processr.sys, 41
system architecture, 38–42
User state, 39
Win32k.sys, 42
Oracle, 619
Out-of-Order execution, 542

P
packets
capturing network packets, 519–522
example packet trace, 102
headers, 117
Network Monitor packet traces, 518–522
overview, 98
payloads, 99
processing, 101–102
PAE. *See* Physical Address Extension (PAE)
Page Fault Monitor (Pfmon.exe)
overview, 132
page faults per process, 445–448
page faults
Available Bytes pool, 66
excessive paging, 435
memory allocation, 445–448
performance considerations, 67
resolving, 65–69
thrashing, 68
Page Faults/sec counter, 267
Page Frame Number (PFN), 87
page replacement, 81–84
Page Table entries (PTEs)
Free System Page Table Entries counter, 274
invalid, 80–81
memory status bits, 79
overview, 78
Page Tables
committed pages, 69
overview, 63–65
Paged Pool Bytes counter, 273
Paged Pool Failures counter, 275
Paged pools
contention, 626–628
virtual memory in system range, 272
Pages/sec counter, 258
paging. *See* memory and paging
paging file number (PFN), 80
paging files, 70–72
partitioning, asymmetric. *See* asymmetric partitioning
PDB. *See* performance database (PDB)
PDH. *See* Performance Data Helper (PDH)
Peak Hour Processor Utilization report, 343
Perflib. *See* Performance Library (Perflib) DLLs
perfmon. *See* Performance Monitor
Performance console, 394
Performance Data Helper (PDH)
counter selection logic, 421
Extensible Counter List (Exctrlst.exe), 209
processing, 208–210
performance database (PDB)
creating, 350–353
populating repositories, 353–354
querying, 373–379
Performance Library (Perflib) DLLs
counter selection logic, 421
error messages, 396–398
overview, 207–208
Performance Logs and Alerts. *See also* alerts; logs
counter logs. *See* counter logs
counter selection logic, 420–423
creating performance logs using Logman, 178
event traces, 213
Event Tracing for Windows (ETW), 213
overview, 130, 168
starting, 171
System Monitor overhead, 418
trace event providers, 215
working with, 178
Performance Monitor
overview, 133
Performance Logs and Alerts, starting, 171
using, 37
viewing charts in real time, 145
performance monitoring
absolute exceptions, 238
advanced performance topics, 537
architecture, 207–211
arrival rate distribution, 30
automated. *See* automated performance monitoring
background counter log sessions, 314
bandwidth, 16
bottlenecks, 22–27
concept overview, 12
concept summary, 36
counter text string files, 208
counters. *See* counters
creating Web pages for, 643
crisis mode interventions, 8
definitions, 13–22
diagnosing performance problems overview, 5–8
Ethernet, 110–111
exception-reporting thresholds, 239
fair scheduling, 24
introducing, 2–12

performance monitoring, *continued*
- learning about, 3
- Little's Law, 33–35
- management by exception, 238
- managing performance logs, 191
- managing queues for optimal performance, 23
- objects, 134–140
- overhead considerations, 7
- overview, 1–12, 235
- page fault resolution, 67
- perfmon. *See* Performance Monitor
- Performance Data Helper (PDH), 208–210
- Performance Library (Perflib) DLLs, 207–208
- Performance Monitor. *See* Performance Monitor
- priority queuing with preemptive scheduling, 24
- proactive, 4–5
- procedures. *See* procedures
- queue depth limits, 32
- queue time, 20, 29–33
- queuing system, 14–16
- relative exceptions, 239
- response time, 20
- scalability, 8–12
- service time, 19
- service time distribution, 30–31
- serving shorter requests first, 25
- statistics overview, 126, 133
- summary, 123
- system architecture. *See* system architecture
- System Monitor. *See* System Monitor
- Task Manager. *See* Task Manager
- throughput, 16
- tools. *See* tools
- triage, 240
- troubleshooting. *See* troubleshooting
- Typeperf.exe. *See* Typeperf.exe
- unfair scheduling, 24
- utilization, 17–19, 29–33
- Utilization Law, 28

permissions, 394

Pfmon.exe. *See* Page Fault Monitor (Pfmon.exe)

PFN (Page Frame Number), 87

PFN (paging file number), 80

Physical Address Extension (PAE)
- extended virtual addressing, 611, 613
- memory-resident disk caches, 277

Physical Disk(n)\% Disk Time counter, 288

Physical Disk(n)\% Idle Time counter, 282

Physical Disk(n)\Avg. Disk Queue Length counter, 286

Physical Disk(n)\Avg. Disk secs/transfer counter, 281

Physical Disk(n)\Current Disk Queue Length counter, 284

Physical Disk(n)\Disk Transfers/sec counter, 283

Physical Disk(n)\Split IO/sec counter, 290

physical disks
- Avg. Disk Queue Length counter, 286
- Avg. Disk secs/transfer counter, 281
- caching controllers, 485–487
- Current Disk Queue Length counter, 284
- derived disk measurements, 286–289
- disk I/O workload characteristics, 472
- % Disk Time counter, 288
- Disk Transfers/sec counter, 283
- establishing disk drive performance baseline, 473–477
- % Idle Time counter, 282
- multipathing, 472
- overview, 280
- performance expectations, 467–472
- RAID, 479–487
- split I/Os, 290
- Split IO/sec counter, 290
- storage controllers, 477–479
- virtual disks, 477–479, 487

physical memory. *See also* memory and paging
- overview, 62
- page fault resolution, 66
- Page Tables, 63–65
- usage, 260

pipelining, 541

Pmon.exe, 132

Pool Nonpaged Bytes counter, 264

Poolmon.exe. *See* Memory Pool Monitor (Poolmon.exe)

pools
- Available Bytes pool, 66, 81–82
- investigating usage, 461–464
- Nonpaged pools, 272
- Paged pools, 272
- system pools, 458–461
- thread pooling. *See* thread pooling

predication, 542

preemptive scheduling
- priority queuing, 24
- threads, 46

prefetching, 73

primary indicators
- overview, 235
- processor utilization, 244

print servers, 303

priority queuing
- deferred procedure call (DPC), 56
- dynamic priority adjustments, 58–60
- interrupt priority, 55
- overview, 55
- preemptive scheduling, 24
- thread dispatching priorities, 56–58

Private Bytes counter, 271

privileged mode, 410

Privileged state, 39

% Privileged Time counter, 254

procedures
- analysis procedures, 404–406
- background performance monitoring, 314
- capacity planning, 316, 379
- counter log scenarios, 388–395
- daily server monitoring procedures. *See* daily server monitoring procedures
- management reporting, 315
- overview, 313
- SQL Server repositories, using, 365
- trending, 379
- troubleshooting counter collection problems, 395–400

process virtual address spaces
- extended user virtual addressing, 77
- overview, 74–75
- shared system addresses, 76

process working set, 85–86

process working set bytes, 262

process working set management
  accounting for process memory usage, 87
  shared DLLs, 87
  system working sets, 88
Process(instancename)\% Processor Time counter, 248
Process(n)\Elapsed Time counter, 242
Process(n)\Private Bytes counter, 271
Process(*)\Working Set counter, 262
processes
  monitoring using Task Manager, 157–161
  virtual memory allocations, 454–456
Processor Queue Length counter, 246
% Processor Time counter, 244, 248
Processor(_Total)\% Interrupt Time counter, 253
Processor(_Total)\% Privileged Time counter, 254
Processor(_Total)\% Processor Time counter, 244
processors
  64-bit architecture, 543–545
  accounting for processor usage, 51–54
  affinity, 60, 558
  asymmetric multiprocessing, 250
  base priorities, 57
  bottlenecks, 246, 407–409
  caches, 543
  context switches, 250
  Context Switches/sec counter, 251
  counter selection logic, 420–423
  decomposition, 409–413
  deferred procedure call (DPC), 56
  DPC mode, 411
  dynamic priority adjustments, 58–60
  dynamic range (1–15), 57
  hard affinity, 61
  hardware basics, 539–540
  identifying runaway processes by using counter logs, 416–430
  identifying runaway processes by using Task Manager, 413–415
  identifying use by module and function, 423–430
  Idle thread, 51
  instruction execution throughput, 539–545
  interrupt priority, 55
  interrupt processing, 48
  % Interrupt Time counter, 253
  involuntary waits, 49
  Kernrate report on usage, 424–430
  log file formats, 420–423
  measuring utilization, 243
  microarchitecture, 542
  multiprocessing, asymmetric, 250
  multiprocessing, symmetric, 250
  multiprocessors. *See* multiprocessors
  normalizing CPU time, 54
  normalizing utilization measures, 245
  Out-of-Order execution, 542
  overview, 43
  performance, advanced topics overview, 538
  pipelining, 541
  predication, 542
  priority queuing, 55–60
  privileged mode, 410
  % Privileged Time counter, 254
  Process(instancename)\% Processor Time counter, 248
  process-level CPU consumption, 248
  processor performance hardware mechanisms, 541–543
  Processor Queue Length counter, 246
  % Processor Time counter, 244, 248
  Processor(_Total)\% Interrupt Time counter, 253
  Processor(_Total)\% Privileged Time counter, 254
  Processor(_Total)\% Processor Time counter, 244
  Ready queue, 54
  real-time range (16–31), 57
  simultaneous multithreading (SMT), 543
  soft affinity, 61
  speculation, 542
  superscalar execution, 542
  symmetric multiprocessing, 250
  System Monitor overhead, 417–419
  System\Context Switches/sec counter, 251
  System\Processor Queue Length counter, 246
  thread dispatching priorities, 56–58
  threads, 43–48
  time-slicing, 50, 546–548
  transient threads and processes, 53
  troubleshooting bottlenecks, 246
  troubleshooting overview, 406
  use by module and function, 417
  use by process, 413
  use by state, 410–412
  user mode, 411–412
  utilization by processor, 250, 409
  utilization by processor and thread, 410
  utilization by processor execution state, 252–256
  utilization by processor rate, 409
  utilization management reports, 342–344
  utilization overview, 242
  voluntary waits, 49
ProcessorThreadMax property, 307
Processr.sys, 41
protocol stacks
  overview, 100–101
  packet processing, 101–102
  processing HTTP GET requests, 102
PTE. *See* Page Table entries (PTEs)
Pviewer.exe, 131

Q

queue depth limits, 32
queue length
  bottleneck analysis, 403
  processors, 407–409
Queue Length counter, 302
queue time
  arrival rate distribution, 30
  overview, 20
  queue depth limits, 32
  service time distribution, 30–31
  utilization, 29–33
queued spin locks, 559
queuing systems, 14–16

R

RAID. *See* Redundant array of independent disks (RAID)
RAID 0, 480
RAID 0/1, 481

RAID 1, 480
RAID 5, 481
RAM. *See* physical memory
Ready queue, 54
ready threads, 44
real memory. *See* physical memory
real-time range (16–31), 57
Redundant array of independent disks (RAID)
  comparing levels, 482–484
  disk mirroring, 480
  disk striping, 480
  dynamic volumes, 484
  overview, 479–487
  RAID 0, 480
  RAID 0/1, 481
  RAID 1, 480
  RAID 5, 481
  rotated parity sectors, 481
relative exceptions, 239
Relog.exe
  capacity planning databases, 348
  command syntax, 192
  consolidating performance data from multiple servers, 339–342
  counter log settings files, 348
  counter settings files, 340–342
  editing daily counter logs, 347
  filtering log files, 194
  formatting output files, 194
  logging local counters to local disks, 320
  managing performance logs, 191
  merging counter logs, 193
  overview, 130
  populating repositories, 353–354
  settings files, 195
  summarizing daily counter logs, 347
  summarizing log files, 196–200
  using, 192
remote logging
  local log files, 391
  remote log files, 392
Replmon.exe, 131
reports
  automated counter log processing, 362
  Daily Processor Utilization report, 342
  Disk Free Space report, 344
  event trace reports, 224
  File I/O report, 505
  Kernrate report on processor usage, 424–430
  Memory Allocation report, 645
  Network Interface Traffic report, 345
  Peak Hour Processor Utilization report, 343
  sample management reports, 342–345
repositories
  accessing counter log files, 375–377
  counter log database schemas, 369–373
  CounterData table, 370, 377–379
  CounterDetails table, 370–371
  DisplayToID table, 372–373
  populating, 353–354
  querying SQL performance databases (PDB), 373–379
  System Monitor, 366–369
  using, 365
resident pages, 263–266
resource utilization
  bottleneck analysis, 403
  processors, 407–409
response time
  Active Server Pages (ASP), 578–579
  Little's Law, 33–35
  overview, 20
  utilization, 33–35
retransmission timeout (RTO)
  calculating, 534
  Karn's algorithm, 534
  overview, 533
rights, 394
rotated parity sectors, 481
round robin
  overview, 24
  thread dispatching priority, 58
round trip time (RTT)
  congestion windows, 530–531
  effective capacity of WAN links, 529
  measuring using Tracert, 528–529
  overview, 527–528
router performance, 115
routing, 112
routing tables, 113
RTO. *See* retransmission timeout (RTO)
RTT. *See* round trip time (RTT)
running threads, 44

S

SAS, 619
scalability
  bottlenecks, 10–12
  ideal vs. actual application scalability, 9
  multiprocessing, 551–562
  overview, 8
  Utilization Law, 10
Scheduled Tasks service, 362
secondary indicators, 235
security, 128–133
Segments Received/sec counter, 295
server applications. *See* applications
Server Performance Advisor (SPA), 516–517
Server Work Queues(n)\Available Threads counter, 301
Server Work Queues(n)\Queue Length counter, 302
Server\Paged Pool Failures counter, 275
Server\Work Item Shortages counter, 299
service time, 19
service time distribution, 30–32
sessions, 120
settings files
  capacity planning databases, 348
  counter logs, 348
  counters. *See* counter settings files
  Log Manager, 182
  Relog.exe, 195
shared memory multiprocessors
  overview, 549
  single system images, 550
  symmetric multiprocessing, 551
Showperf.exe, 133
simultaneous multithreading (SMT), 543, 602
single system images
  overview, 550
  symmetric multiprocessing, 551
sliding windows, 121
Smlogsvc.exe. *See* Performance Logs and Alerts
SMT (simultaneous multithreading), 543, 602
snooping, 555
soft affinity, 61
SPA (Server Performance Advisor), 516–517
speculation, 542

spin locks
  overview, 554
  queued spin locks, 559
Splinfo.exe, 133
split I/Os, 290
Split IO/sec counter, 290
Spoolsv.exe, 303
SQL Server
  32-bit extended virtual addressing, 618
  accessing counter log files, 375–377
  extended virtual addressing in 32-bit machines, 618
  historical data for capacity planning, 347
  performance database (PDB), 350–353, 373–379
  repositories. *See* repositories
  scaling sideways, 618
  thread pooling, 568–570
Srvinfo.exe, 133
starvation, 24
statistical tools. *See* tools
storage controllers
  disk troubleshooting, 477–479
  RAID, 479–487
superscalar execution, 542
symmetric multiprocessing, 250, 551
system architecture
  disk architecture, 92–95
  disk performance expectations, 91
  disk performance measurements, 95–97
  Executive, 41
  hardware abstraction layer (HAL), 41
  I/O subsystems, 89–97
  memory. *See* memory and paging
  network interfaces. *See* network interfaces
  operating systems, 38–42
  overview, 37
  paging. *See* memory and paging
  Performance Monitor, 37
  processors. *See* processors
  Processr.sys, 41
  Win32k.sys, 42
System Cache Resident Bytes counter, 277
system file cache, 272
System Monitor
  ActiveX control. *See* ActiveX control
  automation interface. *See* System Monitor Automation Interface
  chart viewing in real time, 145–146
  Chart Views, 151
  compared to Task Manager, 158, 162–167
  counter logs, 176
  custom monitoring configuration creation, 147
  data view customization, 149–151
  Legend sorting, 153
  overhead, 417–419
  overview, 130, 144
  performance, advanced topics overview, 628
  printing displays, 153
  properties, 150
  real-time data, saving, 148
  sampling intervals, changing, 146
  shortcut menu commands, 149
  SQL Server repositories, 366–369
  tips for working with, 151–154
  toolbar buttons, 148
  y-axis scaling, 152
System Monitor Automation Interface
  ActiveX control CollectSample method, 634
  ActiveX control event methods, 635
  ActiveX control OnCounterDeleted method, 636
  ActiveX control OnSampleCollected method, 635
  adding ActiveX control to Web pages, 629
  adding performance counters to ActiveX control, 639
  configuring ActiveX control appearance, 636
  configuring ActiveX control color schemes, 637
  configuring ActiveX control display types, 632
  configuring ActiveX control font styles, 638
  configuring ActiveX control performance counters, 640
  configuring ActiveX control sampling rates, 634
  configuring System Monitor user interface, 637
  creating Web pages for monitoring performance, 643
  customizing ActiveX control, 630
  drag-and-drop support, 645
  manually retrieving performance data, 634
  Memory Allocation report, 645
  overview, 628
  removing performance counters from ActiveX control, 641
  sample commands for configuring ActiveX control, 631
  tracking performance counters using counter paths, 642
system pools
  32-bit virtual memory addressing limits, 458–461
  extended virtual addressing in 32-bit machines, 620
System Up Time counter, 241
system virtual memory
  32-bit virtual memory addressing limits, 457
  shortages, 620
system working sets, 88
System\Context Switches/sec counter, 251
Systeminfo.exe, 130
System\Processor Queue Length counter, 246
System\System Up Time counter, 241

T

Task Manager
  compared to System Monitor, 158, 162–167
  identifying runaway processes, 413–415
  monitoring applications, 156
  monitoring networks, 163–167
  monitoring performance, 161–163
  monitoring processes, 157–161
  monitoring users, 167
  overview, 130, 154
  performance monitoring, 161–163
  starting, 155
  working with, 155–156
Taskkill.exe, 130
Tasklist.exe, 130

TCP. *See* Transmission Control Protocol (TCP)
TCP/IP, 294–296
TCPvn\Connections Established counter, 295
TCPvn\Segments Received/sec counter, 295
Terminal Services, 310
Terminal services\Total sessions counter, 310
thrashing, 68
thread pooling
- ASP response time measurements, 578–579
- ASP.NET, 581–583
- concurrency levels, 586
- File Server service, 570–573
- Internet Information Services (IIS), 573–578
- lock collisions, 584
- max threads, 585
- overview, 565–568
- server applications, 583
- SQL Server, 568–570

threads
- base priorities, 57
- context switches, 44, 250
- dispatching priorities, 56–58
- dynamic priority adjustments, 58–60
- dynamic range (1–15), 57
- hard affinity, 61
- ideal processors, 61
- Idle thread, 51
- interrupt processing, 48
- involuntary waits, 49
- maximum number of, 585
- multithreading, 45
- normalizing CPU time, 54
- overview, 43
- pooling. *See* thread pooling
- preemptive scheduling, 46
- processor Ready queue, 54
- ready threads, 44
- real-time range (16–31), 57
- running threads, 44
- soft affinity, 61
- thread states, 46
- time-slicing, 50
- transient threads and processes, 53
- upper limit, 585
- voluntary waits, 49
- waiting threads, 44

throughput
- overview, 16
- scalability, 9

thunking, 545
time-slicing, 50, 546–548
tools
- Adlb.exe, 132
- administrative controls, 128
- Checkrepl.vbs, 132
- Clearmem.exe, 132
- Consume.exe, 132
- counter logs. *See* counter logs
- Custreasonedit.exe, 132
- Depends.exe, 131
- Devcon.exe, 131
- DH.exe, 132
- Diruse.exe, 131
- Disk Defragmenter, 499–504
- Empty.exe, 132
- event traces. *See* event traces
- Event Tracing for Windows (ETW). *See* Event Tracing for Windows (ETW)
- Extensible Counter List (Exctrlst.exe). *See* Extensible Counter List (Exctrlst.exe)
- Freedisk.exe, 129
- Health_chk.cmd, 131
- Interrupt-Affinity Filter (Intfiltr.exe). *See* Interrupt-Affinity Filter (Intfiltr.exe)
- Kernrate.exe. *See* Kernrate.exe
- load generating and testing, 128
- Lodctr.exe, 129
- Log Manager (Logman.exe). *See* Log Manager (Logman.exe)
- Memory Pool Monitor (Poolmon.exe). *See* Memory Pool Monitor (Poolmon.exe)
- memory reporting tools, 443–448
- Memsnap.exe, 131
- Memtriage.exe, 132
- Msinfo32.exe, 130
- Netcap.exe, 131
- Network Monitor. *See* Network Monitor
- operating system performance-related tools, 129
- overview, 125
- Page Fault Monitor (Pfmon.exe). *See* Page Fault Monitor (Pfmon.exe)
- Performance Logs and Alerts. *See* Performance Logs and Alerts
- performance statistics overview, 126
- performance-related support tools, 131
- Pfmon.exe. *See* Page Fault Monitor (Pfmon.exe)
- Pmon.exe, 132
- Poolmon.exe. *See* Memory Pool Monitor (Poolmon.exe)
- Pviewer.exe, 131
- Relog.exe. *See* Relog.exe
- Replmon.exe, 131
- security required for usage, 128–133
- SPA system network diagnosis, 516–517
- Showperf.exe, 133
- Smlogsvc.exe. *See* Performance Logs and Alerts
- Splinfo.exe, 133
- Srvinfo.exe, 133
- summary of, 126–133
- System Monitor. *See* System Monitor
- Systeminfo.exe, 130
- Task Manager. *See* Task Manager
- Taskkill.exe, 130
- Tasklist.exe, 130
- Trace Report (Tracerpt.exe). *See* Trace Report (Tracerpt.exe)
- Tracert, 528–529
- TSSCalling.exe, 133
- Typeperf.exe. *See* Typeperf.exe
- Unlodctr.exe, 130
- Vadump, 133
- Volperf.dll, 133
- Windows Resource Kit performance–related tools, 132

Total sessions counter, 310
TPC (Transaction Performance Council) performance benchmarks, 561
trace logs
- Active Directory, 221
- configuring, 214–217
- event trace reports, 224
- Internet Information Services (IIS), 221
- Log Manager (Logman.exe), 217–224
- properties, 216
- trace event providers, 215
- trace providers, 219–223

Trace Report (Tracerpt.exe)
  event trace reports, 224
  file-level access patterns, 505
  overview, 130
Tracert, 528–529
Transaction Performance Council (TPC) performance benchmarks, 561
transition faults, 266, 267
Transition Faults/sec counter, 266
Transmission Control Protocol (TCP)
  byte sequence numbers, 120
  congestion control, 522–524
  congestion windows, 123
  overview, 118
  session connections, 120
  sliding windows, 121
trending, 379
triage, 240
troubleshooting
  analysis procedures, 404–406
  baseline data, 402
  bottleneck analysis, 402–404
  counter collection problems, 395–400
  current performance levels, 403
  decomposition, 404, 409–413
  disabled performance counters, 398
  disks. *See* disk troubleshooting
  identifying runaway processes by using counter logs, 416–430
  identifying runaway processes by using Task Manager, 413–415
  memory. *See* memory and paging troubleshooting
  missing performance counters, 396–399
  networking error conditions, 297
  networks. *See* network troubleshooting
  overview, 401
  paging. *See* memory and paging troubleshooting
  Perflib error messages, 396–398
  performance problems, 5–8
  processor bottlenecks, 246, 407–409
  processors overview, 406
  queue length, 403, 407–409
  resource utilization, 403, 407–409
  restoring corrupt performance counters, 400
TSSCalling.exe, 133
Typeperf.exe
  automating usage, 204–206
  command syntax, 200
  data formats of performance logs, 206
  monitoring multiple counters, 204
  monitoring performance from the command line, 203
  monitoring performance of remote computers, 204
  obtaining lists of performance counters, 201
  overview, 130, 200
  retrieving counters from remote computers, 203
  sample output, 203
  sampling intervals, 205
  sampling rates, 205
  writing output to counter logs, 205

U

unfair scheduling, 24
Unlodctr.exe, 130
user mode, 411–412
User state, 39
users, monitoring using Task Manager, 167
utilities. *See* tools
utilization
  arrival rate distribution, 30
  forecasting linear growth, 17
  Little's Law, 33–35
  overview, 17–19
  queue depth limits, 32
  queue time, 29–33
  response time, 33–35
  service time distribution, 30–31
Utilization Law
  overview, 28
  scalability, 10

V

VADs (Virtual Address Descriptors), 87
Vadump, 133
VBScript
  automated counter log processing, 355–358
  constructing file names, 321
  daily file management and cleanup, 319
  executing Log Manager (Logman.exe), 321
  identifying source machines, 321
Virtual Address Descriptors (VADs), 87
virtual addressing
  extended in 32-bit machines. *See* extended virtual addressing in 32-bit machines
  extended user, 77
  overview, 62
  Page Table entries (PTEs), 78
  process virtual address spaces, 74–77
  shared system addresses, 76
virtual disks, 477–479, 487
virtual memory. *See also* memory and paging
  32-bit addressing limits, 456–464
  64-bit virtual memory, 621
  Address Window Extension (AWE), 277
  addressing constraints, 609
  Available Bytes counter, 260–262
  Cache Bytes counter, 263
  cache miss, 277
  Commit Limit, 452
  Committed Bytes counter, 269
  forecasting memory requirements, 622–628
  Free System Page Table Entries counter, 274
  lazy writes, 277
  leaks, 451
  memory contention index, 624–626
  Memory\Available Bytes counter, 260–262
  Memory\Cache Bytes counter, 263
  Memory\Committed Bytes counter, 269
  Memory\Free System Page Table Entries counter, 274
  Memory\Nonpaged Pool Bytes counter, 273
  Memory\Page Faults/sec counter, 267
  Memory\Paged Pool Bytes counter, 273
  Memory\Pages/sec counter, 258
  Memory\Pool Nonpaged Bytes counter, 264
  memory-resident disk caches, 276–279

virtual memory, *continued*
  Memory\System Cache Resident Bytes counter, 277
  Memory\Transition Faults/sec counter, 266
  Nonpaged Pool Bytes counter, 273
  Nonpaged pools, 272
  overview, 62, 256
  page fault resolution, 65–69
  Page Faults/sec counter, 267
  Page Tables, 63–65
  Paged Pool Bytes counter, 273
  Paged pool contention, 626–628
  Paged Pool Failures counter, 275
  Paged pools, 272
  Pages/sec counter, 258
  Physical Address Extension (PAE), 277
  physical memory usage, 260
  Pool Nonpaged Bytes counter, 264
  Private Bytes counter, 271
  process allocations, 454–456
  process working set bytes, 262
  Process(*)\Working Set counter, 262
  Process(n)\Private Bytes counter, 271
  resident pages in system range, 263–266
  Server\Paged Pool Failures counter, 275
  shortages, 448–456
  System Cache Resident Bytes counter, 277
  system file cache, 272
  system range, 272
  transition faults, 266
  Transition Faults/sec counter, 266
  usage, 269
  virtual addressing, 62
  V:R ratio, 624–626
  Working Set counter, 262
Visual Basic, Scripting Edition. *See* VBScript
Volperf.dll, 133
voluntary waits, 49
V:R ratio, 624–626

W

waiting threads, 44
WAN. *See* wide area network (WAN)
Web-based applications
  Active Server Pages\Request Execution Time counter, 305
  Active Server Pages\Request Queue Time counter, 306
  Active Server Pages\Requests Executing counter, 307
  Active Server Pages\Requests Queued counter, 308
  ASP.NET\Request Execution Time counter, 305
  ASP.NET\Request Queue Time counter, 306
  ASP.NET\Requests Executing counter, 307
  ASP.NET\Requests Queued counter, 308
  IIS critical measurements, 303
  overview, 303
wide area network (WAN)
  adaptive retransmission, 532–535
  congestion windows, 530–531
  effective capacity of WAN links, 529
  performance overview, 526
  round trip time (RTT), 527–531
Win32k.sys, 42
Windows On Windows (WOW), 544
Windows Script Host (WSH)
  automated counter log processing, 355–358
  constructing file names, 321
  daily file management and cleanup, 319
  executing Log Manager (Logman.exe), 321
  identifying source machines, 321
  managing Log Manager data collections, 190
Windows System Resource Manager (WSRM)
  asymmetric partitioning, 595–599
  overview, 233
Windows Task Manager. *See* Task Manager
Windows Time Service, 320
Work Item Shortages counter, 299
Working Set counter, 262
working sets, 262
workload parallelism, 552
WOW (Windows On Windows), 544
WSH. *See* Windows Script Host (WSH)
WSRM. *See* Windows System Resource Manager (WSRM)