

Introduction



Today

- Welcome to OS
- Administrivia
- OS overview and history

Next time

- Architectural support

Course overview ...

- Everything you need to know

<http://www.aqualab.cs.northwestern.edu/classes/eecs-343-f10/>

- Course staff

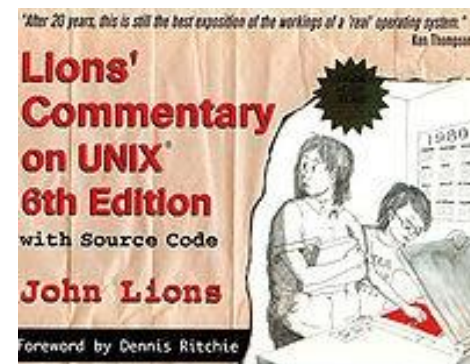
- Fabián Bustamante
- Mario Sánchez (TA)

- Overall structure

- Lectures – read the text before class
- TA Sessions - Once a week and focused on projects
- Homework (4+1)
 - Part of their role is reading enforcers – textbook + xv6 code
 - First one is posted on the web; due in eight days (Sep. 30)
- ...

Course overview ... xv6

- A teaching OS developed in MIT
- Based on UNIX Sixth Edition (aka V6)
 - Used to run on PDP-11 (~40 year arch)
 - Based for Lions' *Commentary on UNIX' 6th Edition*, John Lions



- Ported to x86 and some bug fixes
- 7,300 SLOC (compare: Windows XP – 40 million SLOC; Linux kernel 2.6.32 - 12.6 million SLOC)

Course overview

- Overall structure

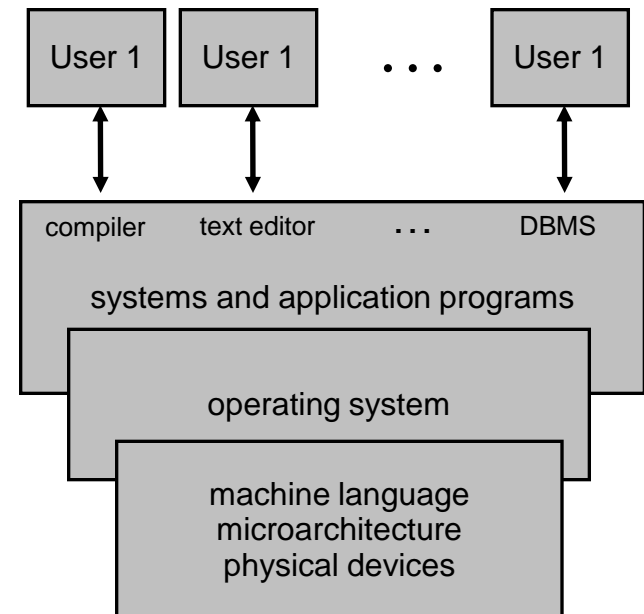
- ...
- Projects (4)
 - First one will be posted next Monday
 - Discuss on Wednesday
 - Due 9 days later (Oct. 6)
- Exams (2)
 - Final on first day of final weeks (Dec. 6)

- *It's not that bad!*

- They are really two classes in one: lectures + projects
- Sometimes they are aligned, sometimes not
- You will work a lot and you will also learn a lot

A computer system - Where's the OS?

- Hardware provides basic computing resources
- Applications define ways in which resources are used to solve users' problems
- *OS controls & coordinates use of hardware by users' applications*
- A few vantage points
 - End user
 - Programmer
 - OS Designer



What is an operating system?

- Extended machine – top-down/user-view
 - Hiding the messy details, presenting a virtual machine that's easier to program than the HW
- Resource manager – bottom-up/system-view
 - Everybody gets a fair-share of time/space from a resource (multiplexing in space/time)
 - A control program – to prevent errors & improper use (CP/M?)
- A bundle of helpful, commonly used things
- Goals
 - Convenience – make solving user problems easier
 - Efficiency – use hardware in an efficient manner (\$\$\$ machines demand efficient use)
 - Easy to modify/evolve

What's part of the OS?

- Trickier than you think: file system, device drivers, shells, window systems, browser, ...
- Everything a vendor ships with your order?
- The one program running at all times, or running in kernel mode
 - Everything else is either a system program (ships with the OS) or an application program
 - *Can the user change it?*
- *Why does it matter? In 1998 the US Department of Justice filed suit against MS claiming its OS was too big*

Why having one?

- For applications
 - Programming simplicity
 - High-level abstractions instead of low-level hardware details
 - Abstractions are reusable across many programs
 - Portability (to != machines configurations/architectures)
- For user
 - Safety
 - Program works within its own virtual machine
 - 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

Why study operating systems?

- Tangible reasons
 - Build/modify one - OSs are everywhere
 - Administer and use them well
 - Tune your favorite application performance
 - Great capstone course
- Intangible reasons
 - Curiosity
 - Use/gain knowledge from other areas
 - Challenge of designing large, complex systems

Major OS issues

- Structure: how is the OS organized?
- Sharing: how are resources shared?
- Naming: how are resources named?
- Security: how is 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?
- Extensibility: can we add new features?
- Communication: how do programs exchange information, including across a network?

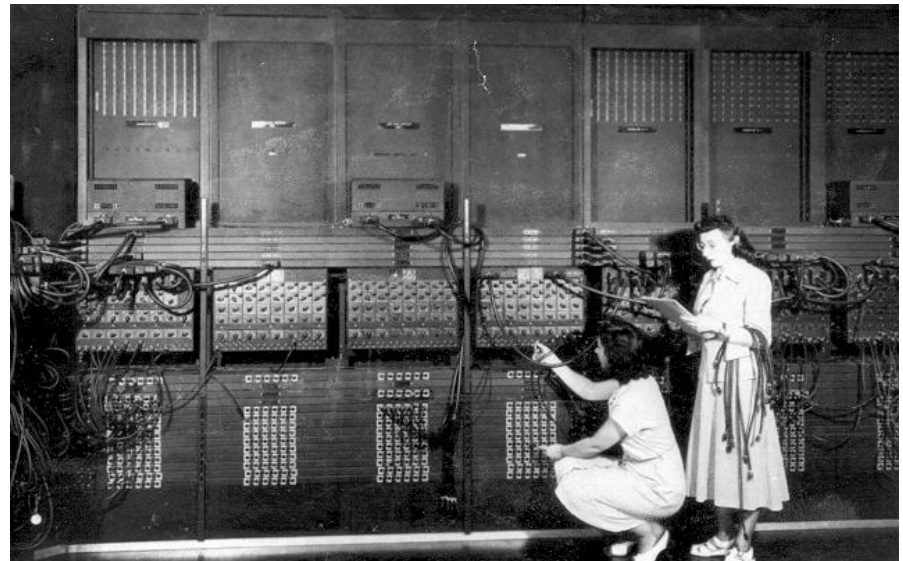
Other OS issues

- Concurrency: how are parallel activities created and controlled?
- Scale and growth: 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? how do we make distribution invisible?
- Accounting: how do we keep track of resource usage, and perhaps charge for it?

There are a huge number of engineering tradeoffs in dealing with these issues!

The evolution of operating systems

- A brief history & a framework to introduce OS principles
- Early attempts – Babbage's (1702-1871)
 - Analytical Engine (Ada Lovelace – World's first programmer)
- 1945-55 – Vacuum tubes and plugboards
 - ABC, MARK 1, ENIAC
 - No programming languages, no OS
 - A big problem
 - Scheduling – signup sheet on the wall



Evolution ... Batch systems (1955)

- Transistors → machs. reliable enough to sell
 - Separation of builders & programmers
- Getting your program to run
 - Write it in paper (maybe in FORTRAN)
 - Punch it on cards & drop cards in input room
 - Operator may have to mount/dismount tapes, setting up card decks, ... setup time!
- Batch systems
 - Collect a tray of full jobs, read them all into tape with a cheap computer
 - Bring them to the main computer where the “OS” will go over each jobs one at a time
 - Print output offline

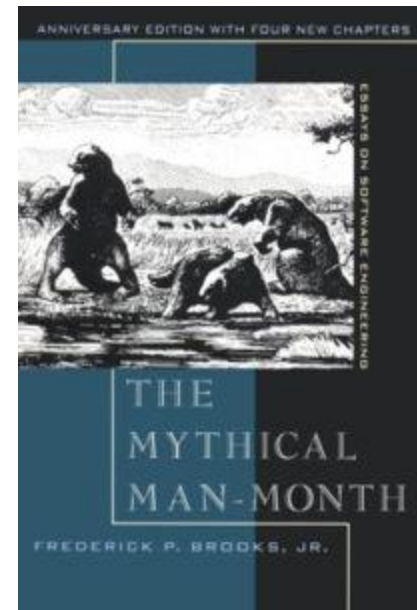
Evolution ... Spooling (1965)



- Disks much faster than card readers & printers
- Spool (Simultaneous Peripheral Operations On-Line)
 - While one job is executing, spool next one from card reader onto disk
 - Slow card reader I/O overlapped with CPU
 - Can even spool multiple programs onto disk
 - OS must choose which one to run next (job sched)
 - But CPU still idle when program interact with a peripheral during execution

Evolution ... Multiprogramming (1965)

- To increase system utilization
 - Keeps multiple runnable jobs loaded in memory at once
 - Overlap I/O of a job with computing of another
 - Needs asynchronous I/O devices
 - Some way to know when devices are done
 - Interrupt or polling
 - Goal- optimize system throughput
 - Maybe at the cost of response time
- IBM OS/360 & the tar pit



Evolution ... Timesharing (1961)

- To support interactive use
 - Multiple terminals into one machine
 - Each user has the illusion of owning the entire machine
 - Goal – optimize response time maybe at the cost of throughput
- Time-slicing
 - Dividing CPU equally among users
 - If jobs are truly interactive, CPU can jump between them without users noticing it
 - Recovers interactivity for the user (why do you care?)
- CTSS (Compatible Time Sharing System), MULTICS (second-system effect) and UNIX

Evolution ... Parallel systems (1962)

- Some applications can be written as multiple parallel threads or processes
 - Can speed up the execution by running on several CPUs
 - 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, multi-core)
 - SMT (simultaneous multithreading [“hyperthreading”])
 - MPPs (massively parallel processors)
 - NOWs (networks of workstations) [clusters]
 - computational grid (SETI @home)

Evolution ... PCs (197x)

- Large-scale integration >> small & cheap machines
- 1974 – Intel's 8080 & Gary Kildall's CP/M
- Early 1980s – IBM PC, BASIC, CP/M & MS-DOS
- User interfaces, XEROX Altos, MACs and Windows



Xerox Alto 1973

IBM PC circa 1981



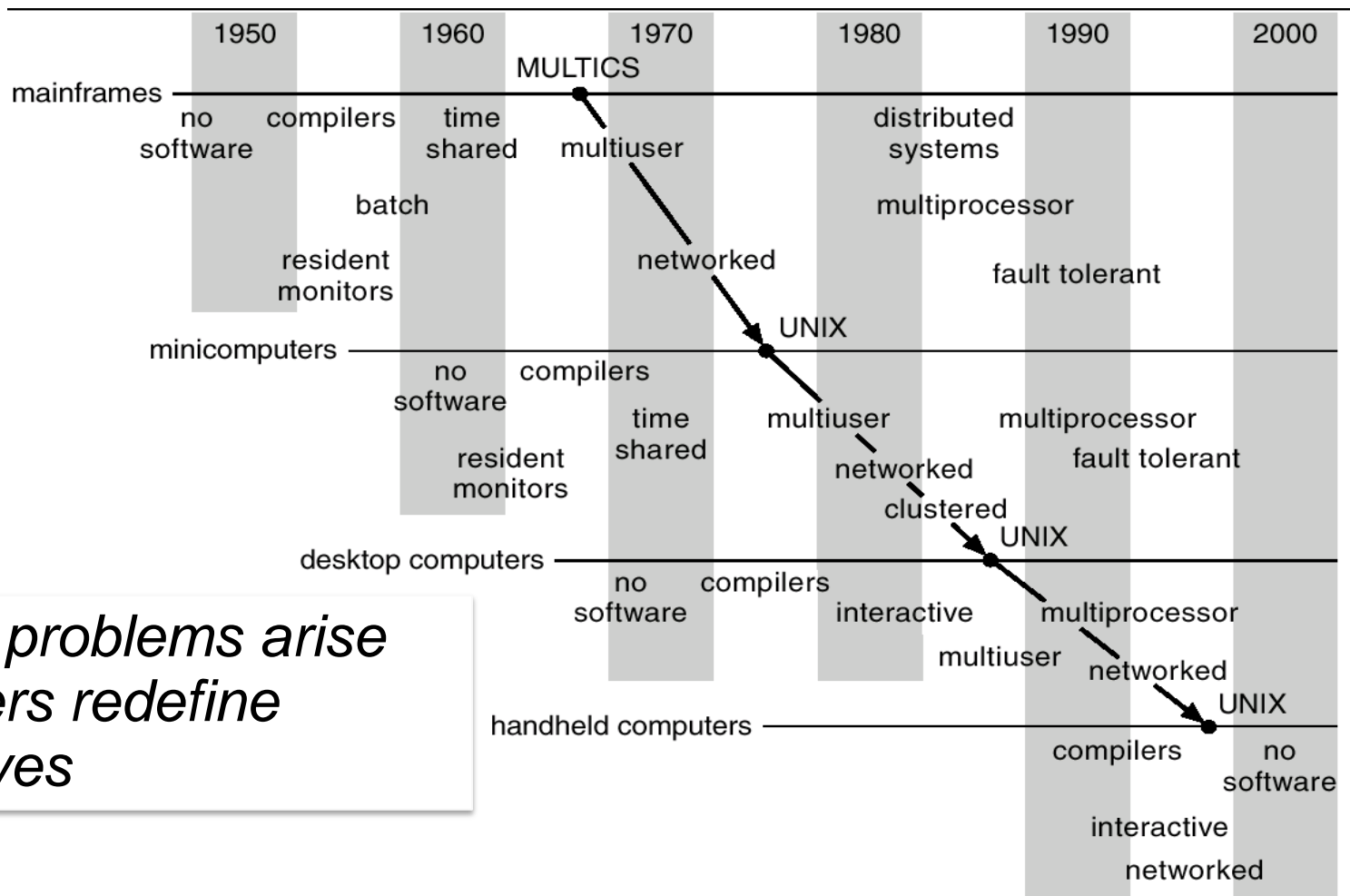
Evolution ... Distributed and pervasive

- Facilitate use of geographically distributed resources
 - Workstations on a LAN or across the Internet
- Support communication between programs
- Speed up is not always the issue, but access to resources
- Architectures
 - Client/servers
 - Mail server, print server, web server
 - Peer-to-peer
 - (Most) everybody is both, server and client

Evolution ... Embedded and pervasive

- Pervasive computing
 - cheap processors embedded everywhere
 - how many are on your body now? in your car?
 - cell phones, PDAs, games, iPod, network computers, ...
- Typically very constrained hardware resources
 - slow processors
 - small amount of memory
 - no disk or tiny disk
 - typically only one dedicated application
 - limited power
- But technology changes fast

"Ontogeny recapitulates phylogeny"*



But new problems arise and others redefine themselves

The development of an embryo repeats the evolution of the species (* Ernst Haeckel)

Summary

- In this class you will learn
 - Major components of an OS
 - How are they structured
 - The most important interfaces
 - Policies typically used in an OS
 - Algorithms used to implement those policies
- Philosophy
 - You may not ever build an OS, but
 - As a CS/CE you need to understand the foundations
 - Most importantly, OSs exemplify the sorts of engineering tradeoffs you'll need to make throughout your careers