CS 4410/4411
Systems Programming
and
Operating Systems

Fall 2008

Instructor: Hakim Weatherspoon

Who am I?

- Prof. Hakim Weatherspoon
  - (Hakim means Doctor, wise, or prof. in Arabic
  - Background in Education
    - Undergraduate University of Washington
      - Played Varsity Football
        - » Some teammates collectively make $100’s of millions
        - » I teach!!!
    - Graduate University of California, Berkeley
      - Some class mates collectively make $100’s of millions
      - I teach!!!
  - Background in Operating Systems
    - Peer-to-Peer Storage
      - Antiquity project - Secure wide-area distributed system
      - OceanStore project – Store your data for 1000 years
    - Network overlays
      - Bamboo and Tapestry – Find your data around globe
    - Tiny OS
      - Early adopter in 1999, but ultimately chose P2P direction

Goals for Today

- Why take this course on Operating Systems?
- What is an Operating System?
- History of Operating System design
- Oh, and “How does this class operate?”

Interactive is important!

Ask Questions!

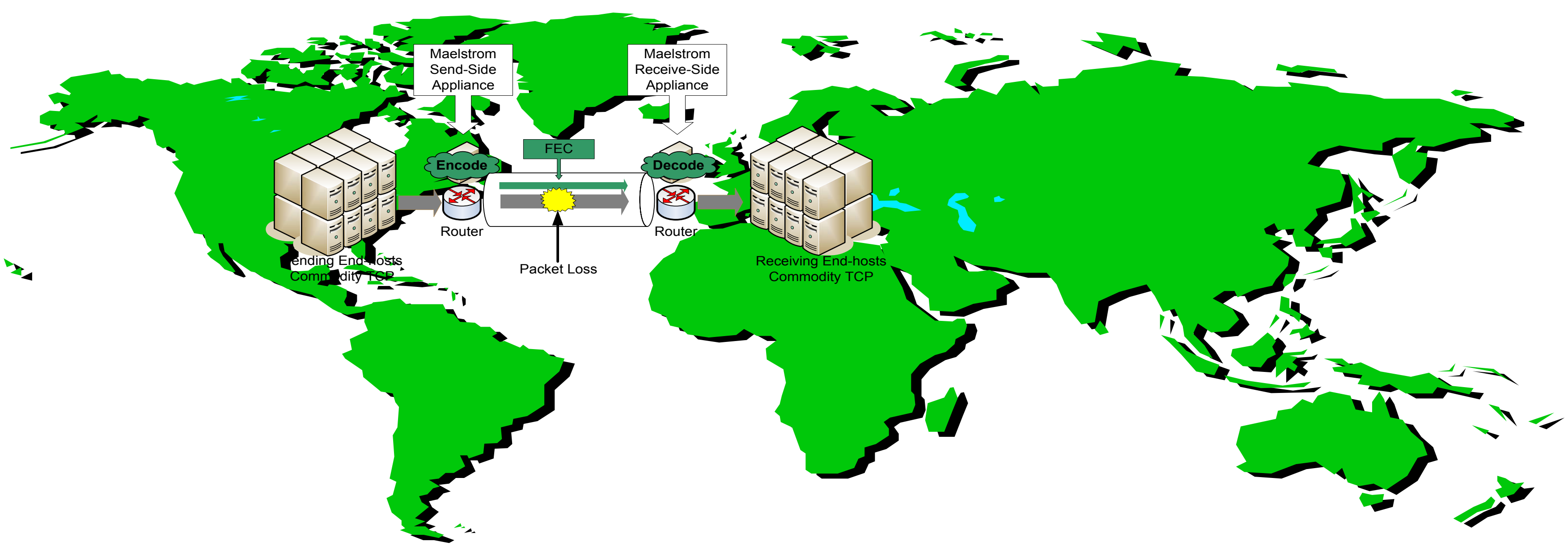
Why take this course?

Why take this course?

- Operating systems are the core of a computer system
  - OS is magic, unknown, frustrating, and/or scary to most people.
  - Course will demystify OS!

Why take this course?

- So that you can help me!



Functionality comes with complexity?

- Every piece of computer hardware different
  - Different CPU
    - Pentium, PowerPC, ColdFire, ARM, MIPS
  - Different amounts of memory, disk, …
  - Different types of devices
    - Mice, Keyboards, Sensors, Cameras, Fingerprint readers
  - Different networking environment
    - Cable, DSL, Wireless, Firewalls,…
- Questions:
  - Does the programmer need to write a single program that performs many independent activities?
  - Does every program have to be altered for every piece of hardware?
  - Does a faulty program crash everything?
  - Does every program have access to all hardware?

The purpose of an OS:

- Two main functions:
- Manage physical resources:
  - It drives various devices
    - Eg: CPU, memory, disks, networks, displays, cameras, etc
  - Efficiently, reliably, tolerating and masking failures, etc

- Provide an execution environment to the applications running on the computer (programs like Word, Emacs)
  - 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

But OS designs are Complex

- Operating systems are a class of exceptionally complex systems
  - They are large, parallel, very expensive, not understood
    - Windows NT/XP: 10 years, 1000s of people, …
  - Complex systems are the most interesting:
    - Internet, air traffic control, governments, weather, relationships, etc

- How to deal with this complexity?
  - Abstrations and layering
  - Our goal: systems that can be *trusted* with sensitive data and critical roles

What is an Operating System?

- Magic!

- 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”

What is an OS?

Application

Virtual Machine Interface

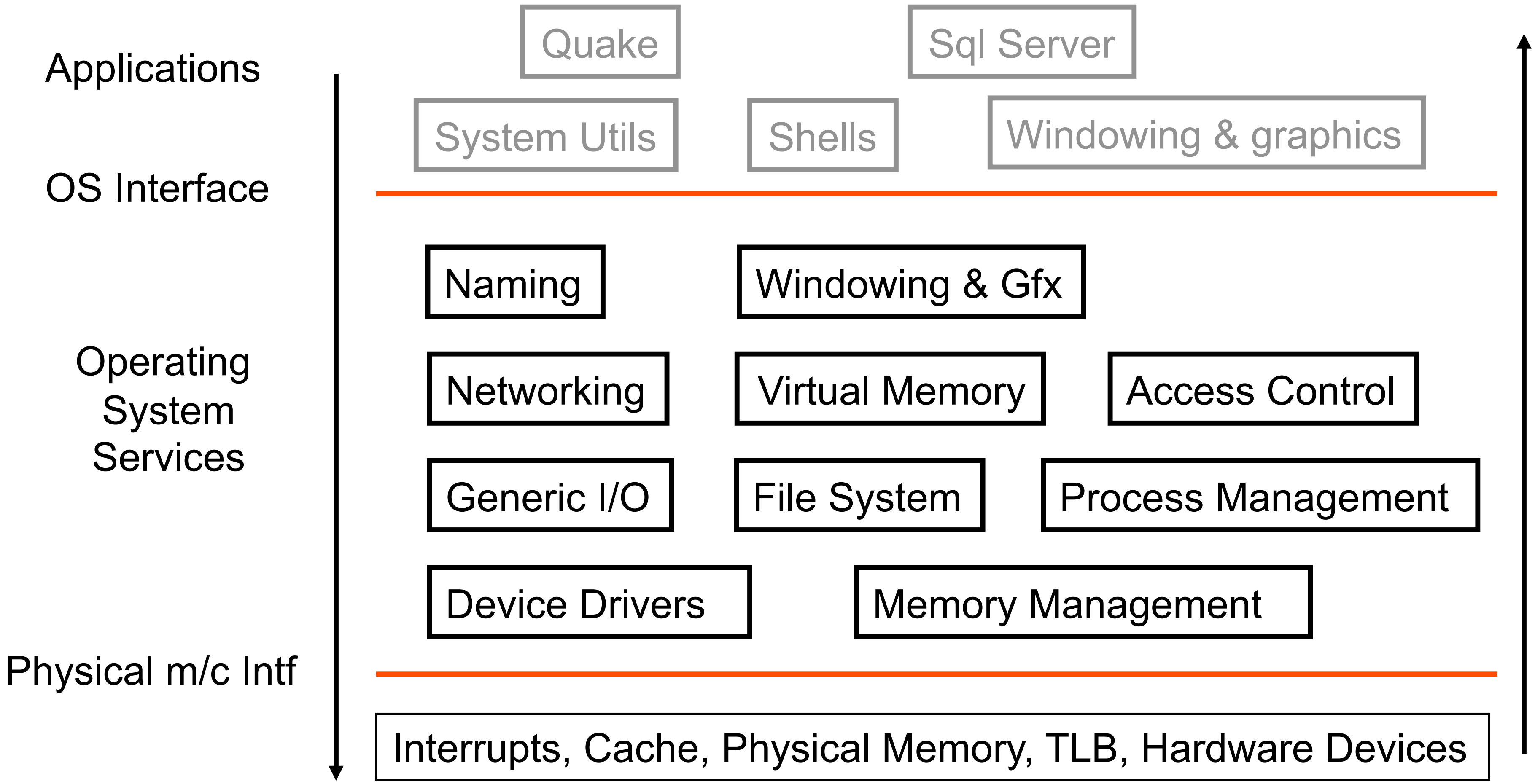
Operating System

Physical Machine Interface

Hardware

- A Virtual Machine Abstraction
  - An operating system implements a virtual machine that is (hopefully) easier and safer to program and use than the raw hardware
- What is the hardware interface?
  - Physical reality
- What is the application interface?
  - Nicer abstraction

What is in an OS?



Logical OS Structure

What are the 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 ?

What are the 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

Why is this material critical?

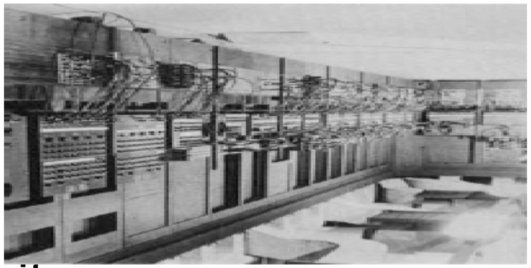
- Concurrency
  - Therac-25, Ariane 5 rocket (June 96)
- Communication
  - Air Traffic Control System
- Virtual Memory
  - Blue Screens of Death
- Security
  - Credit card data

Where's the OS? Melbourne

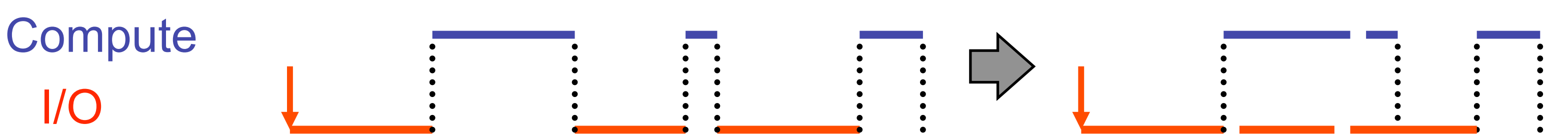
Photo courtesy of zem.

Where's the OS? Mesquite, TX

History of Operating Systems

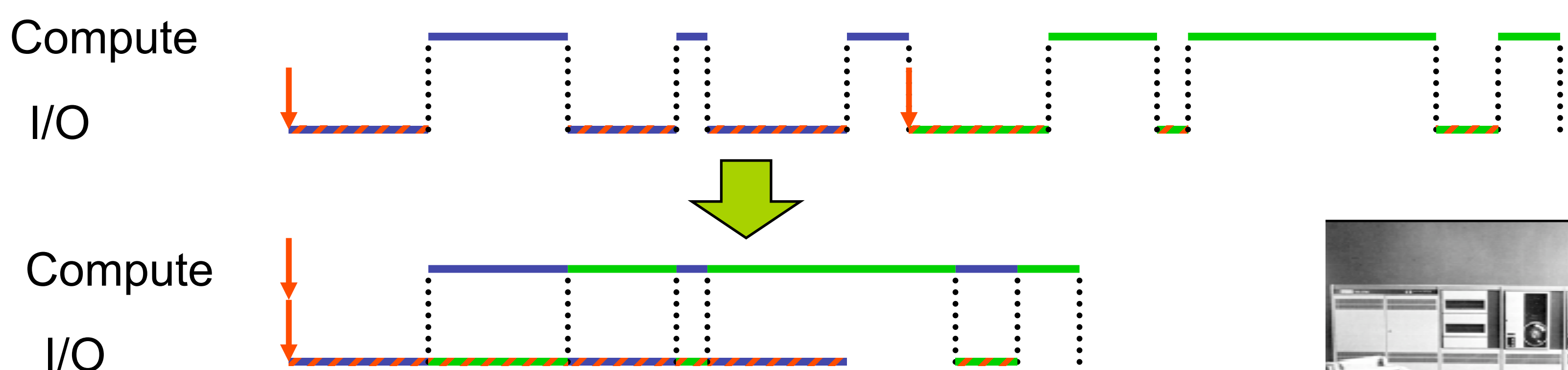


- Initially, the OS was just a run-time library
  – You linked your application with the OS,
  – loaded the whole program into memory, and ran it
  – How do you get it into the computer? Through the control panel!
- Simple batch systems (mid1950s – mid 1960s)
  – Permanently resident OS in primary memory
  – 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

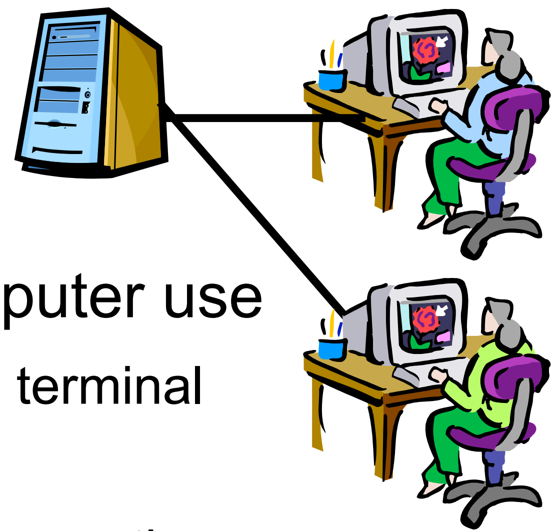


Multiprogramming Systems

- *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*



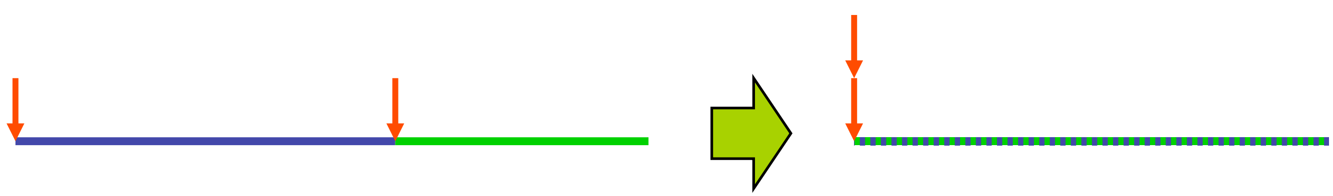
Time Sharing Systems



Timesharing (1970s) allows interactive computer use

- Users connect to a central machine through a terminal
- User feels as if she has the entire 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*

Compute



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

- 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	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

Why Study Operating Systems?

- Learn how to build complex systems:
  - How can you manage complexity for future projects?
- Engineering issues:
  - Why is the web so slow sometimes? Can you fix it?
  - What features should be in the next mars Rover?
  - How do large distributed systems work? (Bit torrent, etc)
- Buying and using a personal computer:
  - Why different PCs with same CPU behave differently
  - How to choose a processor (Opteron, Itanium, Celeron, Pentium, Hexium)? [Ok, made last one up]
  - Should you get Windows XP, Vista, Linux, Mac OS …?
  - Why does Microsoft have such a bad name?
- Business issues:
  - Should your division buy thin-clients vs PC?
- Security, viruses, and worms
  - What exposure do you have to worry about?

Administrative

- Instructor: Hakim Weatherspoon
  - hweather@cs.cornell.edu
  - Office Location: 4116 Upson

- 4411 Instructor/TA: Oliver Kennedy

- 4410 lead TA: Nazrul Alam
  - Grad TA: Levent Kartaltepe

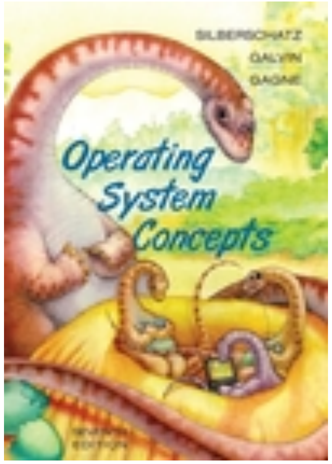
- Lectures:
  - CS 4410: Tu, Th: 10:10 – 11:25 PM, Thurston 203
  - CS 4411: Mon: 3:35 – 4:25 PM, Hollister 372

Course Help

- Course staff, office hours, etc:
  - http://www.cs.cornell.edu/courses/cs4410/2008fa

- Required Textbook:
  - Operating Systems Concepts: 8th Edition (or 7th)

    Silberschatz, Galvin and Gagne



CS 4410: Overview

- Prerequisite:
  - Mastery of CS 3410 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
  - Midterm, Exams
  - Readings: research papers

CS 4411: Overview

- CS 4411: Practicum in Operating Systems
  - This is the lab course for 4410
  - 4411 projects complement 4410 material
  - Expose you to cutting edge system design
  - Best way to learn about OSs

- Concepts covered include:
  - Threads, Synchronization, File systems, Networking, …

- Class structure
  - Work in groups of two or three
  - Weekly sections on the projects
  - 6 bi-weekly project assignments.

Grading

- CS 414: Operating Systems
  - Midterm ~ 30%
  - Final ~ 50%
  - Assignments ~ 10%
  - Subjective ~ 10%

  - Regrade policy
    - Submit written request to lead TA. TA will pick a different grader
    - Submit another written request, lead TA will regrade directly.
    - Submit another written request for professor to regrade.

- CS 4411: Systems Programming
  - Six projects ~ 100%

- 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 can be as severe as an 'F' in CS 4410 and CS 4411

Course Material

- 1 Week: Introduction, history, architectural support
- 1 Week: Processes, threads, cpu scheduling
- 3 Weeks: Concurrency, synchronization, monitors, semaphores, deadlocks
- 1 Week: Memory Management, virtual memory
- 1 Week: Storage Management, I/O
- 1 Week: File systems
- 2 Weeks: Networking, distributed systems
- 1 Week: Security
- 1 Week: Case studies: Windows XP, Linux