Assignment 1

August 16, 2022

Building A Shell

We will first build a simple shell, much like the bash shell of Linux. A shell takes in user input, forks one or more child processes using the fork system call, calls exec from these children to execute user commands, and reaps the dead children using the wait system call. Learn about the fork system call and all variants of the wait and exec system calls before you begin this assignment.

1. Build a simple linux shell using C. On executing your code, the shell should display a prompt and ask the user for input. The format of the prompt should be as follows:

```
[{current_directory}~$ ]
```

Notice there is a space after the $
For example if the *current_directory* = '/home/saylor/Desktop/dev', then the prompt should be

```
/home/saylor/Desktop/dev~$
```

Any entered command will look like:

```
/home/saylor/Desktop/dev~$ echo hi
```

When ENTER is pressed the shell should execute the command and display output (if any). Then it should move on to the next line displaying the prompt and waiting for user input. The shell should terminate on pressing $Ctrl + C$. An incorrect number of arguments or incorrect command format should print an error in the shell. The shell must not crash and should simply move on to the next command. The shell should execute simple commands (commands which are readily available as executables and can be invoked using exec system calls) like ls, cat, echo, sleep.

2. Other than supporting simple command execution, the shell should support running commands in background, support piping between commands, and support environment variables.

3. The shell should also support two special commands, *cmd_history* and *ps_history*.
cmd_history: Display the **Last 5** executed commands(irrespective of failure or successful run) in FIFO order(latest command on top). Example output :

```
echo hii
echo $PID
&python3 file.py
cat file.txt | tail -5
```

ps_history: Display **all** the child processes started by me. Along with their current status(RUNNING or STOPPED). Example output :

```
1134 RUNNING
1124 STOPPED
1145 STOPPED
```

Note that there may be a race condition that while you are printing the output, as process may have stopped. You need not worry about these cases, the only thing that you need to guarantee is that **eventually** (after some calls) the right state of the process will be printed.

The expected format for the user input commands should be as follows:

```
Normal command execution     :    cmd arg1 arg2 ... argn
Start command in background  :    &cmd arg1 arg2 ... argn
Piped command                :    cmd1 arg1 ... argn | cmd2 arg1 ... argn
Set environment variable     :    identifier=value
```

where,

```
cmd = cmd_history  or ps_history or alphanumeric string
arg_i = alphanumeric string or $identifier
identifier, value = alphanumeric string
alphanumeric string = one or more occurrences of elements from
{A,B,C,D....,Z,a,b,c,d....,z,1,2,3,4...9,-,_}
```

Submission Details

Submit a zip file with folder name <Entry_No>_A1 (eg: 2019MT60648_A1). The folder should contain your C code file named shell.c