Lecture 1 - CS 402 Class Overview and Rules

Times and places

- Meets TTh 9:30-10:50
- My Office Hours are the hour immediately following class, SAL 232
- Generally available via email - faber@isi.edu
- Phone during business hours 310-822-1511 x190
- TA is TBA
- Class web page is `http://www.isi.edu/˜faber/cs402`

Classes

- Lecture twice a week you're responsible for what goes on here
- Classes are videotaped and available in Seaver Library
  - There is a lag
  - Be good to the librarians or you will lose your tape privleges.
  - be reasonable to others, or you will lose your tape privleges.
  - If you need your own copies, you can arrange it with ITV in advance. There is a charge.
- Classes are on ITV
  - remote students welcome
  - participate via phone
  - electronic fora
- Lecture Notes (mine) made available via the web page

Academic Integrity

- You must do your own work
- discussing problems/homework is fine, but you must write your own code/answers
- it is inifintely better to hand in your own bad code than someone else's code that you claim as your own.
- **If you violate the Academic Integrity Policy of USC, [1] you fail the class**
  - Read it, Learn it, Live it
  - If you have some question about how the policy applies, talk to me or the Office of Student Conduct [2]

Grading

- 40% project
- 35% final
- 20% midterm
- 5% homework

[1] `http://www.usc.edu/student-affairs/student-conduct/grad_ai.htm`

[2] `http://www.usc.edu/student-affairs/gateway/programs_services/student_conduct.html`

Project

- The project is demanding - don’t expect to do well in this course if you don’t devote time to it
- Unless you are a skilled, experienced programmer, you cannot do these assignments in a week-end.
- Overview
  - build a toy operating system (Nachos)
  - simulator to make this easier
  - we provide the base, you enhance it
- Prerequisites
  - Nachos is written in C++ - a C background will take you far, and learning C++ is good
  - Good programming skill are a plus, of course
- 2-3 person teams
  - teams can only be dissolved if they are immediately reformed into legal teams
  - teams all recieve the same project grades
  - team management is part of your grade
- Slip Days
  - You get 4. Use them on any assignment.
  - These will not save you if you get too far behind.
  - Each assignment is 3 weeks long.
  - **No extensions**
- Comments and documentation count
  - If your assignment is hard to grade, it will be graded down.
  - We do understand the difference between grammar difficulties and not commenting, or commenting the wrong parts
- We will read your code, and everyone else’s. **Do not copy code**
- There will be exam questions based on your experience with Nachos

Exams

- You have to come here to USC to take them, unless you’re on the list of sites exempt from that rule: e.g. Qualcomm, SD.
- Remote students are always welcome to come here to exams.
- My last Midterm and answers are on the web page
- Midterm will tenatively be 28 October 1999
- Final will be 14 December 1999
- No pop quizzes for points, but I like to ask questions

Text

- Tannenbaum’s Modern Operating Systems
- You’re responsible for the assigned reading on exams

Communications

- Web page http://www.isi.edu/˜faber
- Web form for your email address to get broadcasts

A Little History

This is a flying tour; Tannenbaum does a good job covering this information.

In the earliest computers, what we call the OS today was primarily a set of input/output routines so that scientists didn't need to learn the ins and outs of each new disk drive.

CPUs are expensive, and they spend a lot of time idle. Timesharing is born to share this expensive resource between multiple users. Still the systems are *batch* systems - users (physically) submit their jobs, they are queued and run overnight, and users receive output (physically). Multiple jobs take turns with the CPU swapping off after a given time or when the CPU does I/O. Sometimes mainframe computers have specialized "I/O processors," really separate computers, to run the peripherals. (More about this later)

Eventually (mid-1960's-1970's)computers become interactive - waiting overnight to get your compiler errors really sucks. Interactive timesharing brings more changes to the systems, not the least of which is user interfaces. With users appearing and disappearing, all the resource allocations become more dynamic. The increasing complexity of these systems contributed to the development of Software Engineering as a discipline.

The 1980's see widespread deplyoment of small computers. User interface becomes key. Multiprocessors become more common. The world has gone from many people per processor to many processors per person. Distributed applications are real (many computers working on the same problem), e.g., SETI.

As we'll see all of this leads to a complexity of OS that is interesting in it's own right, and worthwhile even if you don't program OS.

Definitions

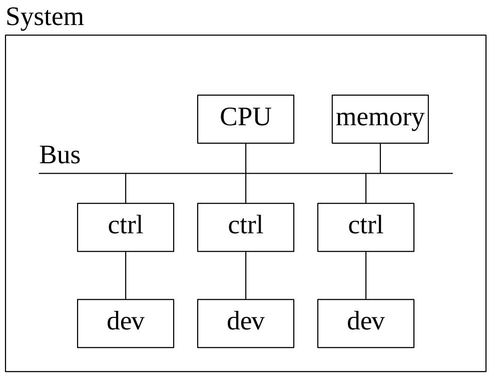
An Operating System provides

- Resource Allocation
- Virtual Machine (programming environment)

Huh? What resources? What's a Virtual machine?

Computer Organization

The OS is not an application program: it must manipulate the physical hardware of the machine. A computer consists of a CPU, associated processing units, peripherals and interconnection systems.



What are these things?

CPU

The CPU runs computations - solves differential equations, balances checkbooks, does ballistics for games. It's "the computer". It has some direct helpers:

- Memory management Unit (MMU) - controls memory accesses

- Floating Point Unit (FPU) - floating point calculations
- Caches - hold instructions or data close to the CPU
- Timers - time of day and alarm clocks

The CPU operates in 2 modes: *supervisor mode* (all instructions can be executed and all memory accessed) and *user mode* (restricted subsets of both). Most programs run in user mode. Changing to supervisor mode is only done under controlled circumstances.

The CPU can also be in *interrupt mode,* which we discuss below.

Interrupts and Traps

Interrupts are asynchronous events from other elements of the system (timers, disk drives). The interrupt structure of the machine lets the CPU determine what element caused the interrupt and how important it is. Common mechanisms are *interrupt vectors* (causes) and *interrupt priorities* (importance).

When an interrupt occurs the CPU

- stops executing code when the current instruction is done, and saves the address and state of the processor (registers, MMU state, etc)
- begins executing an *interrupt service routine* (ISR) in interrupt mode (privleges between supervisor and user mode).
- When the ISR completes, the processor restores state and returns to the code at the saved address.

Traps are software interrupts. They're used to access OS services.