Principles of Computer Systems
Winter 2021
Stanford University
Computer Science Department
Instructors:
Chris Gregg and Nick Troccoli

PDF of this presentation
What is this class all about?

- Principles of System Design: CS110 touches on seven big principles
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution
  - Caching
  - Virtualization
  - Concurrency
  - Client-server request-and-response
Lecture 01: Welcome to CS110: Principles of Computer Systems

- Let’s take a look a the first three of these and jump right in
  - Principles of System Design: CS110 touches on seven big principles
    - **Abstraction**
    - Modularity and Layering
    - Naming and Name Resolution

- Abstraction separates *behavior* from *implementation*.

```bash
$ ls -1
ARCHIVE.README
cgi-bin
final-tests
include
lecture-examples
lib
local
main.cgi
private_data
repos
samples
staff
tools
WWW
```

- Take a look at the result of the Linux file list command, `ls -l` to the left.
- There is a *file list*, but this is simply an abstraction
  - How are files stored on the computer?
  - If everything is a 0 or a 1 to a computer, there must be some translation, and abstraction.
  - There are an infinite number of ways to store files (your second assignment will investigate one!), but the behavior of a file system is well-defined.
Let's take a look at the first three of these and jump right in

- Principles of System Design: CS110 touches on seven big principles
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution

- Abstraction separates *behavior* from *implementation*.

- What kinds of things does an operating system designer need to think about to design a file system?
  - How are the files stored? Assuming that it is in *non-volatile* memory (e.g., a hard drive, or SSD), what is the actual low-level form of the file storage. Keep in mind that the files must be located when required!
  - What is the relationship between a file's name location and its data location? This can be very different!
  - Are small files stored differently than large files?
  - How are files deleted so that the space doesn't go to waste?
  - Can two filenames point to the same file?
  - Does file data share the same space on the disk as metadata? (think: Heap Allocator...)

- These are just some of the questions that must be answered. But, no matter how they are answered, the *behavior* of the system to the user should remain constant. There are many different varieties of Unix file systems, but this should be transparent to the user through the use of abstraction.
Lecture 01: Welcome to CS110: Principles of Computer Systems

- Let's take a look at the first three of these and jump right in
  - Principles of System Design: CS110 touches on seven big principles
    - Abstraction
    - Modularity and Layering
    - Naming and Name Resolution

- Abstraction separates behavior from implementation.

```
ls -1 -i
504030014 ARCHIVE.README
503231001 cgi-bin
503723839 final-tests
503186329 include
503185617 lecture-examples
503186393 lib
503186405 local
504453014 main.cgi
503231019 private_data
503192313 repos
503216939 samples
503216981 staff
503230523 tools
503185411 WWW
```

- If we add the `-i` flag for our directory list, we can dig a bit more.
- It turns out that files on a Linux system have an associated *inode*, which is a form of *modularity* called *layering*. For a computer, it is easier to keep track of a file with a number, but for a human, the textual name is better.
- This was a decision that the Linux file system designers made! It actually bridges the abstraction layer a bit because the user can access these inodes.
Let’s take a look at the first three of these and jump right in

- Principles of System Design: CS110 touches on seven big principles
  - Abstraction
  - Modularity and Layering
  - Naming and Name Resolution

- Abstraction separates behavior from implementation.

- The distinction between inodes and filenames is also an example of naming and name resolution.
- Given a file’s name (and, more concretely, its path), there has to be code that can figure out what the inode is that is associated with that particular path.
- This is non-trivial, especially if you want the lookup to be fast (which you do!)
Lecture 01: Welcome to CS110: Principles of Computer Systems: Instructors

- Chris Gregg (cgregg@stanford.edu)
  - Electrical Engineering undergrad Johns Hopkins, Master's of Education, Harvard, Ph.D. in Computer Engineering, University of Virginia
  - Lecturer in CS, teaching CS 106B/X, CS 107/107E, CS 110, CS208E, CS 298.
  - At Stanford since 2016, at Tufts prior, and high school teaching prior to that.
  - I love the CS 110 material!
    - It is challenging, yet interesting, and it is a new window into systems that you haven't yet seen in the CS curriculum. I guarantee that you will write programs of the sort you have not written before.
  - I love to tinker
    - I'm always happy to chat about Arduino / Raspberry Pi / iOS apps you are working on
Lecture 01: Welcome to CS110: Principles of Computer Systems: Instructors

- Nick Troccoli (troccoli@stanford.edu)
  - Computer Science undergrad and grad at Stanford
  - Lecturer in CS, teaching CS106, 107, 110
  - CS110 was one of my favorite classes!
  - Looking forward to meeting all of you!
Lecture 01: Welcome to CS110: Principles of Computer Systems: Instructors

To reach both Nick and Chris:

cs110-win21-instructors@lists.stanford.edu (also listed on the course homepage)

Please use this email if you need to contact the instructors for anything course related.
Companion Class: CS110A

- CS110A is an extra 1-unit “Pathfinders” or “ACE” section with additional course support, practice and instruction.
- Meets for an additional weekly section and has additional review sessions
- Will meet 10:30am-11:50am PDT starting week 2
- Entry by application - see the course website for details
Lecture 01: Welcome to CS110: Principles of Computer Systems

- Staff and Students
  - 181 students as of January 10, 2021
  - You should know C and C++ reasonably well so that you can...
    - write moderately complex programs
    - read and understand portions of large code bases
    - trace memory diagrams
  - You should be fluent with Unix, gcc, valgrind, and make as covered in CS107 or its equivalent.
- graduate student CAs
  - Nick, Patrick, Ella, Raejoon, Thea, Semir
  - The CAs will hold office hours, lead lab sections, and grade your work
Lecture 01: Welcome to CS110: Principles of Computer Systems

- Course Web Site: https://cs110.stanford.edu
  - Info about upcoming lectures, assignment handouts, discussion sections, and lecture slides
- Lectures
  - Except for today's lecture, all other lecture material will be recorded prior to class, and available on Canvas-Panopto Course Videos.
  - Each video will have a set of short quizzes that cover the material. The quizzes allow multiple attempts, and must be completed before regular lecture time. Each day's worth of lecture quizzes are weighted the same in aggregate (e.g., lecture 2's quizzes will account for the same amount of weight as all other lectures).
  - The recorded content will take approximately the same amount of time as an in-person lecture.
  - We will hold review sessions for the material covered in the videos on Mondays and Wednesdays during lecture time. There will be ample time for questions.
    - E.g., lecture 2 material has been posted and should be watched before Wednesday, and we will review it on Wednesday.
    - Review sessions are optional but we strongly suggest you attend them. They will be recorded.
    - Going forward, each Wed. we will post the videos/quizzes for the next Mon. + Wed.
Lecture 01: Welcome to CS110: Principles of Computer Systems

- Online Forum
  - Peer-collaborative forum: Ed Stem
  - **Best for** course material discussions, course policy questions or general assignment questions (DON’T POST ASSIGNMENT CODE!)

- Office Hours
  - Nick, Chris and the CAs will hold office hours throughout the week, full schedule coming soon, starting 1/12.
  - CA’s have been instructed to not look at code. Ever. (though this is relaxed a bit for the first assignment, only)
  - **Best for** group work, code/intricate debugging questions (with TAs only!) or longer course material discussions

- Contacting the instructors
  - We will publish an instructor email address for you to contact both Chris and Nick for course-related information. In the meantime, email us both at cgregg@stanford.edu and troccoli@stanford.edu.
  - **Best for** private matters (e.g. grading questions, OAE accommodations).
Two Textbooks

- First textbook is other half of CS107 textbook
  - "Computer Systems: A Programmer's Perspective", by Bryant and O'Hallaron
  - Stanford Bookstore stocks custom version of just the four chapters needed for CS110

- Second textbook is more about systems-in-the-large, less about implementation details
  - Provided free-of-charge online, chapter by chapter. Not stocked at Stanford Bookstore by design. You can buy a copy of it from Amazon if you want.
Lecture Examples

Lectures will be driven by slides and coding examples, and all coding examples can be copied/-cloned into local space so you can play and confirm they work properly.

- Code examples will be developed and tested on the myth machines, which is where you’ll complete all of your CS110 assignments.
- The accumulation of all lecture examples will be housed in a git repository at /usr/class/cs110/lecture-examples, which you can initially git clone, and then subsequently git pull to get the newer examples as we check them in.
Lecture 01: Welcome to CS110: Principles of Computer Systems

Lecture Slides
We'll try to make the slides as comprehensive as possible, but working with the code yourself is going to teach you more.

- They are not a substitute for watching lecture
  - We go off script quite a bit and discuss high-level concepts, and you're responsible for anything that comes up in lecture
CS 110 -- more specifically

- Five main topics (more detail in a few slides):
  1. Unix Filesystems
  2. Multiprocessing (multiple *processes* running simultaneously)
  3. Signal Handling (sending a *signal* to a process)
  4. Multithreading (multiple *threads* in a single process running simultaneously)
  5. Networking Servers and Clients

- There will be six assignments, with each assignment at least one week in duration
  1. C and C++ refresher
  2. Unix Filesystems
  3. Multiprocessing Warmup
  4. Multiprocessing: Stanford Shell
  5. Multithreading and ThreadPool
  6. Networking
Course Syllabus

- Overview of Linux Filesystems
  - Linux and C libraries for file manipulation: `stat`, `struct stat`, `open`, `close`, `read`, `write`, `readdir`, `struct dirent`, file descriptors, regular files, directories, soft and hard links, programmatic manipulation of them, implementation of `ls`, `cp`, `find`, and other core Unix utilities you probably never realized were plain old C programs
  - Naming, abstraction and layering concepts in systems as a means for managing complexity, blocks, inodes, inode pointer structure, inode as abstraction over blocks, direct blocks, indirect blocks, doubly indirect blocks, design and implementation of a file system
- Multiprocessing and Exceptional Control Flow
  - Introduction to multiprocessing, `fork`, `waitpid`, `execvp`, process ids, interprocess communication, context switches, user versus kernel mode, system calls and how their calling convention differs from those of normal functions
  - Protected address spaces, virtual memory, virtual to physical address mapping, scheduling
  - Concurrency versus parallelism, multiple cores versus multiple processors, concurrency issues with multiprocessing, signal masks
Course Syllabus

- Threading and Concurrency
  - Sequential programming, desire to emulate the real world within a single process using parallel threads, free-of-charge exploitation of multiple cores (two per myth machine, 12-16 per wheat machine, 16 per oat machine), pros and cons of threading versus forking
  - C++ threads, thread construction using function pointers, blocks, functors, join, detach, race conditions, mutex, IA32 implementation of lock and unlock, spinlock, busy waiting, preemptive versus cooperative multithreading, yield, sleep_for
  - Condition variables, condition_variable_any, rendezvous and thread communication, wait, notify_one, notify_all, deadlock, thread starvation
  - Semaphore concept and semaphore implementation, generalized counters, pros and cons of semaphore versus exposed condition_variable_any, thread pools, cost of threads versus processes
  - Active threads, blocked threads, ready threads, high-level implementation details of a thread manager, mutex, and condition_variable_any
  - Pure C alternatives via pthreads, pros and cons of pthreads versus C++'s thread package
Course Syllabus

- Networking and Distributed Systems
  - Client-server model, peer-to-peer model, telnet, protocols, request, response, stateless versus keep-alive connections, latency and throughput issues, `gethostbyname`, `gethostbyaddr`, IPv4 versus IPv6, `struct sockaddr` hierarchy of records, network-byte order
  - Ports, sockets, socket descriptors, `socket`, `connect`, `bind`, `accept`, `read`, `read`, simple echo server, time server, concurrency issues, spawning threads to isolate and manage single conversations
  - C++ layer over raw C I/O file descriptors, introduction to `sockbuf` and `sockstream` C++ classes (via `socket++` open source project)
  - HTTP 1.0 and 1.1, header fields, `GET`, `HEAD`, `POST`, response codes, caching
  - MapReduce programming model, implementation strategies using multiple threads and multiprocessing
  - Nonblocking I/O, where normally slow system calls like `accept`, `read`, and `write` return immediately instead of blocking
    - `select`, `epoll`, and `libev` libraries all provide nonblocking I/O alternatives to maximize CPU time using a single thread of execution within a single process
Course Expectations

Course Grade Breakdown:
10% Lecture Quizzes
65% Assignments
15% Assessments
10% Lab Section Attendance
Course Expectations

Programming Assignments (65%)

- 6 assignments - some are a single file, others are significant code bases to which you'll contribute.
  - You should *always* become familiar with the header files and the assignment handout before you start writing a single line of code.
- Late policy - every late day *potentially* costs you (read below why it's *potentially*)
  - If you submit...
    - on time: no penalty
    - up to 24 hours later: 90% cap
    - up to 48 hours later: 60% cap
  - No submissions accepted more than 48 hours late
  - **Exception**: first assignment must be submitted on time, no late days allowed
  - Extensions for exceptional circumstances must be approved by Nick and Chris. Please communicate with us! We are here to accommodate you as much as possible.
  - **Note**: you must get 50% functionality on each assignment to pass the class. If this becomes a concern for you, please reach out to Nick and Chris.
Discussion Sections (10%)

- You'll also sign up for an 50-minute section to meet each week, on Zoom
- Mix of theoretical work, coding exercises and development exercises using \texttt{gdb} and \texttt{valgrind}
- Submit your section preferences anytime between \textit{Thurs. 1/14 and Sun. 1/17}. They are \textit{not} first-come-first served. The link will be posted on the course website.
Assessments (15%)

- No traditional exams this quarter
- Instead: low-stakes, open book self-assessments with opportunities to take a pulse on how well you understand the topics taught so far.
- 3 assessments at the end of weeks three, seven, and eight (see the course schedule for exact dates).
- Short, timed, assessments and you will have a window in which to take them over a weekend. Details will be provided closer to the first assessment.
- 5% per assessment
- If you have testing accommodations, please email Chris or Nick as soon as possible.
**Honor Code**

- Please take the honor code seriously, because the CS Department does
  - Everything you submit for a grade is expected to be original work
  - Provide detailed citations of all sources and collaborations
  - The following are clear no-no's
    - Looking at another student's code
    - Showing another student your code
    - Discussing assignments in such detail that you duplicate a portion of someone else's code in your own program
    - Uploading your code to a public repository (e.g. github) so others can find it
      - If you'd like to upload your code to a private repository, you can do so on github or some other hosting service that provides free-of-charge private hosting
  - **Tutoring policy**: tutoring is **not** appropriate for help with work that will be submitted for a grade.
  - **Honor Code Video**: please watch to make sure you understand the Honor Code in CS110
Introduction to UNIX Filesystems

- You should already be familiar with the Linux filesystem as a user. The filesystem uses a tree-based model to store files and directories of files. You can get details of a file in a particular directory with the `ls` command:

```
cgregg@myth58:/~cs110/spring-2019/lecture-examples/filesystems$ ls
alphabet.txt  contains.c  copy.c  list.c  Makefile  search.c  t.c  vowels.txt
```

- You can get a more detailed listing with the `ls -al` command:

```
ls -al
total 23
 drwx------  2 cgregg operator 2048 Mar 29 12:33 .
 drwx------ 10 cgregg operator 2048 Mar 29 12:33 ..
 -rw-------  1 cgregg operator  27 Mar 29 12:33 alphabet.txt
 -rw-------  1 cgregg operator 2633 Mar 29 12:33 contains.c
 -rw-------  1 cgregg operator 1882 Mar 29 12:33 copy.c
 -rw-------  1 cgregg operator  5795 Mar 29 12:33 list.c
 -rw-------  1 cgregg operator  628 Mar 29 12:33 Makefile
 -rw-------  1 cgregg operator 2302 Mar 29 12:33 search.c
 -rw-------  1 cgregg operator 1321 Mar 29 12:33 t.c
 -rw-------  1 cgregg operator    6 Mar 29 12:33 vowels.txt
```

- With this listing, there are two files listed as directories (`d`), "." and "..". These stand for:
  - "." is the current directory
  - ".." is the parent directory

- The "rwx------" designates the permissions for a file or directory, with "r" for read permission, "w" for write permission, and "x" for execute permission (for runnable files).
Introduction to UNIX Filesystems

There are actually three parts to the permissions line, each with the three permission types available:

- `rwx` owner
- `r-x` group
- `r-x` other

In this case, the owner has read, write, and execute permissions, the group has only read and execute permissions, and the user also has only read and execute permissions.

Because each individual set of permissions can be either `r`, `w`, or `x`, there are three bits of information per permission field. We can therefore, use base 8 to designate a particular permission set. Let's see how this would work for the above example:

- permissions: `rwx r-x r-x`
- bits (base 2): `111 101 101`
- base 8: `7 5 5`
- So, the permissions for the file would be, `755`
Introduction to UNIX Filesystems

- In C, a file can be created using the `open` system call, and you can set the permissions at that time, as well. We will discuss the idea of system calls soon, but for now, simply think of them as a function that can do system-y stuff. The open command has the following signatures (and this works in C, even though C does not support function overloading! How, you ask? See here.):

  ```
  int open(const char *pathname, int flags);
  int open(const char *pathname, int flags, mode_t mode);
  ```

- There are many flags (see `man 2 open` for a list of them), and they can be bitwise or'd together. You must include one of the following flags:
  - `O_RDONLY` -- read only
  - `O_WRONLY` -- write only
  - `O_RDWR` -- read and write

We will generally only care about the following other flags when creating a file:

- `O_CREAT` -- If the file does not exist, it will be created.
- `O_EXCL` -- Ensure that this call creates the file, and fail if the file exists already
Introduction to UNIX Filesystems

- When creating a file, the third argument, `mode`, is used, to attempt to set the permissions.
- The reason it is "attempt" is because there is a default permissions mask, called `umask` (see here for some excellent information about `umask`), that limits the permissions. `umask` has a similar octal value to the permissions, although if a bit is set in the `umask`, then trying to set that bit with the `mode` parameter will not be allowed. The `umask` can be set with the following system call:

  ```c
  mode_t umask(mode_t mask);
  // see "man 2 umask" for details
  ```

- The return value is the old mask (the one that was already set).
- If you want to simply check the `umask` value, you must call the function twice. E.g.:

  ```c
  #include <stdio.h>
  #include <sys/types.h>
  #include <sys/stat.h>
  int main() {
    mode_t old_mask = umask(0); // set to 0, but get old mask as return value
    umask(old_mask); // restore to original
    printf("umask is set to %03o\n", old_mask);
    return 0;
  }
  ```

  `$ gcc show_umask.c -o show_umask$
  `$ ./show_umask$
  umask is set to 077
  `}

- This output means that the only permissions that can be set are for the user (rwx). The group and other permissions can not be set because all three bits of their respective permissions are set in `umask`. 

Introduction to UNIX Filesystems

Today's lecture examples reside within `/usr/class/cs110/lecture-examples/filesystems`.

- The `/usr/class/cs110/lecture-examples` directory is a git repository that will be updated with additional examples as the quarter progresses.
- To get started, type `git clone /usr/class/cs110/lecture-examples cs110-lecture-examples` at the command prompt to create a local copy of the master.
- Each time I mention there are new examples (or whenever you think to), descend into your local copy and type `git pull`. Doing so will update your local copy to match whatever the master has become.
Introduction to UNIX Filesystems

- You can override `umask` if you need to set the permissions a particular way.
- The following program creates a file and sets its permissions:

```c
#include <fcntl.h>    // for open
#include <unistd.h>   // for read, write, close
#include <stdio.h>
#include <sys/types.h> // for umask
#include <sys/stat.h>  // for umask
#include <errno.h>

const char *kFilename = "my_file";
const int kFileExistsErr = 17;

int main() {
    umask(0); // set to 0 to enable all permissions to be set
    int file_descriptor = open(kFilename, O_WRONLY | O_CREAT | O_EXCL, 0644);
    if (file_descriptor == -1) {
        printf("There was a problem creating '%s'!
" , kFilename);
        if (errno == kFileExistsErr) {
            printf("The file already exists.
");
        } else {
            printf("Unknown errno: %d\n", errno);
        }
    }
    return -1;
}

$ make open_ex
cc     open_ex.c   -o open_ex
$ ./open_ex
$ ls -l my_file
-rw-r--r-- 1 cgregg operator 0 Mar 31 13:29 my_file
```