mysh I: Looking around the File System

Operating Systems, Fall 2017
Ajou Embedded Intelligent Systems Laboratory

Due: October 9^{th} 2017 23:59 KST

1 Introduction

This semester, we will implement our own shell named - *mysh*. *mysh* is a shell with the basic functionality of a typical shell: for example several built-in commands, program execution, input/output redirection, etc.

In this first assignment as a first step to shell implementation, we ask you to design two functions: **1)** interaction with user using the command line and **2)** basic built-in commands of *pwd* and *cd*. Using this shell you will gain the capability to look around your file system by inputting the *cd* command and realize where you are in the file system with the *pwd* command on your own shell.

2 Objectives

This assignment is targeted towards the following objectives.

- Introduce the Linux development environment.
- Improve programming skills and exception handling.
- Get used to using Unix system calls.

3 *mysh* framework

In this assignment, you will be provided with a template code for implementing *mysh*. This code has the basic functionality to start implementing a shell. For example, this template includes code to get input from the user and classify different commands. You can find the details in the codes of your repository, especially the ./src/main.c file. For the purpose of the assignment, you can simply edit some functions. The details will be discussed in later sections.

Please note the following when implementing the *mysh* framework to proceed your assignment with no unexpected errors.

- DO NOT TOUCH ANY CODE EXCEPT FOR THE ./src/commands.c AND ./src/utils.c FILES. This is important in grading your code automatically. (But if you notice bugs in our mysh framework, please let the TAs know by sending an email or send a Pull Request to our repository. Finding bugs will be extra points!)
- If you want to get an executable, use command *make*. Then you will see the ./mysh executable file.
- If you want to do a unit test, use command *make test*. Googletest will show you what test succeeded, what test failed.

4 Function 1: mysh_parse_command()

If you receive some command from the user, you should build an argument list. Then, you can pass arguments to your executing program. To build this, you are going to go through two steps: 1) tokenize the command from the user and 2) allocate space to store command tokens. For this job, you should program the body of this function, *mysh_parse_command()* of ./src/utils.c. The prototype is as below.

```
/* ./src/utils.c */
void mysh_parse_command(const char* command,
                        int *argc, char*** argv) {
    // TODO: Write your own code.
}
```

- *command (in)*: the string user typed.
- *argc (out)*: the number of arguments.
- *argv (out)*: the argument list.

Below are example code snippets describing how the function `mysh_parse_command()` works.

```
/* testing */
int argc;
char** argv;

mysh_parse_command("cd test", &argc, &argv);
/*
argc == 2
argv == { "cd", "test" }
*/

mysh_parse_command("cat file.txt | grep hi", &argc, &argv);
/*
argc == 5
argv == { "cat", "file.txt", "|", "grep", "hi" }
*/
```

The grading of this part is done by unit testing. The unit testing framework will put various sample inputs to validate your code. You should write exception handling to prevent test failures.

5 Function 2: `pwd` and `cd` commands

You should implement two basic commands `pwd` and `cd`. `pwd` is a command printing out your current working directory. `cd` is a command changing your working directory to an absolute or relative path you entered. The user flow of these commands is same with below example.

Listing 1: "Userflow of `pwd` and `cd`"

```
pwd
/home/your-name/workspace
cd ..
pwd
/home/your-name
cd .
pwd
/home/your-name
cd /bin
```

```
pwd
/bin
```

Using our code base, you should complete four functions in ./src/commands.c, `do_cd()`, `do_pwd()`, `validate_cd_argv()`, and `validate_pwd_argv()`. As long as the code is bug free, you do not have to touch ./src/main.c.

The grading of this part is composed with both unit and functional testing. For unit testing, the unit testing framework puts various input to validate functions to check that your code works correctly and how robust your exception handling code is. In the functional test phase, our autograding system will put sample input strings to your executable program, and compare with the prepared answer sheet.

Hint: Use the system calls `getcwd(3)` and `chdir(2)`.

6 Submission process

For efficient grading, this class will be using an autograding system. Our system will grade your work using unit tests and functional tests automatically. For this, you should follow some processes when submitting your work.

When you start to program, you should do this:

- Go to this link: https://github.com/AEIS-Lab/mysh-0
- *fork* this repository to your own account.
- *git clone* from your repository and start developing

When you want to submit, follow this:

- *git commit* and *git push* all you did to your repository.
- Tell us your github repository link using the Google survey link. (Link address TBA)
- Submit your report using e-class.

7 Grading

The grading policy for this homework is as follows.

- Pass ratio on unit test (40%)
- Pass ratio on functional test (40%)
- Report (20%)

To get the point on unit and functional test, you should pass each test cases, besides you should implement correctly. For example, your implementation is similar with below case, you can't get any point.

```
// test
add(1, 1) == 2

// fake implementation
int add(int a, int b) { return 2; }
```

JK: What does this mean?

Regarding the report, you must explain what you did in this assignment, what you learned, and your comments on this assignment (e.g. This assignment was too easy/hard, too strict/flexible, and any feedback you have on the homework). Lengths of report doesn't matter, but a 2-3 page report is recommended. DO NOT ADD CODE IN THE REPORT UNLESS NECESSARY.