Welcome to CS162!

About Your TA

Name: Angela Juang
Email: juang.angela+cs162@gmail.com or
cs162-ta@imail.eecs.berkeley.edu
Website: http://inst.eecs.berkeley.edu/~cs162-ta
Lab Sections: Friday 10-11am, 6 Evans
11-12pm, 4 Evans
Office Hours: TBD

Announcements

- Make sure you join the newsgroup! (ucb.class.cs162)
- You should be forming project groups – group signups next Tuesday
    - If your group ABSOLUTELY can't make more than one section, you need to send an email to cs162@imail.eecs.berkeley.edu with an explanation (please try not to do this!)
- Start looking at and tracing through Nachos code as soon as possible

Processes and Threads

We're used to being able to run programs on our computers, but how do operating systems manage all those programs? A process is what the OS uses to represent an executing program. Multiple processes can be executing at a time, so the job of the OS is to provide scheduling (so that all processes get CPU time) and protection (so processes don't crash or modify each other).

We can take care of protection between processes by using address translation. Address translation gives each process a distinct mapping to physical memory so that processes have their own address space and don't have access to another process' address space. (Processes can have shared address space too, though – this might be done for communication purposes.) The kernel also protects itself from corruption by user processes with dual-mode operation. With dual-mode operation, user processes run in user mode, while the kernel runs in kernel mode; important resources/commands are reserved for kernel mode. To access these resources, processes must use system calls to request a trap to kernel mode to service the request.

To deal with concurrency, each process is broken down into a collection of threads. A thread is a single stream of execution within the process; threads belonging to the same process can run concurrently with each other but do not need to be protected from each other. Therefore, memory and I/O state corresponding to the entire process are shared between threads and stored by the process control block (PCB), but individual stack pointers, registers, and other such information are kept private to the thread and stored in the corresponding thread control

block (TCB). The OS can give CPU time to different threads/processes by performing a context switch. With each context switch, the current state of the running thread/process will be saved into the corresponding TCB/PCB and the new one will be loaded in.

Mm, Nachos :9

As mentioned in lecture, Nachos is the operating system you and your groups will be working with this semester. You'll probably want to dedicate some of your time this week to reading through Nachos code and understanding how the existing code works so you can start building on it for phase 1. Here's a quick overview to help you get started:

- Packages you should not modify:
  - `nachos.ag`
  - `nachos.machine`
  - `nachos.security`
- Packages you should modify:
  - `nachos.threads` (phase 1)
  - `nachos.userprog` (phase 2)
  - `nachos.vm` (phase 2 & 3)
  - `nachos.network` (phase 4)
- Individual project directories

There is also a Nachos API Javadoc on the CS162 website, which may be helpful.

At this point, you will probably want to start with the following classes (in approximate order):

- `threads/ThreadedKernel`
- `threads/KThread`
- `machine/TCB`

You will also need to look at a number of other classes to do the first project, but these three classes should provide a good starting point to dive into the code. Good luck!