CSE 451: Operating Systems  
Autumn 2009  

Module 1  
Course Introduction  

Ed Lazowska  
lazowska@cs.washington.edu  
570 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 is on the course web page:
  
  http://www.cs.washington.edu/451/

- But to tide you over for the next hour …  
  - course staff  
    - Ed Lazowska  
    - Slava Chernyak  
    - Owen Kim  
  - 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 (several separate components)  
    - we really want to encourage discussion, both in class and in section

- 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  
  - keep up with the reading  
  - homework 1 (reading + problems) is posted on the web now  
    - reading due Friday  
    - problems due at the start of class next Wednesday  
  - project 0 is posted on the web now  
    - will be discussed in section on Thursday  
    - due next Friday (but if you don’t get started this weekend 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 at our limit currently)
  - but, make sure you've filed a petition with the advisors
    - they run the show!
  - give things a few days to settle down

More about 451

- This is really (at least!) two classes:
  - A classroom/textbook part (mainly run by me)
  - A project part (mainly run by the TAs)
- In a perfect world, we would do this as a two-quarter sequence
- The world isn't perfect 😊
- Sometimes the projects and the lectures will mesh, sometimes they won't
- But in any case, you will have to keep up with both the classroom and the projects
- There will be a lot of work
- But you will learn a lot
- In the end, you'll understand much more deeply how computers work
  - "There is no magic"

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?

- 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
  - Key idea: virtualization or abstraction of resources
The OS and hardware

- An OS *mediates* programs’ access to hardware resources (*sharing and protection*)
  - 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 (*ease of use*)
  - processes (CPU, memory)
  - files (disk)
  - programs (sequences of instructions)
  - 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?
- 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!*
Hardware/Software Changes with Time

• 1960s: mainframe computers (IBM)
• 1970s: minicomputers (DEC)
• 1980s: microprocessors and workstations (SUN)
• 1990s: PCs (rise of Microsoft, Intel, then Dell)
• 2000s:
  – Internet Services / Clusters (Amazon)
  – General Cloud Computing (Google, Amazon)
  – Mobile/ubiquitous/embedded computing
• ….
• 2020: it’s up to you!!

Progression of concepts and form factors

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)

Has it all been discovered?

• New challenges constantly arise
  – embedded computing (e.g., iPod)
  – sensor networks (very low power, memory, etc.)
  – peer-to-peer systems
  – ad hoc networking
  – scalable server farm design and management (e.g., Google)
  – software for utilizing huge clusters (e.g., MapReduce, Bigtable)
  – overlay networks (e.g., PlanetLab)
  – worm fingerprinting
  – finding bugs in system code (e.g., model checking)
• Old problems constantly 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 obliviously (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”
- CTSS as an illustration of architectural and OS functionality requirements

- In early 1980s, a single timeshared VAX/780 (like the one in the Allen Center atrium) ran computing for the entire CSE department.
- A typical VAX/780 was 1 MIPS (1 MHz) and had 1MB of RAM and 100MB of disk.
- An iPhone is 400 MIPS, has 128MB of RAM (way too little though) and 8GB of disk.

**Parallel systems**

- 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 [Burroughs D825, 1962]
  - need OS and language primitives for dividing program 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, 1973]
  - 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?
- Effect on OS?

**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?
  - er … 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 only one dedicated application
  - limited power
- But this is changing rapidly!

Ad hoc networking

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 …
  - you will love this course!