CIS 415 Operating Systems

Assignment <#> Report Collection

Submitted to:

Prof. Allen Malony

Author:

<Jacob Rammer>

Report

Introduction

The purpose of this project was to work with OS signals and use them to trigger forked processes to run. In this project, we were provided an input text file where each line represented a command. That command could be then ran in the terminal. The input file could also include invalid commands, so I had to deal with error checking and exiting the process appropriately if this was the case. In part 1, we were just supposed to fork each of the commands and wait for them to exit. I am using waitpid() for this. In part 2, I forked each command, set them to wait for SIGUSR1 and then stopped them and started them. In part 3, it's the same for part 2 except I held the processes in a waiting state and scheduled them to run one-by-one for one second. Part 4 is exactly the same as part 3, except now I was printing info form /proc to the terminal.

Background

I spent a few days before starting this project just looking up how to use signals and working examples. In my lab 5, I used sigaction to handle my sigwait portion. Further on in my project for the alarm portion for scheduling, I used signal. I was told by someone else that it's better practice to use sigaction, and using them together could possibly create issues. In this project I improved my string parsing substantially by creating a separate function to remove newline characters, rather than re-writing it 100 times in my main.

I think for the most part, I think I did a better job on this project than the first one. In the first project I had a lot of repeated code, in project 2 I tried to minimize it and think a little before I wrote something. There aren't as many examples of how to use signals, so I tried to combine them the best I could to create something that worked.

Implementation

As in project 1, I used the strtok_r save pointer to my advantage here. In my top printing function, I was able to get rid of the information category and just keep the process information using strtok_r on ":". So, for example, Name: VSCode" became "VSCode" and I was able to print that out to the terminal. I don't think I did anything really fancy in this project, instead I played it safe and was more concerned about getting it working. The biggest challenge was in part 3 and 4 figuring out how to schedule one process at a time. After I got that figured out, in part 4, I printed out all the running processes after the alarm was sounded.

This brings me to one issue I do have in my part 4: formatting. When I tested everything, I was only using iobound to check to see that my topPrint function was working. After I started to run more processes such as cpubound, some of the tabbing got a little messed up. I spent a few hours today trying to get my topPrint to format correctly, but I was unsuccessful. Also, now that I'm thinking about it, I did notice some commands such as ls sometimes had unititialized data printed to the terminal from the topPrint function.

Example: memcheck-amd64- 10033 10033 0 0000000000000000

My best guess here is that that process had finished executing right before info was grabbed to be printed, leaving it to print a bunch of 0's. As I'm just pulling info directly from a file, I don't believe this is my fault / there's anything I can do about it.

Now to talk about scheduling, this will be a pretty quick section. Essentially what I did for scheduling is just loop through my pid_t* childArray and if the process was valid, I would run it for one second, run the topPrint function, and by this time the alarm should have been sounded. After the alarm was sounded, I stopped the current running

process and chose the next one in sequence, ran topPrint on the child array. Repeat. My topPrint showed the process name, state, Pid, PPid, and VMSize.

Performance Results and Discussion

I did not measure any of my runtime for this program as there are no time constraints.

Conclusion

I learned that signals are a pain to use if you have no idea what you're doing. Which at the beginning of this project, I didn't. Now I have a better understanding on how they work, and it's pretty cool. I've always been curious on how to run a specific function in a certain time interval, and now I know how to.