CMPSC 400
Operating Systems
Spring 2020

Lab 2 Assignment:
Programming Your Own Shell
Submit deliverables through your assignment GitHib repository bearing your name. Place source code in `src/` and written work in `writing/`.

Objectives

We will be following a tutorial to create a personalized system terminal shell that will allow you to run seemingly any program. To complete this terminal shell, you will hard-code the commands into c code.

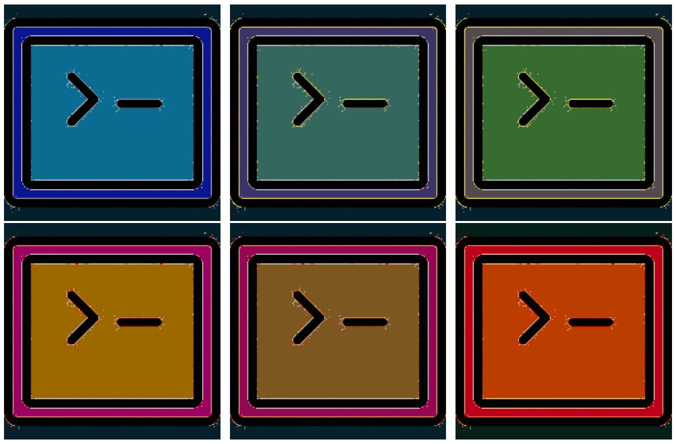

Figure 1: When a user interacts with a OS, commands must be relayed to the kernel. These commands may be sent by working with a graphical interface, or they may be made using key-stroaks. In this lab, we will create a program that will relay user-created key-stroaks to the kernel using system calls.

GitHub Starter Link

`https://classroom.github.com/a/Btsp4ZS7`

To use this link, please follow the steps below.

- Click on the link and accept the assignment.
- Once the importing task has completed, click on the created assignment link which will take you to your newly created GitHub repository for this lab.

- Clone this repository (bearing your name) and work on the lab locally.
- As you are working on your lab, you are to commit and push regularly. You can use the following commands to add a single file, you must be in the directory where the file is located (or add the path to the file in the command):
  - `git commit <nameOfFile> -m ''Your notes about commit here''`
  - `git push`

  Alternatively, you can use the following commands to add multiple files from your repository:
  - `git add -A`
  - `git commit -m ''Your notes about commit here''`
  - `git push`

Introduction

A *Shell* is discussed in Section 1.5.6 of Tanenbaum [1], and while not necessarilty a part of the OS, it is a program that connects the user with the kernel and the general OS. Unless the user is interacting with the kernel using a graphical user interface, there are few oher means for the user to signal tasks to be completed by the kernel. These software program, known as `bash prompt` or *terminals*, depending on their original OS systems, input typed commands from the user. Their commands are then parsed to extract information, such as filenames, options and parameters, which are then used for communication with the OS and the generation of system calls to the kernel.

Tutorial

In this week's lab, we are going to create our own simple UNIX shell using the C programming language by following a tutorial by Stephen Brennan, available from the below link. In this tutorial, the source code for his shell project is given, however you are still to spend some time to study this tutorial in order to understand the underlying mechanisms and functionalities of the provided shell code.

Link: `https://brennan.io/2015/01/16/write-a-shell-in-c/`

Your Shell

Give your shell a name and use the given code from the tutorial to build your own shell program. Please remember to cite the shell's original creator and to provide the referencing website to your own source code. Add your name to your source code file as its modifier to create your own shell. Call your new shell file, `newShell.c` and compile it with the following commands.

```
gcc -o newShell newShell.c
```

If all has gone well, then the `gcc` command has compiled the source code file (*newShell.c*) and created an executable called *newShell*. Please take a moment to try out this shell by executing it using the following command.

```
./newShell
```

Functionality

There are several built-in commands which the author has already created which you are to test for function.

- `help` - Shows these commands.
- `cd` - Changes current directory to another.
- `exit` - Quits the shell.

Once you have completed the tutorial and you have tested the above three commands, you are to add another built-in function called `square` which draws a simple square to the shell's output. Note that square accepts no parameters and will draw a 7 by 7 square to the screen using simple text characters. Your output from using the `square` command in your shell will have the following output.

```
> square
  The Square command.
  Drawing a 7 by 7 square
   x x x x x x x
   x x x x x x x
   x x x x x x x
   x x x x x x x
   x x x x x x x
   x x x x x x x
   x x x x x x x
  Square command is complete!
```

In order to build this new function, modify the `newShell.c` code by concentrating your attention on the following sections for your code updates. Remember to observe how the code has been written already and use this work as a template for your own updates to the code. As you update the code to add the additional function, please visit the following areas for your modifications:

1. Declare the `square` function to: *Function Declarations for built in shell commands*,
2. Add the `square` command to the array *char *builtin_str[]*,
    - Watch out for the commas.

3. Add the `square` command to the array *int (*builtin_func[]) (char **)*,
4. Build a new method called, *int lsh_square(char **args)*,
   - This method is very similar to the existing method, *int lsh_help(char **args)*.

Extra Challenge

Why not continue onward to adding other types of functionalities to your new shell. For instance, you might decide that there are specific types of prompt arguments that you frequently use when you code or compute. Why not add some of those commands here and make this learning tool into something more exciting?!

Testing Your Programs and Evaluating Your Submission

You will be using Docker in this lab. For specific commands and instructions for compiling and running your programs, as well as checking that all required files have been included, please see the README file in your lab repository.

Required Deliverables

This assignment invites you to submit an electronic version of the following deliverables:

- File `src/newShell.c`: The working code for your shell that can be compiled using `gcc`.
- File `writing/report.md`: Report: using screenshots, text and and concrete examples, show that your shell is able to compile and run. Please mention in this report how you compiled and ran this source code.

Honor Code

In adherence to the honor code, students should complete this assignment on an individual basis. While it is appropriate for students in this class to have high-level conversations about the assignment, it is necessary to distinguish carefully between the student who discusses the principles underlying a problem with others and the student who produces assignments that are identical to, or merely variations on, someone elses work. As such, deliverables that are nearly identical to the work of others will be taken as evidence of violating the Honor Code.

References

[1] A. S. Tanenbaum and H. Bos, *Modern operating systems, Forth Edition.* Pearson, 2015.