UMassAmherst

CMPSCI 377
Operating Systems
Introduction

Sean Barker

University of Massachusetts Amherst

Today's Class

- Organizational meeting
  - Course organization & outline
  - Policies
  - Prerequisites & course sign-up
- Intro to Operating Systems

Organizational Information

- Course web page
  - Visit cs.umass.edu/~sbarker/377
  - 377 Moodle: to turn in assignments, discussion forums, etc.
- Contact info
  - sbarker@cs.umass.edu
- TA: Prateek Sharma prateeks@cs.umass.edu
- Grading staff: Armand Halbert
- Discussion section

Prerequisites and Syllabus

- CMPSCI 230: Computer Systems Principles
- Textbook: Operating System Concepts (Silberschatz, Galvin, Gagne) 8th or 9th ed.
- Modern Operating Sys, 3rd ed., Tanenbaum
- Course requirements
  - 4-6 written homeworks (15%)
  - Class participation & in-class exercises (5%)
  - 3-4 programming projects (40%)
  - 3 exams (40%) – two in-class midterms and a final
- Strict late policies and policies on cheating

Course Resources

- Accounts in the EdLab: 30+ Linux-based machines
- Discussion section to help you with Lab assignments and course concepts
- Office hours:
  - Instructor: TuThu 12:30-2, CS 214 or by appt
  - TA: Prateek Sharma
  - Office hrs: MonFri 10:30-12, CS 214

Course Requirements

- Note: Percentages are subject to revision.
- Programming projects: 40%
  - Strict late policy!
- In-class exams: 40%
- Homeworks: 15%
- Class participation and discussions: 5%

Labs

- 3-4 Projects
  - Focus on topics covered in the class
- Projects will use primarily Java
- May work in groups of 2
- Computer Lab accounts or use your own machine
  - Grading on EdLab machines
- 3 late days

Plagiarism

- HW #0: sign Plagiarism policy on Moodle
- Cheating includes:
  - "Borrowing" solutions or code from someone
    - This includes reading previous solutions
  - Giving code to someone (even next year)
  - Googling for solutions or copying code from anyone (including the net)
  - Hiring someone to write your code
  - Submitting someone else's code as your own
  - Looking at anyone else's code

Cell Phone and Laptop Policy

- Class use policy: Don’t!
- Cell phones should be off or silenced
- Texting is strictly prohibited in class
- Laptops and tablets may NOT be used in class: No email, browsing, Facebook, Twitter during class lectures
- Violations may result in penalties

Course Outline & Topics

- Processes and Threads
- Memory Management
- Storage and File Systems
- Distributed Systems

What's An Operating System?

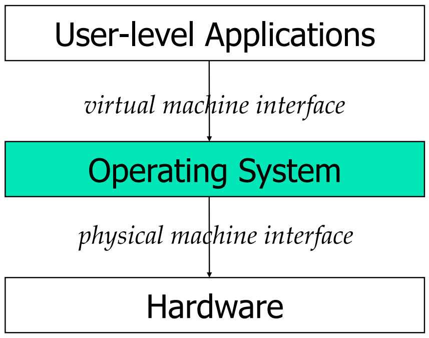
- Definition has changed over years
  - Originally, very bare bones
  - Now, includes more and more
- Operating System (OS)
  - Interface between the user and the architecture
  - Implements a virtual machine that is (hopefully) easier to program than raw hardware.

What's an OS? Bill Gates says…

"even
a ham sandwich"
(Steve B.)

OS: Traditional View

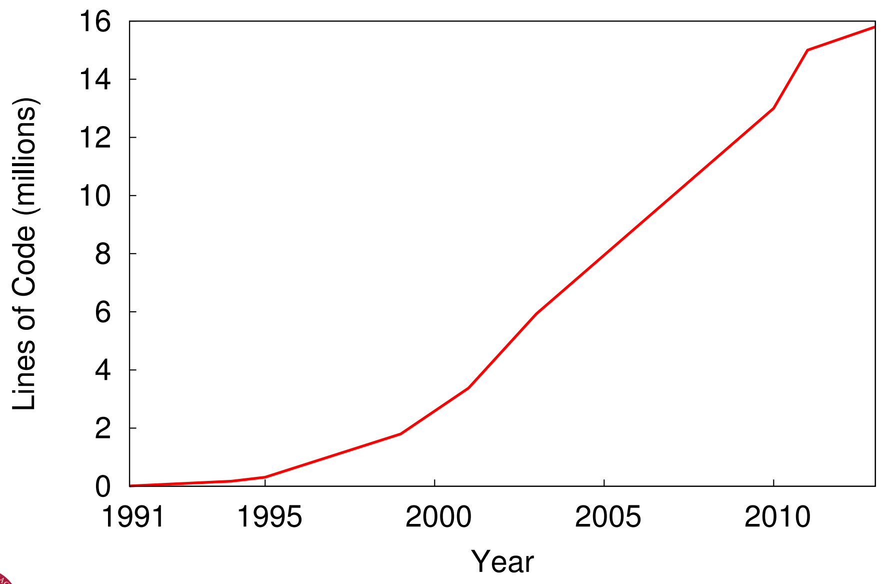
- Interface between user and architecture
  - Hides architectural details
- Implements virtual machine:
  - Easier to program than raw hardware
- Illusionist
  - Bigger, faster, reliable
- Government
  - Divides resources
  - “Taxes” = overhead



New Developments in OS

- Operating systems: active field of research
  - Demands on OS’s growing
  - New application spaces (Web, Grid, Cloud)
  - Rapidly evolving hardware
- Advent of open-source operating systems
  - Linux etc.
  - You can contribute to and develop OS’s!
  - Excellent research platform

Linux Kernel Size



Operating Sys: Salient Features

- **Services:** The OS provides standard services (the interface) which the hardware implements.
  - Examples: the file system, virtual memory, networking, CPU scheduling, and time-sharing
- **Coordination:** The OS coordinates multiple applications and users to achieve fairness and efficiency (throughput).
  - Examples: concurrency, memory protection, networking, and security.
- **Goal:** Design an OS so that the machine is convenient to use (a software engineering problem) and efficient (a system and engineering problem).

Why Study Operating Systems?

- **Abstraction:** How to get the OS to give users an illusion of infinite memory, CPUs, resources, world wide computing, etc.
- **System Design:** How to make tradeoffs between
  - performance and the convenience of OS abstractions,
  - performance and the simplicity of OS design, and
  - putting functionality in hardware or software.
- **Basic Understanding:** The OS provides the services that allow application programs to work at all.
- **System Intersection Point:** The OS is the point where hardware and software meet.

Why Study Operating Systems?

Not many operating systems are under development, so you are unlikely to get a job building an OS. However, understanding operating systems will enable you to use your computer more effectively. They also serve as an excellent example of system design issues whose results and ideas you will apply elsewhere.

- **Background:** To understand this course you must have a solid basic understanding of hardware (CPU instruction sets, memory hierarchies, I/O systems, etc.) and solid programming skills (complex data structures, classes as an encapsulation mechanism, etc.)
  - Obviously, you cannot understand the implications of how components intersect without understanding the components.

Build Large Computer Systems

- OS as an example of large system design
- Goals: Fast, reliable, large scale
- To build these systems, you need to know
  - Each computer:
    - Architectural details that matter
    - C and C++ (nitty gritty & more)
    - Memory management & locality
    - Concurrency & scheduling
    - Disks, network, file systems
  - Across cluster:
    - Server architectures
    - Distributed computing, file systems

History of Operating Systems

- And now, for some historical context
  - From mainframes to web-based systems in nine slides

1. Single-User Computers

- Hardware: expensive; humans: cheap
- One user at a time on console
  - Interacting with as program runs
- Computer executes one function at a time
  - No overlap: computation & I/O
- User must be at console to debug

- Multiple users = inefficient use of machine

2. Batch Processing

- Execute multiple "jobs" in batch:
  - Load program
  - Run
  - Print results, dump machine state
  - Repeat
- Users submit jobs (on cards or tape)
- Human schedules jobs
- Operating system loads & runs jobs

- More efficient use of machine

3. Overlap I/O and Computation

- Before: machine waits for I/O to complete
- New approach:
  - Allow CPU to execute while waiting
  - Add buffering
    - Data fills "buffer" and then output
  - and interrupt handling
    - I/O events trigger a signal ("interrupt")
- More efficient use of machine
  - still one job at a time

4. Multiprogramming

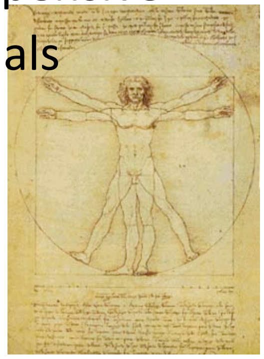
- Several programs to run simultaneously
  - Run one job until I/O
  - Run another job, etc.
- OS manages interactions
  - Which jobs to run (schedule)
  - Protects program's memory from others
  - Decides which to resume when CPU available

OS Complexity

- Increased functionality & complexity
- First OS failures
  - Multics (GE & MIT):
    announced 1963, released 1969
  - OS/360 released with 1000 known bugs
- Need to treat OS design scientifically
- Managing complexity becomes key to…

The Renaissance (1970's)

- Hardware: cheap; humans: expensive
- Users share system via terminals
- The UNIX era
  - Multics:
    - army of programmers, six years
  - UNIX:
    - three guys, two years
    - "Shell": composable commands
    - No distinction between programs & data
- But: response time & thrashing



Industrial Revolution (1980's)

- Hardware very cheap
- Humans expensive
- Widespread use of PCs
  - IBM PC: 1981, Macintosh: 1984
- Simple OS (DOS, MacOS)
  - No multiprogramming, concurrency, memory protection, virtual memory, …
  - Later: networking, file-sharing, remote printing…
  - GUI added to OS ("WIMP")

The Modern Era (1990's-now)

- Hardware cheap; processing demands increasing
- "Real" operating systems on PC's
  - NT (1991); Mac OS X; Linux
- Different modalities:
  - Real-time: Strict or loose deadlines
  - Sensor/Embedded: Many small computers
  - Parallel: Multiple processors, one machine
  - Distributed: Multiple networked processors
    - Think P2P, the Web, Google, cloud

Architectural Trends

- Big Changes
  - In 50 years, almost every computer component now 9 orders of magnitude faster, larger, cheaper

examples	**1983**	**1999**
MIPS	0.5	500
cost/MIP	$100,000	$500
memory	1 MB	1 GB
network	10 Mbit/s	1 Gb/s
disk	1 GB	1 Tbyte

History Lesson

This degree of change has no counterpart in any other area of business.

Examples:

- Transportation -- over the last 200 years, we have gone from horseback (10 miles/hour) to the Concorde (1000 miles/hour) - 2 orders of magnitude.
- Communication -- at the invention of the telephone (voice), TV (video) and fax (text & pictures), communication went from the speed of transportation to nearly the speed of light - 7 orders of magnitude.

Orders of Magnitude

- 10^0

Orders of Magnitude

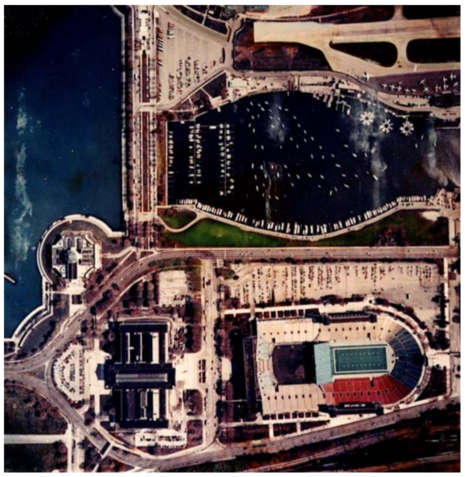
- 10^1

Orders of Magnitude

- 10^2

Orders of Magnitude

- 10^3

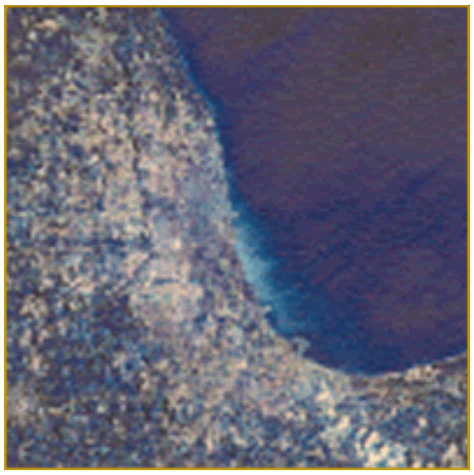


Orders of Magnitude

- 10^4

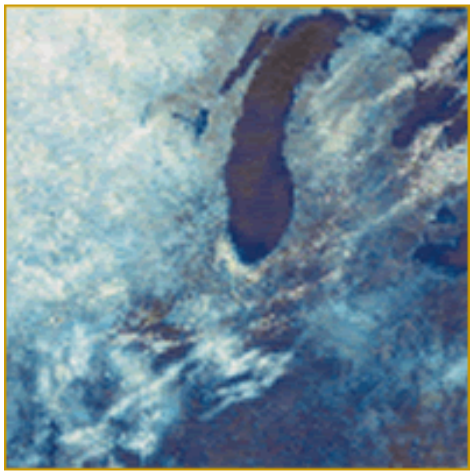
Orders of Magnitude

- 10^5



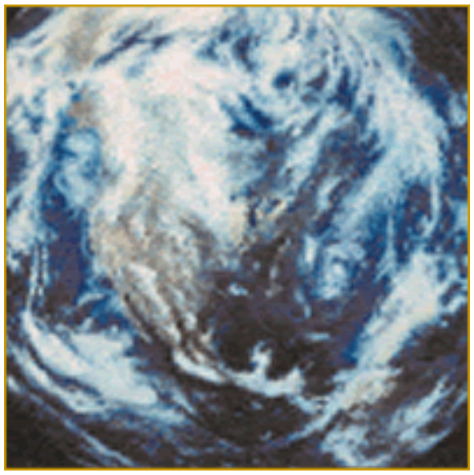
Orders of Magnitude

- 10^6



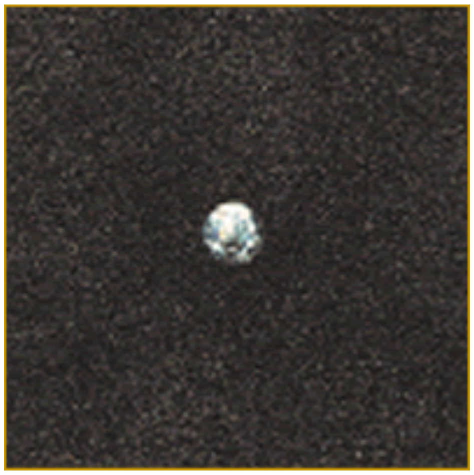
Orders of Magnitude

- 10^7



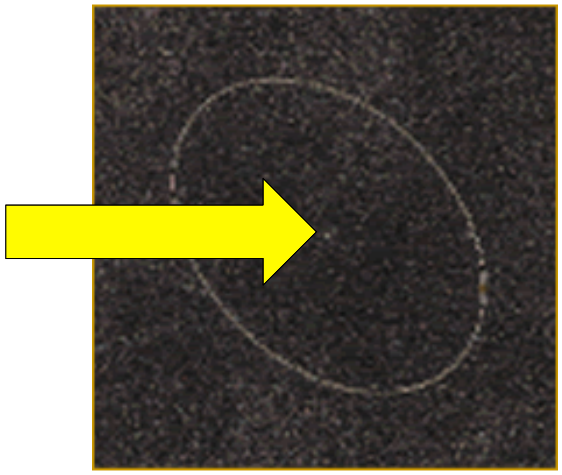
Orders of Magnitude

- 10^8



Orders of Magnitude

- 10^9



Coming Soon

- Moore's Law – running out of steam
- New "features" coming
  - Multiple cores
  - Unreliable memory
  - Solid state drives
  - Serious power/heat constraints
- Other tradeoffs possible
  - Computing power for reliability…