Trent University
COIS3380H
Lab 4
Due: March 9th, 2023

Lab #3 - The SHELL game

In this lab, you will demonstrate your knowledge of process creation and manipulation. The objective of the lab is for you to create your own "shell". When you log into Loki, you are running the bash shell. It will recognize a number of different "verbs" and using new processes, will run those commands and display their output on your screen.

Your solution for lab #4 will be to create a C program which will accept a number of fixed verb names of your choosing. These will be replacements for variations of normal bash commands. Your program is to recognize the verb requested **and create a process** to run the equivalent bash command. To create the process you MUST use the fork() function and have the parent process wait() for it to complete.

Within the child process you MUST use one of the **exec*??*()** family of system calls to execute the required bash verb with your selected parameters. You are not allowed to call aliases or functions defined in your .bashrc (ask me why!).

As an example, in my version I have created a verb called "jobtree". When I run my code and type in "jobtree" at the prompt, it will produce the equivalent to the shell command:

```
ps -o user:32,pid,stime,tty,cmd -U jacques --forest
```

When I run *jobtree* in my shell, the C code will run the bash ps command followed by 5 parameters: "-o", " user:32,pid,stime,tty,cmd", "-U", "jacques", "--forest".

I strongly suggest that you get your commands working/tested in bash before you port them into your shell. It will help with debugging issues when things don't work later on.

Don't forget to add in an extra command to exit your shell and return to the bash shell. As an example I would add an extra verb like "logout" which would "exit" my C code and return to the original bash process. Of course, *logout* shouldn't trigger a fork() as this would just create an extra process and not just exit.

Ensure that you wait() for every child created by fork() or you will be creating zombie processes. Which, I will have to clean up. They waste system resources so, be vigilant. Do a "ps -ef | grep -e $USER" before you log out and make sure you aren't leaving any zombies behind.

For this exercise, you are required to create your own verb equivalents. You are required to use normal bash commands with parameters. Once you have selected 3 or 4 commands (not counting *logout* or *help*), then use exec() function call to execute the code.

Here are the ones I chose:

My shell's verb	Bash equivalent
help	None but prints out a help menu
diskuse	`du -sch`
drives	`df -h -t ext4`
chklog	`grep -e $USER /home/COIS/3380/secure`
jobtree	**`ps -o "user:32,pid,stime,tty,cmd" -U $USER --forest`**
logout	Exits the shell

The help command will print out something like this (change it to match your verbs):

Valid verbs are:

```
help:                          displays this text
lastmod [number of days ago]   displays files modified on that day
diskuse                        shows the amount of disk being used in the current
                                   directory.
drives                         Lists all of the ext4 mounted partitions as well as
                                   their size and free space.
chklog  [username]             scans the access log for a specific username.
                                   If not present, it dumps the whole log file.
jobtree [username]             shows the processes running related to a specific
                                    username.
logout                         Exits this shell and returns to your original shell.


NOTE: elements in square brackets are optional. Should you
omit them, the current username will be used.

--- EOT ---
```

In my implementation, if the username is not specified on the appropriate command lines, I used the getenv() command to retrieve the name of the currently logged in user (the environment variable USER). If the "number of days" was not specified, I assumed a 1.

You are required to hand in to the Lab4 drop box on BlackBoard, a well document and modular program **plus sample output in a log file** showing the functionality of your program. Remember to use your script from lab1 to zip up your work for submission

HINT: if you keep all of your verbs to the same length of text, it will make parsing the command line a lot easier. If you have a single parameter, it is simply the rest of the line.

Logging your work:

The log file that you are required to include should be created AFTER you have done all of your coding and testing. That will keep your log file as clean as possible.

The log should include at the very least:

1) Running your code showing examples of each command in the above list
2) Running any additional shell commands you have created

You should then place your log file in your assignment3 or lab3 directory and then, and only then, create the zip file to be submitted to Blackboard.

Note: if you use the Unix script command to create your logs, you MUST clean them up before submission.

```
/home/COIS/3380/script-cleanup.pl  <  script_output.log    > clean_logfile.log
```