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, time shared OS’s
    - PCs, networked computers, p2p

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
    - Ed Lazowska
    - Ilya Maykov
    - Dave Richardson
  - 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
  - policies
    - collaboration vs. cheating
    - homework exercises
    - late policy

- your to-do list …
  - please read the entire course web thoroughly, today
  - please get yourself on the cse451 email list, today, and check your email daily
  - homework 1 (reading + problems) is posted on the web now; reading due Wednesday, problems due at the start of class on Friday
  - project 0 will be posted on the web imminently; will be discussed in section on Thursday; due at the start of class next Wednesday (but if you don’t get started this week you’ll be in trouble)
Course registration

- If you’re going to drop this course
  - please do it soon!
- If you want to get into this course
  - plan for the worst case (we’re over our limit of 60 currently)
  - but, make sure you’ve filed a petition with the advisors

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 an Operating System?

- An operating system (OS) is:
  - a software layer to abstract away and manage details of hardware resources
  - a set of utilities to simplify application development
  - “all the code you didn’t write” in order to implement your application

What is Windows?

- Applications
- OS
- Hardware

What is .NET?

- Application
- Browser
- TCP/IP
- COM
- Installer
- Printing
- Windows
- Internet
What is .NET?

- Application
- Extenability
- Asynchrony
- Device independence
- XML
- Identity & Security
- .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
- User benefits
  - safety
  - program “sees” own virtual machine, thinks it owns computer
  - OS protects programs from each other
  - OS fairly multiplexes resources across programs
- Device independence: 3Com card or Intel card?

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?
- 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?
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 (why?)
  - 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 Multics system (mid-1960’s) was the first large timeshared system
  - nearly all OS concepts can be traced back to Multics
**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

**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 program into multiple parallel activities
- need OS primitives for fast communication between activities
  - degree of speedup dictated by communication/computation ratio
- many flavors of parallel computers
  - SMPs (symmetric multi-processors)
  - MPPs (massively parallel processors)
  - NOVUs (networks of workstations)
  - computational grid (SETI@home)

**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 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 only one dedicated application
  - limited power

**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 …