CS 414/415
Systems Programming and Operating Systems

Spring 2005

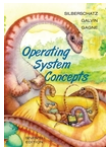
Instructor: Paul Francis

Administrative

- Instructor: Paul Francis, 4108 Upson
- 414 TAs: Joy Zhang, Others…
- 415 TA: Saikat Guha
- Lectures:
  - CS 414: T, R: 10:10 – 11:25 AM
  - CS 415: M: 3:35 – 4:25 PM
- www.cs.cornell.edu/courses/cs414/2005fa/

Adminstration

- Mailing list for student discussions:
  - francis-class-l at cs.cornell.edu
  - http://lists.cs.cornell.edu/mailman/listinfo/francis-class-l
- Class announcements:
  - Via CMS mail, and posts on course website
- Required Textbook:
  - Operating Systems Concepts: 7th Edition Silberschatz, Galvin and Gagne
  - Older editions ok, but you must figure out which sections to read



CS 414: Overview

- Prerequisite:
  - Mastery of CS 314 material
- CS 414: Systems Programming and Operating Systems
  - How OS's work, and how to program over them
  - You can't understand computers and programming withou understanding this!
    - That's why we require it! :)

CS 414: Your load

- Reading for every lecture
  - Do this before lecture!
- Lectures
- Homework (weekly)
- Small programming tasks
  - Designed to emphasize performance trade-offs in OS's
  - (A series of unfortunate events)

CS 414: Evaluation

- Two prelims and a final exam (~60%)
- A few pop quizzes on homework (~20%)
  - In lieu of grading!
  - Pop quiz will be a minor variation on homework
- Small programming tasks (~20%)
  - Style counts!!!!
- Intangible (up to 10%)
- (these are rough percentages and subject to tweaks)

CS 415: Overview

- CS 415: Practicum in Operating Systems
  - Projects that complement course material
  - Best way to learn about OSs
- This semester:
  - Build various components of operating systems
    - Threads, networking, file systems
    - Similar to www.cs.cornell.edu/courses/cs414/2004sp/
  - Work individually or in pairs
  - More detail in first 415 lecture
- Enrollment in CS 415 is optional

Academic Integrity

- I encourage student discussion
- But you must write your own code
  - I reserve the right to "spot check" you on code
- Homework
  - I suggest that you try to do it individually
  - If you have trouble, discuss with others
  - After you finish, compare with others

Course Material

- Introduction, history, architectural support
- Concurrency, processes, threads
- Synchronization, monitors, semaphores
- Networking, distributed systems
- Memory Management, virtual memory
- Storage Management, I/O, filesystems
- Security
- Case studies: Windows XP, Linux

Why take this course?

- I said it already:
- You cannot write good performing software (of any complexity) without understanding the OS
- You cannot design or specify a system without understanding the OS
- And you might even write an OS someday!

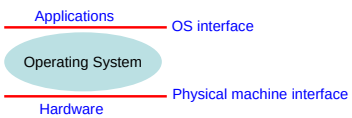
What is an Operating System?

- A number of definitions:
  - Just google for define: Operating System
- A few of them:
  - "Everything a vendor ships when you order an operating system"
  - "The one program running at all times on the computer"
  - "A program that manages all other programs in a computer"
- Required memory varies: less than 1 MB to a few GB
- Whatever it is, every computer has one!

Operating System: Definition

Definition

An Operating System (OS) provides a virtual machine on top of the real hardware, whose interface is more convenient than the raw hardware interface.



Advantages

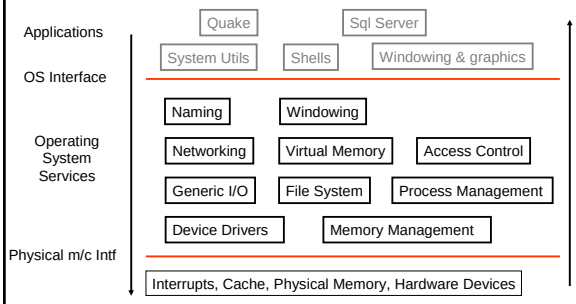
Easy to use, simpler to code, more reliable, more secure, …

You can say: "I want to write XYZ into file ABC"

Operating Systems Services

- Manage physical resources:
  - It drives various devices
    - Eg: CPU, memory, disks, networks, displays, cameras, etc
- Provide abstractions for physical resources
  - Provide virtual resources and interfaces
    - Eg: files, directories, users, threads, processes, etc
  - Simplify programming through high-level abstractions
  - Provide users with a stable environment, mask failures
- Isolate and mediate between entities
  - Trusted intermediary for untrusted applications

What is in an OS?



Logical OS Structure

Issues in OS Design

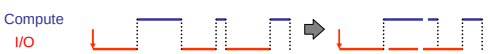
- **Structure:** how is an operating system organized ?
- **Sharing:** how are resources shared among users ?
- **Naming:** how are resources named by users or programs ?
- **Protection:** how is one user/program protected from another ?
- **Security:** how to authenticate, control access, secure privacy ?
- **Performance:** why is it so slow ?
- **Reliability and fault tolerance:** how do we deal with failures ?
- **Extensibility:** how do we add new features ?

Issues in OS Design

- **Communication:** how can we exchange information ?
- **Concurrency:** how are parallel activities created and controlled ?
- **Scale, growth:** what happens as demands or resources increase ?
- **Persistence:** how can data outlast processes that created them
- **Compatibility:** can we ever do anything new ?
- **Distribution:** accessing the world of information
- **Accounting:** who pays bills, and how to control resource usage

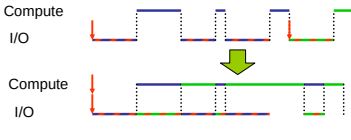
History of Operating Systems

- Initially, the OS was just a run-time library
  - You linked your application with the OS,
  - loaded the whole program into memory, and ran it
  - How do you get it into the computer? Through the control panel!
- Simple batch systems (mid1950s – mid 1960s)
  - Permanently resident OS in primary memory
  - Loaded a single job from card reader, ran it, loaded next job...
  - *Control cards* in the input file told the OS what to do
  - *Spooling* allowed jobs to be read in advance onto tape/disk

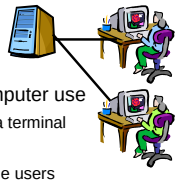


Multiprogramming Systems

- *Multiprogramming* systems increased utilization
  - Developed in the 1960s
  - Keeps multiple runnable jobs loaded in memory
  - Overlaps I/O processing of a job with computation of another
  - Benefits from I/O devices that can operate asynchronously
  - Requires the use of interrupts and DMA
  - Optimizes for *throughput* at the cost of *response time*



Time Sharing Systems



Timesharing (1970s) allows interactive computer use
- Users connect to a central machine through a terminal
- User feels as if he has the entire machine
- Based on time-slicing: divides CPU among the users
- Allows active viewing, editing, debugging, executing process
- Security mechanisms needed to isolate users
- Requires memory protection hardware for isolation
- Optimizes for *response time* at the cost of *throughput*

Personal Operating Systems

- Earliest ones in the 1980s
- Computers are cheap ⇒ everyone has a computer
- Initially, the OS was a library
- Advanced features were added back
  - Multiprogramming, memory protection, etc

Networked Operating Systems

- In the 90's, the Internet pushed us back towards shared systems
- Often PCs operate as "dumb terminals"
- Web applications
  - Web mail
  - Network games
  - Internet commerce
- Network stacks become faster, more sophisticated
  - Though in general this stayed on the main CPU

Distributed Operating Systems

- Cluster of individual machines
  - Over a LAN or WAN or fast interconnect
  - No shared memory or clock
- Asymmetric vs. symmetric clustering
- Sharing of distributed resources, hardware and software
  - Resource utilization, high availability
- Permits some parallelism, but speedup is not the issue
- SANs, Oracle Parallel Server, Google

Parallel Operating Systems

- Multiprocessor or tightly coupled systems
- Many advantages:
  - Increased throughput
  - Cheaper
  - More reliable
- Asymmetric vs. symmetric multiprocessing
  - Master/slave vs. peer relationships
- Examples: SunOS Version 4 and Version 5

Real Time Operating Systems

- Goal: To cope with rigid time constraints
- Hard real-time
  - OS guarantees that applications will meet their deadlines
  - Examples: TCAS, health monitors, factory control
- Soft real-time
  - OS provides prioritization, on a best-effort basis
  - No deadline guarantees, but bounded delays
  - Examples: most electronic appliances
- Real-time means "predictable"
  - NOT fast

Ubiquitous Systems

- PDAs, personal computers, cellular phones, sensors
- Challenges:
  - Small memory size
  - Slow processor
  - Different display and I/O
  - Battery concerns
  - Scale
  - Security
  - Naming
- We will look into some of these problems

Over the years

- Not that batch systems were ridiculous
  - They were exactly right for the tradeoffs at the time
- The tradeoffs change

	1981	2005	Factor
MIPS	1	2000	2000
$/MIPS	$100000	$5000	20000
DRAM	128KB	1GB (2?)	16000
Disk	10MB	80GB (more?)	8000
Net Bandwidth	9600 b/s	10 Gb/s	1000000
# Users	>> 10	<= 1	0.1

- Need to understand the fundamentals
  - So you can design better systems for tomorrow's tradeoffs