CS 4410/4411 Systems Programming and Operating Systems

Spring 2009

Instructor: Ken Birman

Administrative

- **Instructor: Ken Birman, x5-9199**
- **Office hours: 4119b Upson**
  - Schedule via email, any day/time
  - TA office hours: Listed on our web page
- **CS4410 Lectures:** 205 Thurston Hall
  - Tuesday, Thursday 10:10-11:25 AM
- **CS4411 Section:** B14 Hollister Hall
  - Tuesday 3:35-4:25 PM

www.cs.cornell.edu/courses/cs44100/2007sp

Course Help

- News group:
  - Use newsstand.cit.cornell.edu, group "cornell.class.cs4410"
  - Please subscribe today and check often. Post any time you have a question or comment and feel free to help other people out if you know the answer!
- Required Textbook:
  - Operating Systems Concepts: 8th Edition Silberschatz, Galvin and Gagne
  - 7th edition will also work. Copies on reserve in the Engineering library

CS 4410: Overview

- Prerequisite:
  - Mastery of CS 3610 material
- CS 4410: Operating Systems
  - Fundamentals of OS design
  - How parts of the OS are structured
  - What algorithms are commonly used
  - What are the mechanisms and policies used
- Evaluations:
  - Weekly homework
  - Prelims, Exams: **Thursday Feb 26, Thursday April 2, Friday May 8**
  - Readings: research papers

CS 4411: Overview

- CS 4411: Practicum in Operating Systems
  - Projects that complement course material
  - Expose you to cutting edge system design
  - Best way to learn about OSs
- This semester:
  - Build a new kind of file storage system for scientific research on global climate change and environment monitoring
  - Will use normal PCs
  - Designed to be done by a single person working alone
  - Weekly sections on the projects

Grading

- CS 4410: Operating Systems is curved
  - Prelims ~ 25% each
  - Final ~ 40%
  - Assignments ~ 10%
  - We also do some subjective fiddling with grades.
- CS 4411: Systems Programming
  - Six stages~ 100%. No exams.
- This is a rough guide

Academic Integrity

- **Submitted work should be your own**
- Acceptable collaboration:
  - Clarify problem, C syntax doubts, debugging strategy
- Dishonesty has no place in any community
  - May NOT be in possession of someone else's homework/project
  - May NOT copy code from another group
  - May NOT copy, collaborate or share homework/assignments
  - University Academic Integrity rules are the general guidelines
- Penalty depends on circumstances, but can be as severe as an 'F' in CS 4410 and CS 4411

What is an Operating System?

- An operating system (abbreviated O/S) is the software infrastructure that runs a single computer system.
  - It is responsible for managing and coordinating activities and sharing the (limited) resources of the computer.
- It hosts applications:
  - Provides a collection of system calls via library APIs
  - Creates a *virtual runtime environment*
- It exploits concurrency by scheduling activities to occur in parallel

What is an Operating System?

- If you are using a computer… you are using an O/S
- All computers, including cell phones, cameras, video game consoles, and even GPS navigators use an O/S of some type.
  - But there are multiple kinds of operating systems
  - Small devices often have stripped-down operating systems
- Can think of the O/S as being "everything between the programming language and the hardware"

Why take this course?

- Operating systems are the core of a computer system
  - To use computers effectively, you need to know
    - How to code applications in a programming language
    - How those applications can ask the O/S to perform actions for it
    - How the computer hardware really works
  - The O/S is a bridge between the hardware and your application
  - The O/S also creates a number of new abstractions beyond what the hardware has built into it (example: "shared memory")
- Operating systems are exceptionally complex systems
  - Huge, parallel, very expensive, and hard to build
    - Windows Vista, Windows 7.0: 1000s of people, decades …
  - We want to learn to build complex things, too!

Why take CS4411?

- Sometimes, reading about it just isn't the same as doing it…
  - Knowing how an O/S works is useful
  - Hands-on experience building systems is *invaluable*
- You'll get a better job… be happier in your life… get rich…. and will be assured entry into heaven
- (But just the same, CS4411 is optional)

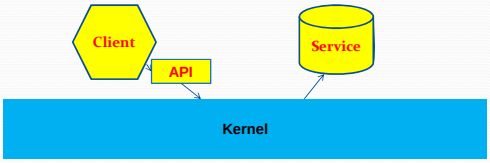
Sparse Petabyte File System

- Many kinds of devices can store files
  - Hard disks, USB memory, etc
- The O/S treats them all "the same"
  - It has a standard file system service
  - That service will work over any device that can read and write blocks of bytes (usually 512 or 1024 at a time)
  - But it assumes that files aren't enormous and that a file with N bytes in it is like a byte string N bytes long

Concept of a "service"

- As O/S got large, designers began to split functionality up into components
  - The kernel is one component: code that needs to run with special privilages
  - The file system is often implemented in a service that runs with high priority but outside the kernel
- There are many services running on a typical O/S
- We'll add an extra one to Windows.
  - It will have its own API (modeled after the standard one)
  - Test programs can talk to the service through this API

Concept of a "service"



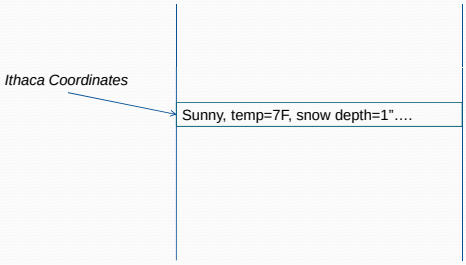
- The Sparse Petabyte File System will run as a service
- At first, it will store data in memory, although we'll add disk "backing" storage later

Sparse Petabyte File System

- Suppose we are collecting, for example, weather data for the entire globe

  - We might take a coordinate system and use it to represent locations on the earth
  - Then the weather, as of 11:10am on Thursday Jan. 22 in Ithaca would be some sort of record stored at some specific place in the file
- The file could be *very big* but perhaps full of holes
  - We only know some of the weather data…

Sparse Petabyte File System



Indexing our file

- Idea is to have a formula that lets us convert coordinates into a location in the file
- For example: Ithaca is at 42°26′37″N, 76°30′00″W
  - Represent as a really big number:
    42263700007630000000
  - Suppose that our weather "record" is 16 bytes long…
  - Then the weather record for Ithaca is at offset
    - 16 x 42263700007630000000
  - So… open file… seek to desired offset… read/write 16 bytes. You'll read zeros if you hit a hole

File access using "seek"

- Imagine that a file has an associated pointer
  - When you open an existing file, it points to offset 0
  - When you read, for example, 16 bytes
    - You get the next 16 starting where the pointer is
    - And the pointer automatically increments by 16
- The *seek* system call lets you set the pointer to any place that you like

Special case: EOF

- We say that the End of File or EOF point is reached when a read tries to go beyond the point where the last data in the file was written
  - E.g. file has 1000 bytes, but you seek to location 2000
- To tell the user (who might care)
  - Fstat tells you how big the file currently is
  - Read tells you how many bytes you actually read, e.g. 0 if you are completely beyond the EOF
  - Thus a read of 16 bytes could return 4 bytes... or 0...
- In our file system, a write will always succeed

Sparse Petabyte File System

- So this leads to the basic idea
  - Build a file system that can store a new kind of file
    - Very, very big (1 PB = 1000 TB; 1 TB = 1000 GB; 1 GB = 1000 MB)
    - But mostly full of holes: the data is in little dribs and drabs here and there
  - Basic interface is like for any file system:
    - Create/read/write/seek/fstat
    - Seek lets us move a "file pointer" to say where the read or write will occur
    - Read and write access variable numbers of bytes
    - Empty chunks within the file read as zeros

Sparse Petabyte File System

- How can it be done?
  - We'll implement a tree-structure to do lookups
- Sounds expensive!
  - And it will be... finding data won't be easy or fast
- Motivating.... A concurrent solution!
  - With multiple threads, our file service will be able to handle multiple reads and writes simultaneously

Sparse Petabyte File System

- What will I learn by doing this project?
  - How to build a new service on a Windows system (in fact the same approach works on Linux too)
  - Thinking about and implementing a good data structure for locating chunks of the data
  - Using threads to maximize concurrency
  - Testing and debugging the solution
  - Tuning it for blazingly high speed
  - Running on a multicore computer donated by Intel
- These are skills that you can use in many settings

Win a cool prize!

- Doing CS4411 is a great experience
  - But also a way to figure out if you "have what it takes" to be a great systems person
  - So... we want solutions that work, and aren't buggy
  - ... but we also want awesome performance
- Winner gets a prize!

But I'm not taking CS4411!

- You'll hear about it from time to time, anyhow
- You could actually still add the class
  - You've only missed one (organizational) meeting
  - The actual assignment hasn't even been posted yet
- Common questions
  - *How hard is the project?*
    - We designed it so that everyone can succeed
  - *Can I do the project with a friend?*
    - Sorry, not in 2009. This is a single-person project

Back to the Future…

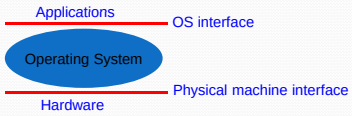
CS4410 Goal?

- Learn to reason about complex systems (especially ones that employ concurrency)
  - Internet, air traffic control, e-government, weather sensing and prediction systems, social networks, etc
  - Operating systems are the "archetypical" example of a highly complex yet structured system.
- We'll want to
  - Understand how to structure a big system
  - Understand how to exploit concurrency
  - Prove the correctness of our solutions

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.

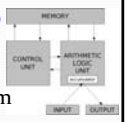


Advantages

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

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

What's a **virtual machine**?



- In 1940's **John von Neumann** proposed a model for a stored-program computing system
- **Alan Turing** showed that any computer system with sufficient power can "emulate" any other
- Today we think of virtual machines
  - A virtual machine looks like a computer from the point of view of the applications running on it
  - But will often be an emulation created by a more primitive computer system hidden from the application

Anti-virus puzzle

- Suppose that you are worried that your PC has been infected by a nasty computer virus
  - So you run an anti-virus program
  - It reports good news: no viruses detected
- *Did it run on the real system, or in a virtual machine?*

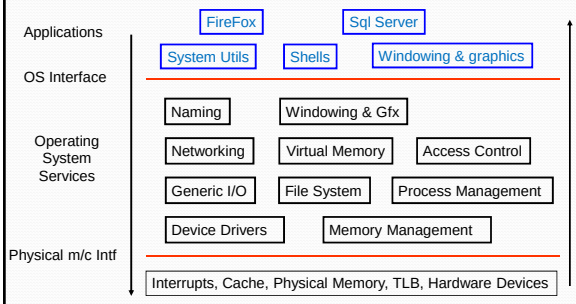
Solution?

- There isn't any solution without special hardware
  - Believe it or not, this is a *real* problem!
  - A virtual machine isn't distinguishable from a real one
- We'll revisit this issue (much) later in the course
- But the basic take-away is that the operating system is a magician that creates virtual worlds… in which our applications run…

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

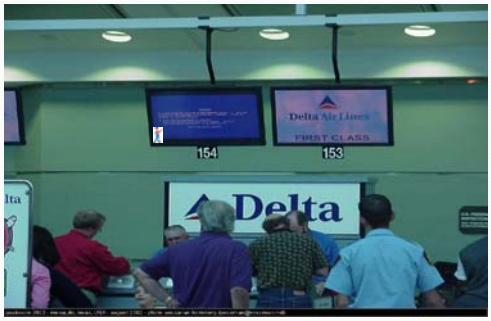
- **Structure:** how is an operating system organized ?
- **Sharing:** how are resources shared among users ?
- **Naming:** how are resources named by users or programs ?
- **Protection:** how is one user/program protected from another ?
- **Security:** how to authenticate, control access, secure privacy ?
- **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

Where's Waldo?

Where's Waldo?

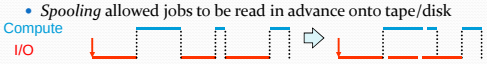


Blue Screen of Death

- Actually illustrates a kind of virtual machine!
- Every modern computer has a BIOS (Basic Input Output Subsystem)
  - This is a small, dedicated operating system used to start the machine (to "boot" it)
  - It operates things like the power switch
  - And if the main O/S crashes… the BIOS shows a blue screen with some data about how it died!

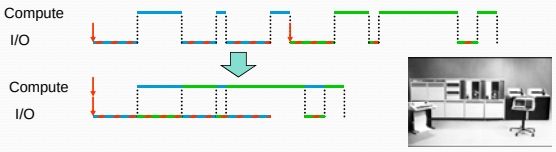
Short O/S History

- 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*

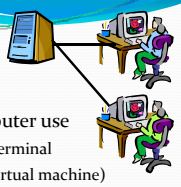


Batch Systems

- User submitted "job", often as a card deck

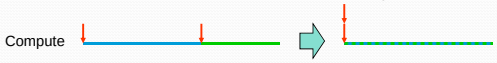
- Computer collected the jobs into a batch, then ran the whole batch at once on some schedule
- Results printed on a shared line printer

Time Sharing Systems



Timesharing (1970s) allows interactive computer use

- Users connect to a central machine through a terminal
- User feels as if she has the entire machine (a virtual machine)
- Based on time-slicing: divides CPU equally 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

Distributed Operating Systems

- Enabled by the emergence of the Internet ~1980
- 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

Parallel Operating Systems

- Aimed at multicore 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
- Recent development: All our machines are multicore now. But as recently as 5 years ago multicore was common only on servers

Cloud Computing

- Really big data centers to support Google, Facebook, Yahoo, eBay, Amazon, Flickr, Twitter...
  - Racks and racks of computers
  - O/S specialized to manage massive collections of machines and to help us built applications to run on them

- Actually, the O/S itself is relatively standard
  - What changes is the set of associated *services*
  - Debate: is the O/S just the thing that runs one machine, or does it "span" the whole data center?

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	1000	1000
$/MIPS	$100000	$5000	20000
DRAM	128KB	512MB	4000
Disk	10MB	80GB	8000
Net Bandwidth	9600 b/s	100 Mb/s	10000
# Users	>> 10	<= 1	0.1

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