CSL309: Operating Systems Lab 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. 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 will be using GradeScrope platform for auto-grading most part of the assignment and for code similarity check. So, do **NOT** design a fancy interface rather you will be advised to follow strict output template. 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 any similar 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 for sequential execution as well as for parallel execution. 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. Again, do not worry about simultaneous use of multiple commands and output redirection, as that won't be tested too.

7. Support command pipelines (*Optional: you can get extra credit of 20% i.e upto 3 marks more for doing this properly)*

You may want to implement pipe (to get extra credits) so that you can run command pipelines such as:

% cat myshell.c | grep open | wc

The '|' token should indicate that the immediate token after the '|' is another command. Your program should redirect the standard output of the command on the left to the standard input

of the command on the right. If there's no following token after '|', your program should print out an appropriate error message. There can be multiple pipe operators in a single command.

Submission instructions for assignment 1 on shell

1. You will be soon added as a student to GradeScope OS course. you would receive an email from 'team@gradescope.com' regarding the same. First, you need to open that email and follow the instructions to reset your password for GradeScope (if not already done). On successful password reset, you should be able to log in to your GradeScope account where on successful login, you will see CSL309 (Operating Systems) course. You will be able to see the assignments in OS course after their submissions open.
2. In OS course, you have to submit your code for 'Assignment 1 - Shell'. **Make sure your code filename is 'myshell.c'**.
3. The submissions will open on 19th August and the submission deadline is 25th August 6:00PM. Make sure your submission upload is completed before this deadline. If you miss this deadline, late submissions will be allowed till 26th August 6:00PM with a penalty of 3 marks deduction. No submissions will be accepted post the late submission deadline.
4. Post submission, your code will be automatically evaluated by Gradescope and marks (out of 9) will be allocated based on passed test cases. You can immediately see your grading result (of automated part) and re-submit the code (if you wish to) as many times as you want before deadline, so that you can change the code in order to get most/all the test cases passed.
5. After the due date, we will evaluate the other test cases manually in the lab and take viva and allocate the remaining marks (out of 6). The mark allocation scheme is given in a separate section (see below: grading scheme).
6. In case if your code is not passing any of the test cases on Gradescope after due adherence to the instructions and formatting but working perfectly fine on your own system (I don't expect this to happen normally), we will give you the opportunity for manual grading of those cases (unless there are many and that will have some penalty)
7. **For all the submissions, code similarity will be checked and appropriate penalty action will be taken on the similar submissions. Note that the provided template code will be excluded from the similarity check. The final autograder marks will be calculated as -** (autograder marks obtained - Late penalty)*Plagiarism penalty. Tentatively the penalty multiplier will be decided as per below table:

Similarity percent	**Penalty multiplier**
< 75	1
75 to 84	0.67
85 to 94	0.53
95 to 99	0.40
100	0.33

Important notes for automated test cases

1. Do **NOT** use sleep() (or any other such method which stalls the program for some time) in your code. GradeScope autograder fails if its execution is stalled.
2. Except for below mentioned text/messages (and except the messages printed for wrong command format for 'cd' command or multiple commands), do **NOT** display any other messages to standard input.
    a. For incorrect command - **printf("Shell: Incorrect command\n");**
    b. Just before exiting shell with 'exit' command - **printf("Exiting shell...\n");**
    c. Prompt for shell - **printf("%s$",currentWorkingDirectory);**
3. Do **NOT** add/remove newline characters (\n) in/from above stated messages.

Grading scheme

The total weightage of the assignment is 15 marks, out of which automated grading will be used for 9 marks and manual grading will be used for 6 marks. Please refer below for the details of automated grading and manual grading.

Automated grading –

Below are the details of commands that will be tested in each of the test case.

Important Note 1: Working of all these test cases depends proper working of 'exit' command, so if your first test case failed, then probably rest of the test cases will also fail.

Important Note 2: Test case number 5 and 6 are both for incorrect command, but only one of them will pass at a time (as intended) and as you can guess from their marks, test case 6 now combines incorrect command test case and a code quality test case.

1. 'exit'– 1 mark.
2. Other basic commands (such as ls, pwd, etc.) – 1 marks.
3. 'cd ..' and 'cd *foldername*' – 1 mark.
4. Running of multiple commands back to back with/without arguments (including cd commands) – 1 mark.
5. Incorrect command – 1 mark.
6. Incorrect command with proper process termination – 2 marks.
7. 2 sequential commands (separate by '##') – 1 mark.
8. 4 sequential commands (separate by '##') including cd commands – 1 mark.
9. Redirection of standard output (implemented with '>') – 1 mark.

Manual grading –

The grading scheme for manual grading will be as explained below.

1. Code explanation (other than code for multiple parallel commands and signal handling) - 3 marks.

2. Correct demonstration and code explanation of multiple parallel commands (separate by '&&') – 1 mark.
3. Correct demonstration and code explanation of signal handling – 1 mark.
4. Code quality (optimal use of fork, wait, exec) (Note: if your shell works exactly similar to bash shell in Ubuntu for the defined functionality, you can assume that most probably you have got this one right) – 1 mark.
5. Correct working and explanation of command pipelines – 3 bonus marks