6.824 2006 Lecture 1: Introduction and O/S Review

Opening
- building distributed systems
- construction techniques, robustness, good performance
- lectures on design, paper reading for case studies
- you'll build real systems
- why take the course?
  - synthesize many different areas in order to build working systems
  - Internet has made area much more attractive and timely
  - hard/unsolved: not many deployed sophisticated distrib systems

Example:
- how to build HotMail?
- mail arrives from outside world
- store it until...
- user's Outlook/Eudora reads/deletes/saves it

Simple Solution:
- One server w/ disk to store mail-boxes
- [picture: MS, sending "clients", reading clients]
- What happens as your mail service gets popular?

Topic: Stable performance under high load
- Example: Starbucks.
  - 5 seconds to write down incoming request. 10 seconds to make it.
  - [graph: x=requests, y=output]
  - max thruput at 4 drinks/minute.
  - what happens at 6 req/min?
  - thruput goes to zero at 12 requests/minute.
- Efficiency *decreases* with load -- bad.
- Careful system design to avoid this -- flat line at 4 drinks.
  - Peets, for example.
- Better: build systems whose efficiency *increases* w/ load
  - w/ e.g. batching, disk scheduling

Topic: scalable performance
- What if more clients than one Hotmail server can handle?
- How to use more servers to handle more clients?
- Idea: partition users across servers
  - bottlenecks: how to ensure incoming mail arrives at the right server?
  - scaling: will 10 servers allow us to handle 10x as many users?
  - load balance: what if some users get much more mail than others?
  - layout: what if we want to detect spam by looking at all mailboxes?

Topic: high availability
- Can I get at my HotMail mailbox if some servers / networks are down?
- Yes: replicate the data.
- Problem: replica consistency. delete mail, re-appears.
- Problem: physical independence vs communication latency
- Problem: partition vs availability. airline reservations.
- Tempting problem: can 2 servers yield 2x availability AND 2x performance?

Topic: global scalability
- this is really an opportunity

```
  we have the entire Internet as a resource
  what neat new big systems can we build that take advantage?
  are there any principles to be discovered?
    finding objects
    storing objects "out there"
    serving same objects to many consumers
    widely distributed computing (e.g. grid computing)

Topic: security
  old view: secrecy via encryption (msg to Moscow embassy)
            user authentication via passwords &c
            all parties know each other!
  Internet has changed focus.
  global exposure to random attacks from millions of bored students
    and serious hackers, e.g. intrusions for spam bot nets
  you fetch a new Firefox binary, how do you know it hasn't been
hacked?
  how do you know that was Amazon you gave your credit card number to?
  how does Amazon know it was you?
  no purely technical approach is likely to solve these problem

We want to understand the individual techniques, and how to assemble
them.

--------------

Course structure
  URL
  meetings: 1/2 lectures, 1/2 paper discussions
  research papers on working systems, starting next week
  must read papers before class
    otherwise boring, and you can't pick it up by listening
  we will post paper questions 24 hours in advance
  hand in answer on paper in class, one or two paragraphs
  two in-class quizzes (no final)
  Labs: build a real cluster file server, cache consistency, locking
  Project. look at the project information page!
    design, implement, report
    teams
    proposal
    conferences
    two drafts
    demo
    report
  Emil is TA, office hours TBA
  Look at the web site:
    sign up for course machine accounts
    look at the first lab, due in a week

--------------

O/S kernel overview
  context in which you build distributed systems
  o/s has big impact on design, robustness, performance
  sometimes because of o/s quirks
  mostly because o/s solves some hard problems
  This should be review for most of you
```

```
    Want to tell what I think is important
    Give you a chance to ask questions

What problems does o/s solve?
  sharing hardware resources
  protection
  communication
  hardware independence
  (everyone faces these problems)

Approach to solutions?
  o/s designers think like programmers, abstractions + interfaces

UNIX abstractions
  (we'll be programming UNIX in labs, my favorite O/S)
  process
    address space
    thread of control
    user ID
  file system
  file descriptor
    on-disk file
    pipe
    network connection
    device

All this is implemented by a "kernel" with hardware privileges

Note we're partially virtualizing
  o/s multiplexes physical resource among multiple processes
  CPU, memory, disk, network
  to share, to control, to provide a simple model to apps
  abstraction helps virtualization: easier to share TCP conns than enet

Can't completely virtualize
  file system and network stack not the same as physical foundation
  the differences make sharing possible

abstractions interact, must form a coherent set
  if o/s can start programs, it must know how to read files

System call interface to kernel abstractions
  looks like function call, but special
  fork, exec
  open, read, creat

Standard picture
  app (show two of them, mark addresses from zero)
  libraries
  -----
  FS
  disk driver
  (mention address spaces, protection boundaries)
  (mention h/w runs kernel address space w/ special permissions)

Why Big Kernels have been successful.
  easy for kernel subsystems to cooperate
```

```
    disk buffer shares phys mem with virtual mem system
  all kernel code is 100% privileged
    very simple security model
  easy to implement sophisticated and efficient services

Why UNIX abstractions are not perfect
  kernel is big
  kernel has room for lots of bugs; it's all privileged
  kernel limits flexibility
    multiple threads per process?
    single thread crossing into a different address space?
    control disk layout of files for performance?
    don't like the kernel's TCP implementation?
  we'll discuss a number of improved abstractions

Alternate set of abstractions: micro-kernel
  Move complex abstractions to server processes
  Talk to FS server, rather than FS module in kernel
  Kernel mostly handles IPC
    also grants h/w access to privileged servers
    e.g. FS server can read/write disk h/w
  Looks like a miniature distributed system!
  Move FS server to a different machine, via network?
  Lots of overlap with our concerns in this class.

Let's review some basics which will come up a lot:
  process / kernel communication
  how processes and kernel wait for events (disk and network i/o)

Life-cycle of a simple UNIX system call
  [diagram. process, kernel]
  See the handout...

Interesting points:
  protected transfer
    h/w allows process to get kernel permissions
    but only by jumping to *known* entry point in kernel
  process suspended until system call finishes

What if the system call needs to wait, e.g. for the disk?
  We care: this is what busy servers do
  sys_open(path)
    for each pathname component
      start read of directory from disk
      sleep waiting for the disk read
      process the directory contents
  sleep()
    save *kernel* registers to PCB1 (including SP)
    find runnable PCB2
    restore PCB2 kernel registers (SP...)
    return

Note:
  each user process has its own kernel stack [draw in diagram]
  kernel stack contains state of partially executed system call
  "kernel half"
  trap handler must execute on the right stack
```

"blocking system call"

What happens when disk completion interrupt occurs?
  Device interrupt routine finds the process waiting for that I/O.
  Marks process as runnable.
  Returns from interrupt.
  Someday process scheduler will switch to the waiting process.

Now let's look at how services use this kernel structure.

Explain server_1 web server in handout

Problem
  [draw this time-line]
  Time-lines for CPU, disk, network
  Server alternates waiting for each of them
  CPU, disk, network are each idle much of the time
  OK if only one client.
  Not OK if there are clients waiting for service.
  We may have lots of work AND idle resources. Not good.
  s/w structure forces one-at-time processing
  How can we use the system's resources more efficiently?