CS1003/1004: Intro to CS, Spring 2004

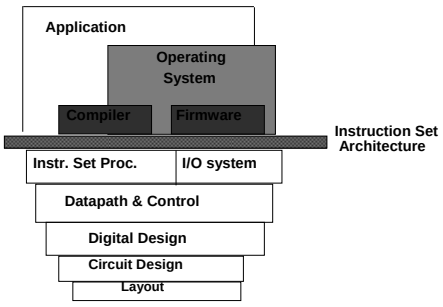
Lecture #12: OS & Networks

Janak J Parekh
janak@cs.columbia.edu

Administrivia

- Three weeks left in the semester!
- HW#5 due next Tuesday
  - If you have not started already… you'd better start today
  - Don't expect to write the programming up over a weekend
- Thanks to Suhit for teaching last week
  - How was he? ;-)

The big picture



The big picture (II)

- Given hardware and compiled (machine) code, you can run it directly, but that's a huge hassle
  - What if you want to run multiple programs?
  - If so, how do we share resources between programs?
  - How do we let the user manipulate various programs?
  - How do we let *multiple users* manipulate various programs?
- Solution: employ a special piece of software that allows multiple user applications/tasks to cooperate

History of operating systems

- *Batch processing:* back in the single-task days, people would submit jobs to the computer for the entire company, and wait in line for their job to be done
  - Used a *queue* abstraction to handle the job list
  - No interactivity – submit job, wait for results
  - Very cumbersome for iterative development
- Interactive processing
  - Allow the user to interact
  - Still had to wait for your shot to use the computer
  - Anyone remember DOS?
- Modern OSes *multitask*

Operating systems

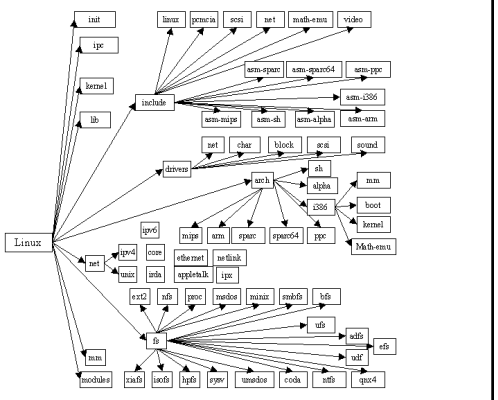
- Considered *system software*, as compared to *application software*
  - The latter run as *processes* alongside an OS
- Two major components:
  - A *kernel*, which handles resource management, multitasking, etc. in the background;
  - A *shell*, which provides a user frontend to the operating system

Kernels

- Several important components
  - *Device drivers:* used to enable the OS to communicate with computer hardware
    - Device drivers *abstract* the hardware away from the OS, so that you can "plug-in" new drivers
  - *Memory manager:* Keeps track of computer's memory allocation per process; also supports *virtual memory*, which enables the use of hard disks as additional memory
  - *Scheduler:* Control what tasks are running on the processor at any given time
  - *Network stack:* Provides networking facilities

The Linux kernel

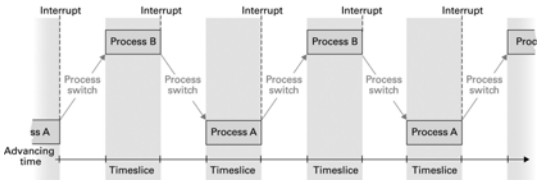
- Popular learning kernel, since it's open source
- You can grab your own copy from www.kernel.org, if you want to take a look
- A Linux operating system distribution (like Red Hat) consists of the *Linux kernel* and a bunch of tools (including GNU tools)
- Here's the directory structure of the kernel…



Multitasking

- Given multiple *processes*, coordinate them so that they can run concurrently
- Well, not concurrently – the CPU handles a fixed number of instructions at any given time
  - Instead, *timeslice*, so that each process does a little work at a time, and keep on switching
  - Operating system keeps separate register sets, etc. for each application, and magically handles them cleanly for you
  - "Virtual machine": As an application designer, you *feel* like you have control over the machine, but the OS is actually managing many such processes

Multitasking (II)



How do *you* multitask in UNIX?

- The "&" operator
  - "emacs &" starts up emacs as a *background process*
  - Lets you continue to use the shell while running emacs in its own window
  - "jobs" lists the currently running jobs in the background
- Or… multiple ssh sessions
- The machine is actually handling all of these user sessions in parallel as collections of processes
  - UNIX is *multiuser*, unlike older client versions of Windows

Multiuser and other trivia

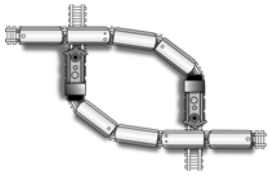
- By being multiuser, UNIX must worry about user accounts, passwords, and permissions
  - *root:* administrative UNIX account (like Windows "Administrator" user)
- "w" or "finger" will list the currently logged-in users on the current machine
  - Note that CUNIX is a *cluster* of machines, not just one machine
- "ps" lists the processes on a machine
  - "ps auxw" (Linux/BSD) or "ps –ef" (Solaris/SysV)
  - top lists most active processes on a machine
- "kill" kills a process

Process competition?

- What if two different processes need to access the same resource?
  - In the old days, if two programs want to print, you'd get a printout that was a mix of both
  - Now, a *print spooler* coordinates output and keeps them separate
  - The OS is responsible for handling such *race conditions* between processes

Process competition (II)

- More complicated resource contention requires *locking*; concept is similar to the barriers at a train track crossing
  - Semaphores == fancy locks
- Avoid *deadlock:*



Networks

- Now that we've discussed all the pieces on *one* computer, let's talk about networking computers together
- More and more computing solutions are *distributed* across networks
- Several different kinds:
  - LAN (Local Area Network)
  - WAN (Wide Area Network)

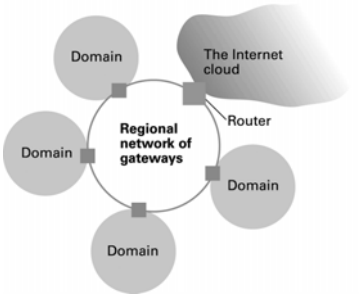
LANs

- Most common LAN architecture today is Ethernet
- 10BASE-T/100BASE-T Ethernet use telephone-like wire to network computers together
  - Very cheap, and popular ("CAT 5" wiring)
- *Topology:* how to organize these networks?
  - Typically a hierarchical star topology nowadays
  - Columbia's network is a hybrid of Ethernet and fiber

WANs

- Typically collections of LANs, with high-speed telecommunications links connecting them together
  - POTS (plain old telephone system): typically < 56kbps
  - DSL/cable: typically 128kbps-1.5Mbps
  - T1: 1.544Mbps
  - T3: 45Mbps
  - OC3: 155Mbps
  - OC12: 622Mbps
- Columbia has an OC3 to the commodity Internet
  - not enough…

The Internet



The Internet

- A very, *very* large WAN
- http://research.lumeta.com/ches/map/gallery/index.html
  - *Extremely* complicated
  - "The Internet has a diameter of 10,000 pookies"
- Active research as how to accurately map Internet topography
  - We just had a Ph.D. student come yesterday as a faculty candidate talk on this very topic

So how does the Internet work?

- On top of a series of *network protocols* that define how computers should talk to each other
- Internet Protocol (IP) is the most important
  - Current one (IPv4) was made over 20 years ago(!)
  - http://www.ietf.org/rfc/rfc0791.txt
  - Next version is IPv6: "coming soon"
- Describes how computers should be *addressed*, how to *route* between networks, and how to carry data

IP addressing

- IPv4: "dotted-quad notation"
  - Each machine has an address of the form xxx.yyy.zzz.www
  - Many "restricted" addresses
  - DNS (domain name service) maps a name to an IP address
  - chambers.psl.cs.columbia.edu → 128.59.14.155
- LANs typically have contiguous IP addresses
  - Columbia (wired): 128.59.*.*
  - Columbia (wireless): 160.39.*.*
  - We're getting slowly more fragmented
- *Routers* "route" packets between one LAN to another based on addresses and a "routing table"

IP "packets"

- A *packet* is a bag of data, typically up to 1500 bytes
- Contains some *headers* specifying things like source and destination, and some *data*
- The Internet is a "packet-switched" network
- TCP (Transmission Control Protocol) is one protocol that takes large amount of data to be sent and breaks them up into these small packets
- TCP/IP – the most common combination (RFC 793)
- I can take a look at the packets if I'm bored…

What services run on the Internet?

- E-mail: specified by its own protocols
  - SMTP (RFC 821, 2821) – Specifies how to transfer email from a source to a destination via a chain of mail servers
  - POP3/IMAP are simply *retrieval* protocols to retrieve your mail from a mailbox
- Web: two main standards
  - HTTP: Hypertext Transfer Protocol (RFC 2616)
  - HTML: Hypertext Markup Language
- Both work over TCP/IP
  - "Stacking" protocols on top of each other
  - *Port* abstraction to separate services over TCP/IP

Other services

- Telnet: simple text over TCP/IP
  - In fact, I can telnet to an HTTP server and talk HTTP or SMTP if I know how to
- FTP: File Transfer Protocol
- ssh: like telnet, but encrypted for security's sake
  - I can actually read the data typed over telnet or ftp using tcpdump… if I'm root or have control over a switch
- Others?
  - kazaa, AIM, MSN, you name it
  - Once you learn more, you can make your own

So how do you stay secure?

- Effective password management
  - Change your passwords every so often
  - Don't use your last name as the password
- Use secure protocols
  - These use *encryption*, which makes it difficult for a third-party
  - SSL, ssh are two of several out there
- Don't run random programs on your computer
  - Viruses and spyware can do network traffic communication behind your back, and convey your own data to other parties

What does this mean for you?

- OSes and networks are the context of all the work we do with computers nowadays
- If you program in the future, you'll likely have to interact with both in a more involved form
- Both C and Java have ways of communicating with the operating system and with other computers on LANs and the Internet, so you can write your own Kazaa's or webbrowsers…

Next time

- In labs:
  - C – more pointers and structs
  - Java – basic graphics programming
- ***Make sure to come to us with questions <u>this</u> week***
- Lecture: basic AI concepts