Assignment 2

CS 301 - Operating Systems

Due date: 15th September, 2021

1 Introduction

In this assignment, you'll be building a shell, similar to the Bash shell you use on your Linux machines. When you open a terminal window on your computer, you are running a shell program. The purpose of a shell is to provide an interface for users to access an operating system's services. Shells can be interactive or non-interactive. For instance, you are using bash non-interactively when you run a bash script. It is important to note that bash is non-interactive by default; run `bash -i` for a interactive shell. By building your own shell, you'll become more familiar with these interfaces and you'll probably learn more about other shells as well. This assignment is due on 15th September, 2021, end of day.

To get started, access the `assignment2` link assigned to you on GitHub classroom. After cloning the repo on your virtual machine, run `make` inside the folder followed by `./shell` to run the shell. To terminate the shell after it starts, either type `exit` or press `CTRL-D`.

2 Support for `cd` and `cwd`

Every shell needs to support a number of built-in commands, which are functions in the shell itself, not external programs. For example, the `exit` command needs to be implemented as a built-in command, because it exits the shell itself. So far, the only two built-ins supported are `?`, which brings up the help menu, and `exit`, which exits the shell.

Using the `chdir` and `getcwd` functions (and any other functionality you might require), add two new commands — `cwd` that prints the current working directory to standard output, and `cd` that takes one argument, a directory path, and changes the current working directory to that directory.

3 Program Execution

If you try to type something into your shell that isn't a built-in command, you'll get a message that the shell doesn't know how to execute programs. Modify your shell so that it can execute programs when they are entered into the shell.

The first word of the command is the name of the program. The rest of the words are the command-line arguments to the program. When your shell needs to execute a program, it should fork a child process, which calls one of the `exec` functions to run the new program. The parent process should wait until the child process completes and then continue listening for more commands.

Assume that the first word of the command will be the full path to the program. You should use the functions defined in `tokenizer.c` for separating the input text into words. You do not need to support any parsing features that are not supported by `tokenizer.c`. Once you implement this step, you should be able to execute programs like this on the shell (i.e., after running `./shell`):

```
/usr/bin/wc shell.c
nn nnn nnnn shell.c
```

4 Our Pipes

When running programs, it is sometimes useful to provide the output of a program as the input of another program. The syntax "`[A] | [B]`" tells your shell to pipe the output of process `A` to the input of process `B`. Simply put, the output of `A` becomes the input of `B`. Modify your shell so that it supports pipes between programs. You can assume there will always be spaces around the special character `|`. Outputs may be piped more than once. For example, "`[A] | [B] | [C]`" can be passed as an argument to the program.