COSC 315: Operating Systems - Project 1

Due: Wednesday, Feb 5 @ 11:59 pm

The purpose of this mini project is to give gain some experience working with system calls and learn about process management (process creation, termination etc). In this mini project, you will write a C program making use of several UNIX system calls. You will also need to write a design document explaining how your program is organized (details below).

Clone Shell in C

The goal of this lab is to implement a simple shell in C, which will require you to use system calls directly. In class, we touched on how a few system calls (notably fork and exec) can be used to implement a command-line shell like Bash. In this exercise, you will implement **closh** (Clone Shell), a simple shell-like program designed to run multiple copies of a program at once.

Like any other shell, closh takes as input the name of the program to run (e.g., hostname, pwd, echo, etc.). However, closh also takes three additional inputs:

1. The number of copies (processes) of the program to run. This is an integer from 1 to 9.
2. Whether the processes should execute concurrently or sequentially.
3. Optionally, a timeout (also an integer from 1 to 9) specifying the maximum duration of each process in seconds (reset between processes if running sequentially). If a process takes longer than the timeout, it is terminated. A timeout value of zero specifies no timeout.

Closh executes the given program the specified number of times, then returns to the prompt once all processes have either completed or timed out. Here is a simple example of using closh:

```
anarayan@ubuntu-ubco$ ./closh
closh> hostname
  count> 3
  [p]arallel or [s]equential> p
  timeout> 5
elnux2
elnux2
elnux2
closh>
```

Additionally, each new process your program creates should print its process ID before executing the command, as well as any other output you would like that demonstrates how your program is operating.

We will provide a program skeleton (closh.c) that implements all required parsing and interface logic. The skeleton simply executes the given command once and exits. Your task is to replace this single system call with the real process logic of closh.

Run the following commands on your own machine to get the source code.

```
$ tar xzvf closh-starter.tar.gz
```

Tips

- Useful functions and system calls include fork, exec (specifically the execvp variant, in conjunction with the cmdTokens variable), sleep, waitpid, and kill. You should use the SIGKILL signal value in kill to terminate a process.
- If you're trying to use waitpid and get a warning like "warning: implicit declaration of function 'waitpid'", you probably need to include an additional system header file. Add the line "**#include <sys/wait.h>**" to the top of your file alongside the other #include statements.
- Be careful when adding calls to fork -- if you write an infinite loop with a fork in it, a fork bomb will result. If in doubt, you can add a sleep(1); before each fork during debugging, which will slow the rate of process creation.
- Closh can execute a program with arguments, but cannot execute multiple programs using Bash constructs (e.g., 'sleep 3 && echo hello' to sleep for 3 seconds, then print hello). However, you can accomplish the same by making a new Bash file (e.g., the included 'sleephello' script) and calling that from within closh (e.g., './sleephello'). This is useful for testing that your program correctly handles both parallel and sequential execution. If you do this, make sure the script you are trying to call is executable ('chmod +x sleephello').
- If you have difficulties with C syntax or errors, email or see the TA. While minimal features of C are required for this assignment, we don't want you to spend too much time debugging issues with C itself.

Design Document and Observation

Once you have written your version of the shell, write a design document that documents your design choices.

How to Turn in Miniproject 1

All of the following files must be submitted on Canvas as a zip file in the form GroupName.zip to get full credit for this assignment.

1. Include a copy of all source files.
2. Include a README file containing an outline of what you did. It should contain the names of all your group members and contributions of each of you. It should also explain and **document your design choices**. Keep it short and to the point. If your implementation does not work, you should document the problems in the README, preferably with your explanation of why it does not work and how you would solve it if you had more time. Of course, **you should also comment your code**. We can't give you credit for something we don't understand!
3. Finally, include a file showing sample output from running your program.
4. **Note:** We will *strictly* enforce policies on cheating. Remember that we routinely run similarity checking programs on your solutions to detect cheating. Please make sure you turn in your own work.

   You should be very careful about using code snippets you find on the Internet. In general your code should be your own. It is OK to read tutorials on the web and use these concepts in your assignment. Blind use of code from web is strictly **disallowed**. Feel free to check with us if you have questions on this policy. And be sure to CLEARLY document any Internet sources/ tutorials you have used to complete the assignment in your README file.

Miniproject 1 Grading Scheme

(max 100) Total Grade

- (30) use of fork and exec
- (15) parallel and sequential execution
- (15) process timeouts
- (10) code structure
- (10) code comments
- (20) design document

Late Policy: Miniproject 1 is due at 11:59 PM on Wednesday, February 5. Please refer to the course syllabus for late policy on lab assignments. This late policy will be strictly enforced. Please start early so that you can submit the assignment on time.