CS111

Discussion 1D - Week 1
TA: Shaan Mathur (he/him)
shaan@cs.ucla.edu
I’m still awaiting confirmation, but my office hours are tentatively set on Tuesdays from 1-3pm at Boelter 3256S (at Board B). If this doesn’t work, we can talk about it.

**Important:** The CS111 TA’s are solely responsible for the projects, not course material. The professor is in charge of all course material, and I’m not allowed to answer questions that don’t pertain to projects.
A Compulsory Slide about Myself

- I got my Computer Science B.S. from UCLA last Spring 2019. I took this class!
- I’m currently an Computer Science M.S. student through UCLA’s ESAP program (guaranteed admission if you meet certain requirements, check it out if interested!). This is my first discussion as a TA!!
- I’ve chosen the M.S. thesis option, where I’m working on private deep learning. In a nutshell, you can perform deep learning predictions on inputs that are obfuscated; moreover you can train a network on data that is obfuscated!
- I want to pursue a Ph.D. to become a professor at a research university!
Let’s be real. Why Come to Discussion?

- I really think we can make discussion productive for everyone. The hard part is that people have varying levels of understanding of the projects: some want to learn the fundamentals, others want specific help.

- I think the following format is a happy medium. Roughly the first half of class we will talk about how to do the project. Then after we will have everyone work on their projects so we can all help each other do it!
Questions?
Project 0
open and creat

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

- open is a system call that requests access to the file specified by `pathname`, whose access permissions are specified by `flags`. For example, if I just want read access to a file, I would use `open(pathname, O_RDONLY)`, which returns a file descriptor (an integer) that abstractly represents that file.

```c
int creat(const char *pathname, mode_t mode);
```

- creat is a specific call of open where the flags are set to `O_CREAT|O_WRONLY|O_TRUNC` (create file if it doesn’t exist, write only access, and make the file empty if it exists and you have write access). The mode argument specifies the read/write/execute access for user/group/other. A simple way to give everyone read/write access is to set mode to 0666.
close, dup, and File Redirection

```c
int close(int fd);
```
- close takes a file descriptor and asks the OS to close the corresponding file. If it returns 0, success! If it returns -1, then the global `errno` is set to the corresponding error.

```c
int dup(int oldfd);
```
- dup returns a second file descriptor for the file referred to by `oldfd`; moreover this second descriptor is the *lowest nonnegative file descriptor available*. This is useful for file redirection! Say we have input file descriptor `infd`, and we do the following: close keyboard input by `close(0)`, call `dup(infd)` so that file descriptor 0 now references the input file, and then `close(infd)`. We’ve successfully rerouted STDIN (file descriptor 0) as the input file!
read and write

ssize_t read(int fd, void *buf, size_t count);

- read attempts to read count bytes from the file pointed at fd and places it in buf. count is an upper bound; the return value will specify how many bytes were actually read (and how many bytes forward the file offset has been incremented). You want to keep reading into a buffer and writing that buffer to STDOUT/output file until the return value is nonpositive (0 means you’ve reached the end-of-file, -1 is an error and errno is set accordingly).

ssize_t write(int fd, const void *buf, size_t count);

- write attempts to write count bytes pointed to by buf into the file referenced by fd. It returns the number of bytes actually written, or -1 if there’s an error and errno is set accordingly.
exit and signal

void exit(int status);

- exit simply forces the process to exit and tells the OS the exit code status. It is convention that 0 denotes “exited successfully.” The project spec details specific exit codes for specific issues (bad argument, cannot open input/output file, caught segmentation fault)

sighandler_t signal(int signum, sighandler_t handler);

- signal tells the OS that if the signal corresponding to signum occurs, invoke the function handler, which is given the (single) argument signum. To tell the OS to run some function foo(int sig) on a segmentation fault, invoke signal(SIGSEGV, foo)
getopt_long

static struct option long_options[] = {
    {"create",  no_argument,  0, 'c' },
    {"file",    required_argument, 0, 'f' },
    {0,         0,                 0,  0 }
};

while((c=getopt_long(argc,argv,"cf:",long_options,NULL)) != -1) {
    switch(c){
    case 'c': // create
        break;
    case 'f': // file
        //global variable 'optarg' set to the corresponding argument
        break;
    default: // unknown, error!
        exit(1);
    }
    // Done with argument! Loop to get next arg (or exit if c == -1)
}
First, **make sure you compile your program with -g** (this compiles your program with debug symbols attached to the executable, so gdb has knowledge of the source code). To start gdb, run “$ gdb ./lab0”. You then have the following commands at your disposal.

- **b <function or line number>** - sets a “breakpoint” so that execution of the program halts when the function or line is reached
- **r <arg1> <arg2> …** - **run** the program from the beginning with given args
- **n** - execute the **next** line in the source code
- **c** - **continue** execution until hitting a breakpoint or the program terminates
- **bt** - print the **backtrace** (i.e. the current call stack)
- **p <variable or assignment>** - **print** the value of a variable (if you write an assignment like “p i = 2” then it will set the variable i to 2 in your program!)
- **info locals** - print all **local** variables in scope
- **l** - **list** out the lines surrounding the current line of execution
gdb demo!