CS032 2009 Lab 09

Systems Programming

While you were working on Fileview, you were learning C but you really didn't know what was going on behind the scenes. As it turns out, most of the functions you will use over your C careers are abstractions upon a set of highly legit functions called system calls, or syscalls for short. These are the functions that interact with the operating system itself, and they are the basis of all systems programming. You have all seen malloc and free, and those are software level abstractions to deal with the memory. You've probably used printf, which is just really slick software built on top of the read and write syscalls. Readdir and stat? Yep, you just talked to the file system.

Abstractions like printf and scanf are all well and good, and if you do any software engineering in C, you'll probably use them. It's the same reason you generally use HashMap in Java: it's fast, it's easy, and everything is taken care of. But in order to earn the right to use HashMap, you probably needed to waste a bunch of time on ArrayList and LinkedList. The same principle applies here. Even if you never use the low level functionality of C, you need to understand it. And fork doesn't have a software analogue, so you'll need to know how it works anyway if you want to be the most leet haxor in history omg.

1 Launcher

You will be writing a smaller version of an old project called Launcher. You will create a shell from which you will launch programs. This will be done using a combination of read(), write(), fork(), execve(), waitpid(), kill(), and more.

It's not quite a shell, because that's the first assignment for CS167, but it is close. Shell will require you to do your own string manipulation without strtok, account for arguments, and handle terminal redirection, all while using only one input buffer. Launcher will focus on simple programs with no arguments. If you want, you can handle arguments, but it's not like you'll get extra credit and you already did some of that with scanf in fileview. If you want to learn strtok, see a TA or read the man page.

1.1 Your Mission

Your shell will have a loop that waits for input. The input will be the full path of the program you want to run. If you are going to launch a program, you must keep track of its pid, because if Ctrl+C is pressed at any time, you must handle this signal and kill the running process, terminating launcher at the end. How you do this is up to you[1].

1.2 Support Code

You have two hours to do this lab. We do not expect a robust program with error checking and full functionality, but we expect you to demonstrate the functionality of read, write, fork, exec, wait, and signal. We've provided a stencil, but the rest is up to you. We've also provided write, because it is unbelievably tricky to get working. Originally, the lab would have been more like an actual shell with the ability to run numerous programs at once, but it was gonna suck. Trust me. I wrote it.

2 Syscalls

It doesn't make sense to go into detail on all the system calls you'll be using, because the man pages actually do a pretty good job explaining how they work. Sometimes there are subtleties to the man pages, which we will clarify here, but you'll be doing most of the research.

[1] Sorta. Use signal(). Sigaction is annoying.

2.1 read/write

Think of a file descriptor as a numerical handle that refers to an open file or a stream. In this case, youll be using stdin and stdout as your input and output streams, which are referred to by file descriptors 0 and 1.

2.2 execve

This will not return, because it replaces the address space of the program with the address space of the executed program. In order to make sure it doesn't clobber launcher, you must fork first.

2.3 fork

This is the most important function ever. Without fork, there would be no operating system. Fork is pretty complicated, but in essence, it duplicates the current process. In the spawned process, fork returns 0; in the parent, it returns the pid of the child.

2.4 signal

See fileview

3 Checklist

- Understand the system calls
- Implement Launcher
- Show a TA
- Explain why you don't need to add the process to the list in execute_w
- Take CS167[2]

[2]You'll still get credit for the lab if you don't, but I won't be happy about it. That, and your CS career is all leading up to it so you will have wasted your life. No big whoop.