CSE 320 Spring 2019 HW #3 Part III

April 1, 2019

Deadline: April 7, 23:59 Stony Brook time (Eastern Time Zone)

1 Introduction

The goal of this homework is to get your hands dirty with more C programming but more importantly, to help you to familiarize yourself with such concepts as creating new processes using `fork()`, using *signals* and how to block them, and how to run programs using `execve()`. This homework consists of three parts.

In the third part, you need to implement a relatively small program that will combine various concepts that you have seen in the applications from the second part of this homework.

2 Programming Assignment

In this part of the homework, you need to implement an application called `artist_manager`. This application imitates hiring/firing artists and tracking which of them are busy doing work or available for work. You will need to implement this application using the concepts we have gone through in the recent lectures and that you were using in the previous part of this homework.

Your program should start with the shell and please put a shell prompt to make it more visible, e.g. "> " or "shell> "will do just fine. Your program should support the following list of commands:

- help
- date
- hire N
- fire X
- fireall
- assign X
- withdraw X

- list
- exit

Let's now discuss what each command should do.

help. This command should print some reasonable help message describing available commands. You don't have to produce a huge print out as something concise will be enough. You can also check the examples of help messages of the Linux tools to get an idea of how such a message may look like.

date. This command should print the current date by executing "date" application, which absolute path is usually "/bin/date". After executing it, your program should return back to the shell prompt.

hire N**.** This command imitates hiring N artists that may be assigned to some jobs later. Code-wise, you need to spawn N child processes that will keep running and do nothing (think of infinite loop inside of their code) unless some specific events described later will happen. Please note that this command can be called multiple times.

fire X**.** This command imitates firing a given artist X. By X we will understand PID of one of the child processes. Please note that this command can be called multiple times.

fireall. This command imitates firing all currently hired artists.

assign X**.** This command imitates assigning a given artist X to a job whereby X we will understand PID of one of the child processes. When this happens, the child process with PID equals to X should print "ARTIST X IS ASSIGNED TO A JOB" where you need to substitute X with the PID of that child process.

withdraw X**.** This command imitates withdrawing a given artist X from job whereby X we will understand PID of one of the child processes. When this happens, the child process with PID equals to X should print "ARTIST X IS WITHDRAWN FROM A JOB" where you need to substitute X with the PID of that child process.

list. This command should list all the current artists' PIDs and statuses in the following form:

PID STATUS

where the PID is a PID of a child process and status may be either "ASSIGNED" or "WAITING". What data structure to use to track that information is up to you.

exit. This command exits the shell and your program. Don't forget to free memory/data structures, close files, etc.

For most of the commands, after each command, your program should return back to the shell prompt. Let's go now through the internals of some commands. Firing an artist means that you need to gracefully terminate respective child and clean up associated data structures. Please note that all code for manipulating data structures to track artists and their statuses should be located in the `artist_ds.c` file. If you are about to fire an artist who is assigned a job then you need to withdraw the artist from the job first and then fire. If for some

reason child process died by itself (simulating an artist that decided to leave your company) then you need to possibly withdraw that artist from a job and then clean respective data structures.

For communicating with children processes you need to use signals. To assign a job to an artist you need to send SIGUSR1 signal and to withdraw from a job you need to send a SIGUSR2 signal to a particular child process.

3 Printing Output

To make the grading process faster and easier we introduce a **REQUIREMENT** how you should produce output in your program for anything except your shell prompt. Please also don't print anything in your final version of submission, which was not specifically asked as it may severely affect your grade.

We provide you with a function `cse320_print(char *message)`, which will print given message to the `stdout`. During the grading, we will modify this function to test your results by performing the exact string comparison. It also means that it is important to develop your code inside the VM to avoid possible differences between newline characters between different operating systems. Basically, that function just prints a given text.

Please note, that `printf()` is a formatted print function. Thus, if you need to print formatted string then you may need first to construct it and then pass the resultant string to provided `cse320_print(char *message)` function.

So far we only saw functions that have a strict number of parameters. In programming languages such as *C++* and *Java* you have an option of having functions with the same name but the different number of parameters. This technique is called *function/method overloading* but it is not supported in *C*. But in the case of `printf()` function, the number of parameters is not just unknown but also may vary drastically. Thus, you may become curious how does functions like `printf()` and `scanf()` work. In short, there is a special class of functions called *variadic functions* that accepts a variable number of arguments. Please refer to the readings at the end of the document to find more information about these functions.

4 Implementing/Testing Your Program

When you write your code try to split it into some logical blocks/steps. So every time such piece of code is complete you can test it and once testing is done, push it to the GitHub. Trying to complete the whole program all at once and debug only after will require much more efforts and will negatively affect the amount of time being spent on this homework.

It is also a good idea to write documentation for your program and put the important decisions about your design into the `README` file.

5 Requirements

You need to follow the naming conventions and the asked way to implement these exercises. If you are asked for some output then you need to print only what was asked and nothing else. Otherwise, your grade may be severely affected. Please find the list of the requirements for this part of the homework below:

- You need to use `cse320_print(char *message)` for any output except shell prompt.
- You should print only what was asked and nothing else.
- The `Makefile` should be located in the root directory of your repository.
- The executable should be called `artist_manager`.
- We should be able to compile your program by typing "`make`" in the root directory of your repository.
- All source code files should be in the "src" folder.
- All header files should be in the "inc" folder.
- You should not include "*.c" files into other "*.c" files.
- All code for manipulating data structures to track artists should be located in the `artist_ds.c` file.

And, similar to the previous assignments, no zombies, no memory leaks/errors, no crashes.

6 Submission

Please click on the link below that will set up a repository for you for this homework. In case, it will take more than a few minutes please contact me so I can set up the repository for you manually.

https://classroom.github.com/a/KrHM1LpH

For the second part please create a folder "part_2" and put all files related to the Part II into that folder.

As before, we grade the latest submission.

7 Interesting Readings

Links below are not necessary to complete this homework but you may find them interesting.

https://linuxconfig.org/bash-prompt-basics

https://en.wikipedia.org/wiki/Function_overloading

https://en.wikipedia.org/wiki/Variadic_function
C functions with variable number of arguments
printf() and scanf() examples

8 Extra Credit I

In our homework when we need to fire an artist who is currently assigned to a job we will withdraw the artist first. Let's modify this behavior now. Instead of firing right away you need to introduce a timer that will check every 5 seconds if that artist was withdrawn from a job and fire the artist only then. You may find SIGARLM useful for this extra credit.

9 Extra Credit II

You can see that every time we assign a job we assign it to a particular artist. However, we lack the functionality of just assigning the job and system being able to assign it to the first available artist. Let's fix it by creating a new command "`assignjob`". This command should pick the first available artist and assign a job to that artist. If all of the artists are assigned jobs then "assignjob" command should hire one to do this job.