Principles of System Design

- **Let's take a step back and look at the big picture.**
  - I'm all about implementation-driven lectures. I prefer to rely on code to teach the material.
  - However, you also need to walk away from this course with an understanding of the basic principles guiding the design and implementation of large systems, be they file systems (your Assignments 1 and 2), multithreaded RSS feed aggregators (your Assignments 4 and 5), HTTP proxies (your Assignment 6), or MapReduce frameworks (which we'll talk about on Friday).
  - An understanding of and appreciation for these principles will help you make better design and implementation decisions should you take our more advanced systems courses.
    - CS140: Operating Systems, which has you design and *implement* processes, threads, virtual memory, and a much more robust filesystem than what you were charged with implementing for your Assignments 1 and 2.
      - CS110 emphasizes the client use of processes, threads, concurrency directives, and so forth. CS140 is all about implementing them.
    - CS143: Compiler Construction, which has you implement a pipeline of components that ultimately translates source code (not unlike C++) into an equivalent stream of assembly code instructions.
    - CS144: Computer Networking, where you study how computer networks (the Internet in particular) are implemented.
      - CS110 emphasizes the client use of sockets and the socket API functions as a vehicle for building networked applications. CS144 is all about understanding and (in some cases) implementing the various network layers that allow for those functions to work and work well.
  - *These slides are the result of a series of email exchanges and conversations with Mendel Rosenblum (CS110 and CS140 instructor) and Phil Levis (CS144 instructor).*

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
    - Separating behavior from implementation (trivial example: **`sort`** has one interface, but a billion different implementations).
    - Defining of a clean interface to a component or subsystem that makes using and interacting with a system much, much easier.
    - Examples of abstractions we've taken for granted (or will soon take for granted) this quarter in CS110:
      - **filesystems** (you've dealt with C **`FILE *`**s and C++ **`iostream`**s for a while now, and knew little of how they might work until we studied them this quarter). We did learn about file descriptors this quarter, and we'll soon leverage that abstraction to make other data sources (e.g. network connections) look and behave like files.
      - **processes** (you know how to fork off new processes now, even though you have no idea how **`fork`** and **`execvp`** work).
      - **signals** (you know they're used by the kernel to message a process that something significant occurred, but you don't know how they're implemented).
      - **threads** (you know how to create C++11 **`thread`**s, but you don't *really* know how they're implemented).
      - **HTTP** (you're just now learning how to use the protocol that networked computers often use to exchange resources such as text documents, images, audio files, and so forth).
  - Modularity and Layering
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
    - Subdivision of a larger system into a collection of smaller subsystems, which themselves may be further subdivided into even smaller sub-subsysytems.
    - Example: filesystems, which use a form of modularity called **layering**, which is the organization of several modules that interact in some hierarchical manner, where each layer typically only opens its interface to the module above it. Recall the layering scheme we subscribed to for Assignments 1 and 2:
      - symbolic link layer
      - absolute path name layer
      - path name layer
      - file name layer
      - inode number layer
      - file layer
      - block layer
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
    - Subdivision of a larger system into a collection of smaller subsystems, which themselves may be further subdivided into even smaller sub-subsysytems.
    - Example: filesystems
    - Example: **`g++`**, which chains together a series of components (which is, in a sense, another form of layering).
      - the **preprocessor**, which manages **`#include`**s, **`#define`**s, and other preprocessor directives to build a translation unit that is fed to...
      - the **lexer**, which reduces the translation unit down to a stream of tokens which are fed in sequence to...
      - the **parser**, which groups tokens into *syntactically* valid constructs, which are then semantically verified by...
      - the **semantic analyzer**, which confirms that the syntatically valid constructs make sense and respect the rules of the type system, so that x86 instructions can be emitted by...
      - the **code generator**, which translate your C++ code into equivalent machine code.
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
    - Subdivision of a larger system into a collection of smaller subsystems, which themselves may be further subdivided into even smaller subsubsysytems.
    - Example: filesystems
    - Example: **`g++`**
    - Example: computer networks, which rely on a programming model known as TCP/IP, so named because its two most important protocols (TCP for Transmission Control Protocol, IP for Internet Protocol) were the first to be included in the standard.
      - TCP/IP specifies how data should be packaged, transmitted, routed, and received.
      - The implementation is distributed across four different layers.
        - application layer (the highest layer in the stack)
        - transport layer
        - internet layer
        - link layer (the lowest layer in the stack)
      - We just learned the application-level API calls (**`socket`**, **`bind`**, **`connect`**, and so forth) needed to build networked applications (your Assignment 6 has you implement an HTTP network proxy, and your Assignment 7 has you build a small, distributed system).
      - CS144 teaches all four layers in detail and how each layer interacts with the one below it.
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
    - Subdivision of a larger system into a collection of smaller subsystems, which themselves may be further subdivided into even smaller subsubsysytems.
    - Example: filesystems
    - Example: **`g++`**
    - Example: computer networks
    - Example: **`Facebook`**, which includes so many modules and subsystems I can't even identify all of them. (Note that this is an example of a large system where some subsystems use layering [the caching layers come to mind] and others do not). Here are just a few of the large components I worked on during my time at Facebook.
      - huge number of servers
      - many, many databases
      - several caching layers (C++ implementation of Apache's APC, specialized memcached, TAO)
      - dedicated photo and video upload servers
      - customized photo storage system optimized to hold most file metadata in main memory and to minimize disk seeks
      - Linux kernels hyperoptimized to be fast at things important to Facebook
      - homegrown PHP-to-C++ compiler, so the servers are processes instead of PHP scripts
      - standalone PHP libraries of reusable UI components (buttons, scrollbars, menus, etc, all with the Facebook look-and-feel)
      - dedicated grid of machines in place to provide real-time, machine-learning-driven story selection for your News Feed
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution
    - Names provide a way to refer to system resources, and name resolution is a means for converting between human-readable names and machine-friendly ones.
    - We've already seen two examples:
      - Humans prefer absolute and relative pathnames to identify files, and computers work better with inode and block numbers. You spent a good amount of energy with Assignment 1 managing the discovery of inode numbers and file block contents given a file's name.
      - Humans prefer domain names like www.google.com, and computers work better with IP addresses like 74.125.239.51. We spent time in lecture understanding exactly how the human-readable domain name is converted to an IP address.
    - Other examples: the URL (a human-readable form of a resource location), the process ID (a computer-friendly way of referring to a process), and the file descriptor (a computer-friendly way of referring to a file [or something that behaves like a file—yay virtualization and abstraction!])
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution
  - Caching
    - Simply stated, a cache is a component—possibly in hardware, possibly in software— that stores data so that future requests can be handled more quickly.
    - Examples of basic address-based caches (as taught in CS107)
      - L1-level instruction and data caches that serve as a staging area for CPU registers.
      - L2-level caches that serve as a staging area for L1 caches.
      - A portion of main memory—the portion not backing the virtual address spaces of active processes—used as a disk cache to store pages of files.
    - Examples of caches to store results of repeated (often expensive) calculations:
      - Web browsers that cache recently fetched documents, provided the server is clear that the documents can be cached, either indefinitely or for a period of time.
      - Web proxies (much like the one you're implementing for Assignment 6) that cache static resources so that *other* clients requesting the same data can be served more quickly.
      - DNS caches, which hold a mapping of recently resolved domain names to their IP addresses.
      - memcached, which maintains a dictionary of objects frequently used to generate web content.
      - The four different caches in the Solaris file system layers
        - The Directory Name Lookup Cache stores vnode-entry-to-path-directory-lookup information.
        - The inode cache collects and maintains file metadata information stored more permanently in inodes.
        - The rnode cache stores information about the many nodes contributing to an NFS filesystem.
        - The buffer cache stores inodes and indirect block disk I/O (and is reminiscent of the type of caching work you implanted into your Assignment 2 submission).
  - Virtualization
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution
  - Caching
  - Virtualization
    - Virtualization is an abstraction mechanism used to make many hardware resources look like one. Examples include:
      - RAID, which aggregates many (often inexpensive) storage devices to behave as a single hard drive.
      - the **Andrew File System**, aka AFS, which grafts many independent, networked file systems into one rooted at **`/afs`**. You descend into various subdirectories of **`/afs`** with, in principle, zero knowledge of where in the world those directories are stored.
      - a web server load balancer, where hundreds, thousands, or even hundreds of thousands of servers are fronted by a much smaller set of machines that intercepts all requests and quickly forwards them to the server least loaded.
    - Virtualization is *also* an abstraction mechanism used to make a single hardware resource look like many. Examples include:
      - virtual-to-physical memory mapping, which allows each process to believe it owns the entire address space. We covered virtual-to-physical memory mapping in lecture a few weeks ago.
      - threads, where the stack frame of a single process is subdivided into many stack frames so that multiple threads of execution, each with their own context, can be rotated through in much the same way the scheduler rotates through multiple processes. (Threads, in fact, are often called **virtual processes**. A single process is made to look like many).
      - virtual machines, which are software implementations designed to execute programs as a physical machine would. Virtual machines can do something as little as provide a runtime environment for, say, a Java or C# executable, or it can do as much as run several different operating systems (Mendel Rosenblum's VMware comes to mind) on an architecture that otherwise couldn't support them.
  - Concurrency
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
    - We have a good amount of experience with concurrency already:
      - Multiple processes running on a single processor, seemingly at the same time.
      - Multiple threads running within a single process, seemingly at the same time.
    - When multiple processors and/or multiple cores are available, processes can truly run in parallel, and threads within a single process can run in parallel.
    - Signal and interrupt handlers are also technically concurrent programs. Program execution occasionally needs to be halted to receive information from the outside (the file system, the keyboard, the mouse, the Wacom tablet, the attached MIDI controller, or the network).
    - Some programming languages—Erlang comes to mind— are so inherently concurrent that they adopt a programming model that makes race conditions virtually impossible. (Other languages—I'm thinking of pure JavaScript—adopt the view that concurrency, or at least threading, is too complicated and error-prone to support).
    - You'll soon read about MapReduce as a programming model that employs multiple threads across multiple processors across multiple machines to process huge amounts of information and whittle that information down into useful, aggregated data sets.
  - Client-server request-and-response

Principles of System Design

- **CS110 touches on seven such principles:**
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response
    - Request/response is a way to organize functionality into modules that have a clear set of responsibilities.
    - We've already had some experience with the request-and-response aspect of this.
      - system calls (**`open`**, **`write`**, **`fork`**, **`sleep`**, **`bind`**, and so forth are all userland wrappers around a special type of call into the kernel... user space and kernel space are two separate modules with a very, very hard boundary between them).
      - HTTP, IMAP
      - NFS, AFS
      - DNS