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

Fall 2021 **Due Date:** 10/01/2021, Midnight

Assignment # 2C

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 **3rd and last phase** of the multi-part assignment, where you will be developing one final functionality 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!

Background about PIPEs:

Pipe is a kind of redirection (transfer of STDOUT to some other destination), which is used in Linux/Unix Operating Systems to output of one command/program/process to another command/program/process for further processing. With pipes, you can combine two or more commands, where output of one command acts as input to another command. Typically, the data flows from left command to right command through the pipeline.

For example,

command_1 | command_2| command_3 | … |command_n

Another example: If I want to do a look all files/directories available in my home directory in a dictionary sort, I can use the following two commands in pipeline.

$ ls ~| sort

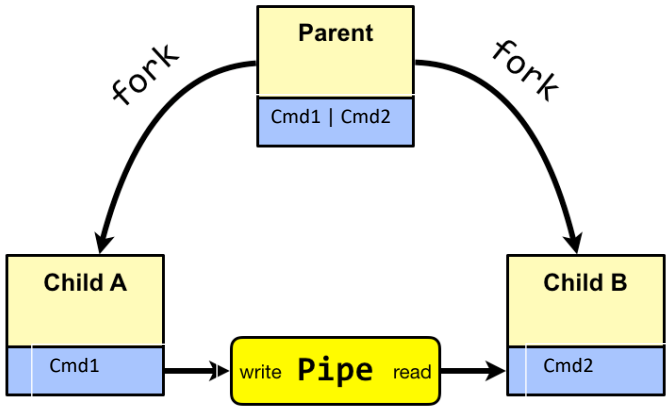
In the above example, "sort" command takes the output of "ls ~" as its input.
Read about Unix Pipelining from the wiki https://en.wikipedia.org/wiki/Pipeline_(Unix)

Task Details:

Now that you have already implemented a basic working shell with file redirection. It's time to add a new functionality called "pipes". Here is another article that will help you understand how pipes are implemented in UNIX: http://web.cse.ohio-state.edu/~mamrak.1/CIS762/pipes_lab_notes.html and https://people.cs.rutgers.edu/~pxk/416/notes/c-tutorials/pipe.html

Augmenting your "Minershell" with pipelining functionality:

Your task is to modify your previous C-based shell program to implement the pipeline functionality, where you can execute commands with **at least one pipe** (e.g**. ls | wc**). The way your implementation would work is shown in the figure below.



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));
```

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.