

# Announcements

## ■ Assignment 1 Out

- Your first assignment goes out today and comes due a week from Tuesday, just before midnight.
- One key difference between this first CS110 assignment and assignments in prior classes is that this code base is larger. Most of the `.c/.h` pairs define and implement a key layer contributing to the overall system. In this case, you really need to understand the vast majority of code that's given to you.
- Understand that [Sections 2.2 and 2.5](#) of the Salzer and Kaashoek textbook cover the theory backing Assignment 1.
- You may not use any late days on this first assignment, since we want to grade them as soon as possible and get you feedback so you don't make the same mistakes on Assignment 2.

## ■ Reading

- Read through [Sections 6.1, 6.2, and 6.3](#) of the Salzer and Kaashoek textbook, which discusses performance, caching, and multilevel memory hierarchies. This will help you out with Assignment 2, which goes out the day Assignment 1 is due.

## ■ Next week

- Come next week (probably on Wednesday), we'll start up on some (awesome) new material.
  - We'll discuss exceptional control flow and how it can be used to support asynchronous interrupts (so that, say, your mouse can signal the mouse driver when and only when it moves), faults (e.g. when a virtual address in your stack frame isn't yet mapped to physical memory, or when you dereference **NULL** because you still think that's okay for some reason ☺), traps (which are intentional calls in exceptional circumstances because the normal flow of code needs to halt while some system service runs to open a socket, create a new executable, or exit the program), and aborts (three guesses what that does).
  - The exceptional control flow material will also grant us the opportunity to write programs that spawn off and interact with other executables. It'll be our first foray into concurrency and parallelism, how it works, and what the OS needs to do to support it. It's also fascinating to know that it's possible in the first place.

# Today's Lecture

## ■ Agenda

- I need to work through one of the three remaining examples I posted in this [past Wednesday's slide deck](#).
- I want to describe the design and implementation of the Unix v6 filesystem, so that we're clear how blocks are the building blocks (har) of inodes, and that each and every file maps to precisely one inode. I want you to understand what an 18-byte file, a 2048-byte file, and 34471936-byte file looks like in memory.
- I also want to be clear how blocks work differently for us depending on whether their contents contribute to a regular file or to a directory. I'll finally want to draw the state of a small Unix v6 filesystem that contains a very small directory structure, where all leaf directories contain 3 regular files.
- Work through the **copy** and **filedump** examples posted last Wednesday. These two examples introduce the file descriptor and how it can be used to perform low-level I/O on files (and, eventually, over network connections).
- We'll also spend 10 or 15 minutes talking about how Domain Name Server (otherwise known as DNS) lookup works. I do so for two reasons:
  - DNS is a simple, fault-tolerant, distributed system that's easy to explain.
  - There are a good number of features common to both a Unix v6 file system and a DNS. That there are so many architectural similarities is no accident. Both rely on a hierarchy of human-oriented names (/afs/ir.stanford.edu/class/cs110/ and graph.facebook.com, for instance) that map to more computer-palatable names like device 1, inumber 49990, blocks 388821, 338291 for the path, 31.13.75.17 for domain name.
  - Realistically, I'm probably not going to get to the DNS material before Monday, but I can dream we'll get to it today.
- The fact the Unix v6 filesystem's design and DNS's architecture rely on naming and name resolution speaks to the widely-held belief that **naming in computer systems** is a **fundamental principle in system design**.