Assignment 1

Problem statement: Write your own command shell using OS system calls to execute built-in Linux commands.

Important note: *A template is being provided with this assignment, which should be used to keep your code modular and easy to read/understand. Add proper comments for every logical step and follow proper code indentation. As this assignment will be evaluated without your physical presence, it is your responsibility to make your code readable and avoid unnecessarily complex flows as lack of readability and clarity in understanding will ultimately cost marks. Moreover to bring more objectivity and efficiency in the evaluation process, we are exploring appropriate platform/testing frameworks where we will try to autograde most part of the assignment along with code similarity check. So, do **NOT** design a fancy interface rather you will be advised to follow strict output template. More details about that and submission instructions will be provided soon. For more code related instructions, please read the instructions at the start of the template file.*

Your command shell should support below listed features.

1. Basic stuff

The shell should run an infinite loop (which will only exit with the '**exit**' command) and interactively process user commands. The shell should print a prompt that indicate the current working directory followed by '`$`' character.

For reading the complete input line, use `getline()` function. For separating the multiple words (in case of multiple commands or command with multiple arguments) from the input line, use `strsep()` function. To understand how these functions work, carefully read their man pages (http://manpages.ubuntu.com/manpages/bionic/man3/getline.3.html, http://manpages.ubuntu.com/manpages/bionic/man3/strsep.3.html). Avoid the use of such library functions that either needs some installation or needs passing command line arguments while compiling the program, as this may cause problems while submitting the code to automated grading platform.

To execute the commands with a new process, use `fork`, `exec` and `wait` system calls. Go through the man pages to understand different variants of `exec` and `wait` system calls so that you should be able to use the desired call for different cases. For more details on these, you may refer to the lab exercise document provided earlier. Do **not** use `system()` function to run the user commands.

2. Changing directory

Your shell should support '`cd`' command. The command '`cd <directoryPath>`' should change its working directory to `directoryPath` and '`cd ..`' should change the working directory to the parent directory. You may use `chdir()` system call for implementing this. To understand how it functions works, go through its man page (http://manpages.ubuntu.com/manpages/bionic/man2/chdir.2.html).

3. Incorrect command

An incorrect command format (which your shell is unable to process) should print an error message '**Shell: Incorrect command**' (do **not** change this message). If your shell is able to execute the command, but the execution results in error messages generation, those error messages must be displayed to the terminal. An empty command should simply cause the shell to display the prompt again without any error messages.

4. Signal handling

Your shell should be able to handle signals generated from keyboard using '**Ctrl + C**' and '**Ctrl + Z**'. When these signals are generated, your shell should continue to work and the commands being executed by the shell should respond to these signals. Your shell should stop only with the '**exit**' command.

5. Executing multiple commands

Your shell should support multiple command execution either sequentially or in parallel. The commands separated by '**&&**' should be executed in parallel and the commands separated by '**##**' should be executed sequentially. Also your shell must wait for all the commands to be terminated (for parallel and sequential executions, both) before accepting further inputs. Do not worry about simultaneous use of '**&&**' and '**##**', we will not test that.

6. Output redirection

Your shell should be able to redirect STDOUT for the commands using '>' symbol. For example, '**ls > info.out**' should write the output of '**ls**' command to '**info.out**' file instead of writing it on screen. To keep it simple, do not worry about simultaneous use of multiple commands and output redirection, as that wont be tested too.