CSCI 3150 Introduction to Operating Systems: Assignment One

1. Basic Information

- Topic: implementation of a simple shell program in C
- Total Marks: 100
- Deadline: 18:00:00 p.m., Mon, Feb 13th
- Submission: Github Classroom
    - Click the link to accept assignment one in Github Classroom:

2. Task Instruction

In assignment one, you will be guided to implement a simple shell program. Please check below sections for instructions on the assignment.

2.1. Target

Suppose we have a directory structure shown below as an example:

- dir1
    - dir2

Implement a shell program that supports:

- one built-in command: `cd` command

  Your shell program should be able to change current working directory with `cd` command. For example:

  ```
  user@OSLAB1:/dir1/dir2$ cd ..
  user@OSLAB1:/dir1$
  ```

  We only require you to support cd command **with one argument**, which is a path. We do not require you to support other scenarios such as cd with flags.

- single external commands, i.e. `ls`, `ps`, etc.

  Your shell program should be able to execute single command with arguments. For example:

  ```
  user@OSLAB1:/dir1$ ls -l
  total 4
  drwxrwxr-x 2 user user 4096 Jan  12 12:00 dir2
  user@OSLAB1:/dir1$
  ```

- pipes between commands. The program should support up to **two pipes**.

  Your shell program should be able to execute commands with up to two pipes between them. For example:

```
user@OSLAB1:/dir1$ ls -l | grep dir | wc -l
1
user@OSLAB1:/dir1$
```

Note: your program should be able to handle the case in which **there are at least one space(or tab, or both) before and one space(or tab, or both) after the pipe meta character "|"**. If there is no space before or after "|", for example, for the following case: "ls|" is treated as **one argument** to be executed (rather than "ls" and "|"). Although in real bash shells, "ls|" will still be treated as command "ls" followed by pipe meta character, we do not require you to deal with such scenarios in assignment one. We are also aware of this when designing our grading test cases.

Check `README.txt` in `assign1.zip` for more test cases.

2.2. Implementation Hints

2.2.1. Built-in commands & external commands

Built-in commands are contained within the shell itself, while other commands are supported by compiled binaries. One key difference between these two kinds of commands is that the shell will execute built-in commands in its main process and will fork a child process to execute external commands.

For example, linux shell supports the `cd` command to change current working directory, which is a built-in command and should be done in the main process of shell since changing directory in a child process will not affect the main process. Linux shell also supports the `ls` command, which is not a built-in command. When you type `ls` in the shell and enter, shell will invoke a child process and execute the corresponding binary in the child process if the binary is found under directories specified in environment variables. For `ls` command, the corresponding binary named `ls` is under `/bin` directory, which is usually a directory included in `PATH` variable so the shell can find where the binary of `ls` is. Forking child process to execute external commands can facilitate parallel execution of several commands of a shell.

In this assignment one, your shell program is only required to support one built-in command, which is `cd`. You should take the behavior difference between built-in commands and external commands into consideration when finishing assignment one.

2.2.2. Functions to use

Please utilize the following functions to implement your shell program.

- `chdir()`
- `execvp()`
- `dup(), dup2()`
- `pipe()`

Note: other functions can also be utilized but we recommend using the above functions.

2.2.3. TODOs in the code

Please revise `simple-execute.c` to implement required functions of your shell program. In `simple-execute.c`, we provided a general framework to help you better understand the code structure. You can fill in the code in the positions of `TODO`. However, you may also ignore this framework and implement in your own way.

3. Submission and Grade Posting

You only need to submit your revised `simple-execute.c`, which will be compiled with the same `Makefile` provided in `Assign1.zip` and tested with several test cases. Please make sure all your codes are resided in `simple-execute.c`.

How to use Github Classroom.

Grade posting done in Github classroom PR.

4. Other Notes

- We will compile, run and grade your program with Ubuntu 18.04.6 and gcc 7.5.0. Please make sure your program sources are compatible with the corresponding version of Ubuntu and gcc. Otherwise, 0 marks will be given.
- Note that we can only grade what you submit in the repo created by Github Classroom after acceptance of assignment one. Late submission will be graded based on the submission time and our late submission policy. Please find related policies on the course website.
- Several new test cases will be utilized when grading. Test cases in `README.txt` are provided for validation.
- TA WU, Shaofeng is responsible for this assignment. Questions about the assignment via Piazza are welcomed and preferred but you may also contact shaofeng via email: wsf123@link.cuhk.edu.hk. Requests including but not limited to asking TA to set up environment, write code and debug for you will be rejected according to regulations.