CS4375: Theory of Operating Systems **Instructor: Dr. Deepak Tosh**

Fall 2021 **Due Date:** 11/17/2021, Midnight

Assignment # 2B

Goal: In this assignment, you will implement a *command line interpreter (CLI)* or, as it is more commonly known, a *shell*. The shell should operate in this basic way: when you type in a command (in response to its prompt), the shell creates a child process that executes the command you entered and then prompts for more user input when it has finished.

The shell you implement will be similar to, but simpler than, the one you run every day in Unix. If you don't know what shell you are running, it's probably **bash**. One thing you should do on your own time is learn more about your shell, by reading the man pages or other online materials.

Note: This is the **2nd phase** of the multi-part assignment, where you will be incrementally developing advanced functionalities to make it work similar to the shell program. It is better to execute this program in a UNIX environment as most of the shell commands will work better there. We encourage to use an Ubuntu 20.04 LTS VM to develop and test this assignment, if you are presently working on a Windows.

Program Spec

Your shell "**minershell**" is going to be an interactive loop, i.e. repeatedly prints the prompt "**minersh**$ ". Your program parses the input, executes the command specified on that line of input, and waits for the command to finish. This is repeated until the user types **exit**.

```
prompt>./minershell
minersh$
```

At this point, **minersh** is running, and ready to accept commands. Type away!

Prerequisites:

1. Familiarize yourself with the I/O related system calls, for e.g., STDIN, STDOUT, STDERR.
2. Familiarize in file manipulation in C.
3. Read about duplicating file descriptors using **dup2**() from https://man7.org/linux/man-pages/man2/dup.2.html

Task Details:

Now that you have already implemented a basic working shell that can take simple in-built commands, it's time to add a new functionality called "redirection". Here is a short tutorial (https://www.baeldung.com/linux/pipes-redirection) which you can read and learn about redirection and pipes. We will implement pipe in the next week's task, but it is good to get a head start from now.

Enabling File Redirection in your Shell

Many times, a shell user prefers to send the output of a program to a file rather than to the screen. Usually, a shell provides this nice feature with the "**>**" (greater than symbol) character. Formally this is named as redirection of standard output. To make your shell users happy, your shell should also include this feature, but with a slight twist (explained below).

For example, if a user types **"ls -la /tmp > output"**, nothing should be printed on the screen. Instead, the standard output of the **ls** program should be rerouted to the file output. In addition, the standard error output of the program should be rerouted to the file output (the twist is that this is a little different than standard redirection).

If the output file exists before you run your program, you should simple **overwrite** it (after truncating it).

Note: don't worry about redirection for built-in commands (e.g., we will not test what happens when you type path /bin > file). You should get the basic redirection working. Please do not get worried by the corner cases.

Program Errors

The one and only error message - You should print this one and only error message whenever you encounter an error of any type. The error message should be printed to STDERR as shown below.

```
char error_message[30] = "An error has occurred\n";
write(STDERR_FILENO, error_message, strlen(error_message));
```

Hint: you would be using **dup2** system call for changing the STDOUT value to the file where output will be redirected.

Grading:

The assignment will be graded on following items:

1. Completeness and correctness on your program.
2. Inclusion of evidence with enough test scenarios (in form of screenshots) in a report
3. Programs' readability and correctness
4. Hurdles faced while implementing

5. References (including links where you found some sample code.