**Assignment 3 - Simple Shell**

Welcome to your third homework assignment.

Let me remind you of the expectations. Code should be neat, well documented. Variables should have meaningful names and be in a consistent format (I do not care if you use camelCase or under\_scores in variables but be consistent. In addition, each file should have a standard header as defined below. (Keep comments to around 80 characters per line, and not line grater than 100 characters).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Class: CSC-415-0# Spring 2023

\* Name:

\* Student ID:

\* GitHub ID:

\* Project: Assignment 3 – Simple Shell

\*

\* File: <name of this file>

\*

\* Description:

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

You can (and should) work in groups to research how to do the assignment

In our slides we talked about a simple shell. For this assignment you will implement your own shell that runs on top of the regular command-line interpreter for Linux.

Your shell should read lines of user input, then parse and execute the commands by forking/creating new processes. For each command, your shell should call fork() followed by execvp(). Following each command, your shell should wait for its child process to complete, and then print the child PID and the **return result from the child process**. The user should be able to specify the command to execute by giving a path to the executable file (e.g. /bin/ls) or by using path expansion to locate the executable file (i.e. searching each directory in the PATH environment variable). (Note that the execvp() function perform this processing automatically; you do not need to program it yourself.) This assignment does not use pipes.

If your shell encounters an error while reading a line of input it should report the error and exit. If your shell encounters EOF while reading a line of input, it should exit gracefully without reporting an error. Ensure that you do not overflow a 102 byte buffer when fetching the line of input (functions that do not accept the size of your buffer are not able to prevent overflows whereas functions that do accept a size generally do; be sure to check the manpage of any function you use carefully). You do not need to report an error if the user's input line is larger than the 102 byte buffer; just use the truncated input as the command.

You can test the EOF by running make run < commands.txt or from the prompt enter CTRL-D

Before your shell forks a new process to call execvp(), it should parse the input string and separate it into a collection of substrings representing the executable file and any command-line arguments. If the user entered an empty line, report an error and fetch a new line of input. Your code must handle at least four command-line arguments (in addition to the name of the executable file itself).

You should store pointers to the substrings in an array (similar to the “argv” array passed to main()) and pass this array of arguments to execvp(). Note that the number of command-line arguments is variable; this is indicated in the array by including a NULL pointer in the array after the last substring. (This means that if the user specifies N substrings, your array must hold N + 1 pointers where the last pointer is NULL.) If the user enters the **exit** command, your shell should terminate (returning to the regular shell).

Note: your shell does not need to support cd (change directory).

Your program must also accept a command line argument which is the prefix prompt. If no value is specified use “> ” as the prompt.

Here is a sample execution:

student@student-VirtualBox:~/CSC415/assignment3$ ./assn3 prompt$

prompt$ ls -l

total 20

-rwxr-xr-x 1 student student 13216 Feb 23 13:44 assn3

-rw-r--r-- 1 student student 1583 Feb 23 13:44 assn3.c

Child 2124, exited with 0

prompt$ ls foo

ls: cannot access 'foo': No such file or directory

Child 2125, exited with 2

prompt$ exit

student@student-VirtualBox:~/CSC415/assignment3$

You should submit your source code file(s), and Makefile along with a writeup in PDF format (using the template) that includes a description of what you did issues you had and resolutions and the compilation and execution output (screen shots) from your program to GitHub and just the PDF also to Canvas. Your execution output should include commands with command-line arguments. Then use the exit command to exit your program and show the output of the same commands in the regular command-line interpreter for that machine to ensure they match.

All filenames should be <lastname>\_<firstname>HW<#>\_<optional>.<proper extension>

Rubric

| **Grade Item** | **Grade Points** |
| --- | --- |
| Standard Header | 1 |
| Command Processing | 35 |
| Custom Prompt | 2 |
| Output Status | 10 |
| Code Comments | 5 |
| Writeup | 7 (Description, Compilation, Sample Output) |