CSE 451: Operating Systems
Spring 2006

Module 1
Course Introduction

John Zahorjan
zahorjan@cs.washington.edu
534 Allen Center

Today’s agenda

• Administrivia
  – course overview
  – course staff
  – general structure
  – the text
  – policies
  – your to-do list
• course registration
• OS overview
  – functional
  – resource management, etc.
  – historical
  • batch systems, multiprogramming, timeshared OS’s
  • PCs, networked computers, p2p, embedded systems

Course overview

• Everything you need to know will be on the course web page:
  http://www.cs.washington.edu/451/

• But to tide you over for the next hour …
  – course staff
    • John Zahorjan
    • Kurtis Heimerl
    • Yongchul Kwon
  – general structure
    • read the text prior to class
    • class will supplement rather than regurgitate the text
    • homework exercises provide added impetus to keep up with the reading
    • sections will focus on the project (5 components)
    • we really want to encourage discussion, both in class and in section

– the text
  • Silberschatz, Galvin & Gagne, Operating System Concepts, seventh edition
    – if using an earlier edition, watch chapter numbering, exercise numbering
– other resources
  • many online; some of them are required reading; some of them are prohibited reading
– policies
  • collaboration vs. cheating
  • homework exercises
  • late policy

– your to-do list …
  • please read the entire course web thoroughly, today
  • (if you haven’t already received a post to the cse451 mailing list) please get yourself on the cse451 email list, today, and check your email daily
  • keep up with the reading
  • homework 1 (problems) is posted on the web now
    – due at the start of class on Monday
  • project 0 is posted on the web now;
    – due at 1:00 pm next Tuesday (but if you don’t get started this week you’ll be in trouble)
    – C/programming help/intro session tomorrow
  • project0 skeleton code discussed in section on Thursday
What is an Operating System?

• The text:
  – “an intermediary between the user of a computer and the computer hardware”
  – “manages the computer hardware”
  – “each [piece] should be … well delineated …, with carefully defined inputs, outputs, and functions”
  – “an amazing aspect of operating systems is how varied they are in accomplishing these tasks … mainframe operating systems … personal computer operating systems … operating systems for handheld computers …”
  – “in 1998, the United States Department of Justice filed suit against Microsoft, in essence claiming that Microsoft included too much functionality in its operating system … for example, a web browser was an integral part of the operating system”

What is Windows?

What is .NET?
The OS and hardware

- **An OS mediates** programs’ access to hardware resources
  - Computation (CPU)
  - Volatile storage (memory) and persistent storage (disk, etc.)
  - Network communications (TCP/IP stacks, Ethernet cards, etc.)
  - Input/output devices (keyboard, display, sound card, etc.)
- **The OS abstracts** hardware into logical resources and well-defined interfaces to those resources
  - processes (CPU, memory)
  - files (disk)
  - sockets (network)

Why bother with an OS?

- **Application benefits**
  - **programming simplicity**
  - see high-level abstractions (files) instead of low-level hardware details (device registers)
  - abstractions are reusable across many programs
  - **portability** (across machine configurations or architectures)
  - device independence: 3Com card or Intel card?
- **User benefits**
  - **safety**
  - program “sees” own virtual machine, thinks it owns computer
  - **OS protects** programs from each other
  - **OS fairly multiplexes** resources across programs
  - **efficiency** (cost and speed)
  - share one computer across many users
  - concurrent execution of multiple programs

The major OS issues

- **structure**: how is the OS organized?
- **sharing**: how are resources shared across users?
- **naming**: how are resources named (by users or programs)?
- **security**: how is the integrity of the OS and its resources ensured?
- **protection**: how is one user/program protected from another?
- **performance**: how do we make it all go fast?
- **reliability**: what happens if something goes wrong (either with hardware or with a program)?
- **extensibility**: can we add new features?
- **flexibility**: are we in the way of new apps?
- **communication**: how do programs exchange information, including across a network?

More OS issues...

- **concurrency**: how are parallel activities (computation and I/O) created and controlled?
- **scale**: what happens as demands or resources increase?
- **persistence**: how do you make data last longer than program executions?
- **distribution**: how do multiple computers interact with each other?
- **accounting**: how do we keep track of resource usage, and perhaps charge for it?

There are tradeoffs, not right and wrong.

Progression of concepts and form factors

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mainframe</td>
<td>IBM 1401</td>
<td>IBM 360</td>
<td>IBM 3033</td>
<td>IBM 3081</td>
</tr>
<tr>
<td>Microcomputer</td>
<td>DEC PDP-8</td>
<td>DEC VAX</td>
<td>Compaq</td>
<td>IBM RISC System/6000</td>
</tr>
<tr>
<td>Handheld computer</td>
<td>IBM 5100</td>
<td>IBM 2100</td>
<td>IBM 6100</td>
<td>IBM 4761</td>
</tr>
</tbody>
</table>
```

Multiple trends at work

- “Ontogeny recapitulates phylogeny”
  - Ernst Haeckel (1834-1919)
  - (“always quotable, even when wrong”)
- “Those who cannot remember the past are condemned to repeat it”
  - George Santayana (1863-1952)
- But new problems arise, and old problems re-define themselves
  - The evolution of PCs recapitulated the evolution of minicomputers, which had recapitulated the evolution of mainframes
  - But the ubiquity of PCs re-defined the issues in protection and security
### Protection and security as an example

- none
- OS from my program
- your program from my program
- my program from my program
- access by intruding individuals
- access by intruding programs
- denial of service
- distributed denial of service
- spoofing
- spam
- worms
- viruses
- stuff you download and run knowingly (bugs, trojan horses)
- stuff you download and run unknowingly (cookies, spyware)

### OS history

- In the very beginning...
  - OS was just a library of code that you linked into your program; programs were loaded in their entirety into memory, and executed
  - interfaces were literally switches and blinking lights
- And then came batch systems
  - OS was stored in a portion of primary memory
  - OS loaded the next job into memory from the card reader
  - job gets executed
  - output is printed, including a dump of memory
  - repeat...
  - card readers and line printers were very slow
  - so CPU was idle much of the time (wastes $$)

### Spooling

- Disks were much faster than card readers and printers
- Spool (Simultaneous Peripheral Operations On-Line)
  - while one job is executing, spool next job from card reader onto disk
  - slow card reader I/O is overlapped with CPU
  - can even spool multiple programs onto disk
  - OS must choose which to run next
  - job scheduling
  - but, CPU still idle when a program interacts with a peripheral during execution
  - buffering, double-buffering

### Multiprogramming

- To increase system utilization, multiprogramming OSs were invented
  - keeps multiple runnable jobs loaded in memory at once
  - overlaps I/O of a job with computing of another
  - while one job waits for I/O completion, OS runs instructions from another job
  - to benefit, need asynchronous I/O devices
  - need some way to know when devices are done
    - interrupts
    - polling
  - goal: optimize system throughput
    - perhaps at the cost of response time...

### Timesharing

- To support interactive use, create a timesharing OS:
  - multiple terminals into one machine
  - each user has illusion of entire machine to him/herself
  - optimize response time, perhaps at the cost of throughput
- Timeslicing
  - divide CPU “equally” among the users
  - if job is truly interactive (e.g., editor), then can jump between programs and users faster than users can generate load
  - permits users to interactively view, edit, debug running programs (why does this matter?)

- MIT CTSS system (operational 1961) was among the first timesharing systems
  - only one user memory-resident at a time (32KB memory!)
- MIT Multics system (operational 1968) was the first large timeshared system
  - nearly all OS concepts can be traced back to Multics!
  - “second system syndrome”
Parallel OS

- Some applications can be written as multiple parallel threads or processes.
  - Can speed up the execution by running multiple threads/processes simultaneously on multiple CPUs.
  - Need OS and language primitives for dividing programs into multiple parallel activities.
  - Need OS primitives for fast communication among activities.
    - Degree of speedup dictated by communication/computation ratio.
  - Many flavors of parallel computers today:
    - SMPs (symmetric multi-processors)
    - MPPs (massively parallel processors)
    - NOWs (networks of workstations)
    - Computational grid (SETI@home)

Personal Computing

- Primary goal was to enable new kinds of applications.
- Bit mapped display [Xerox Alto 1972]
  - New classes of applications
  - New input device (the mouse)
- Move computing near the display
  - Why?
- Window systems
  - The display as a managed resource.
- Local area networks [Ethernet]
  - Why?

Distributed OS

- Distributed systems to facilitate use of geographically distributed resources.
  - Workstations on a LAN
  - Servers across the Internet
- Supports communications between programs.
  - Interprocess communication
    - Message passing, shared memory
    - Networking stacks
- Sharing of distributed resources (hardware, software)
  - Load balancing, authentication and access control, ...
- Speedup isn’t the issue.
  - Access to diversity of resources is goal.

Client/Server computing

- Mail server/service
- File server/service
- Print server/service
- Compute server/service
- Game server/service
- Music server/service
- Web server/service
- Etc.

Peer-to-Peer (p2p) systems

- Napster
- Gnutella
  - Example technical challenge: self-organizing overlay network
  - Technical advantage of Gnutella?
  - Or … legal advantage of Gnutella?
Embedded/Mobile/Pervasive computing

- Pervasive computing
  - cheap processors embedded everywhere
  - how many are on your body now? in your car?
  - cell phones, PDAs, network computers, …
- Typically very constrained hardware resources
  - slow processors
  - very small amount of memory (e.g., 8 MB)
  - no disk
  - typically oriented towards one application
  - limited power
- But this is changing rapidly!

CSE 451

- In this class we will learn:
  - what are the major components of most OS’s?
  - how are the components structured?
  - what are the most important (common?) interfaces?
  - what policies are typically used in an OS?
  - what algorithms are used to implement policies?
- Philosophy
  - you may not ever build an OS
  - but as a computer scientist or computer engineer you need to understand the foundations
  - most importantly, operating systems exemplify the sorts of engineering design tradeoffs that you’ll need to make throughout your careers – compromises among and within cost, performance, functionality, complexity, schedule …