Your task is to create an application in C *myshell* to parse command line input from the user and execute the command as child process. The objective is that you become familiar with fork(), exec(), wait(), dup2(), pipe() in Linux.

The design for *myshell* program will essentially consist of parsing through the input command strings, and executing the commands using system calls. Consider the following elements:

1- Shell Commands with No Arguments (ls, ps etc)

Each command irrespective of whether they have arguments or no arguments will be executed in an individual *child* processes spawned for its purpose from the main process. The main (parent myshell process) will fork and wait till all its child processes are finishes to prompt the user again for input. A simple pseudo code to describe this functionality is given below:

```
If fork doesn't equal 0
      In parent; wait for all child processes
Else
      Execute command
```

The system call used to execute the command will be execvp(). The first element of the input array will be the program name. execvp() will automatically search for this program in the path environment, and execute.

2- Shell Commands with Arguments (ls -l, ps aux etc)

This will be similar to the previous category, except that the command and its arguments will be provided as a string array arguments to execvp().

3- Redirecting output

There are three I/O streams which are stdin, stdout and stderr. The corresponding file descriptors are 0, 1 and 2 receptively. By default, these streams are connected to a terminal. But we can redirect the output to a file using the > operator. Shell can implement this feature using dup2() system call.

```
>> echo hello
hello
```

```
>> echo hello > new-file
>> cat new-file
hello
>> ls frhtrhrjy
ls: cannot access frhtrhrjy: No such file or directory
>>
>>ls frhtrhrjy 2> errout-file
>>cat errout-file
ls: cannot access frhtrhrjy: No such file or directory
```

4- Redirecting input

Like stdout and stderr, stdin (Standard input stream) is bounded with the terminal where the program is running. You can change this default behaviour using symbol < to bound the stdin to a file. Shell can implement this feature using dup2() system call.

```
>> echo hello > new-file
>> cat < new-file
hello
```

5- Pipes

It connects the standard output of one command to the standard input of another. You do this by separating the two commands with the pipe symbol |. Your shell has to wait for the last process in the pipe-line to terminate to show the prompt for the next command. This can be achieved using pipe() system call.

Example:

```
>> echo hello there | grep hello | grep there
hello there
```

6 - Make sure to use **exit** to exit the program

7 - Extensive commenting explaining every single line in the code

Noteworthy Points

- Although it is understood that you may exchange ideas on how to make things work and seek advice from your fellow students, sharing of code is not allowed.

- If you use code that is not your own, you will have to provide appropriate citation (i.e., explicitly state where you found the code). Otherwise, plagiarism questions may ensue. Regardless, you have to fully understand what and how such pieces of code do.

- Extensive and proper commenting must be done for all the program.

- Submissions:

    - C files + Make file
    - Report