CSCE3613 Operating Systems
Programming Assignment Two
Part two of implementing our simple Shell

By Wing Ning Li

1 Problem Description

It is assumed that you have read the problem description of assignment one and this is a continuation of that.

In programming assignment one, we wrote a program in C that runs in Turing (Linux) that mimics the behaviors of the Shell (bash shell) program in parsing a user input string. In that assignment, as a warm up of using C and Turing (Linux), a user input string is given as a command line argument in double quotes. We will enhance the functionality of that program in this assignment as follows.

1. Making parsing command string code into a function of which the prototype is shown as:

   ```
   int parse_command(char* line,
                     char** cmd1,
                     char** cmd2,
                     char* infile,
                     char* outfile);
   ```

   This function will parse the user input given in the parameter `line`. The function returns

   (a) 0 if `line` is "quit";

   (b) 1 if `line` is a simple command or program (without redirection and without pipe);

   (c) 2 if `line` is a simple command with only input redirection (without output redirection and without pipe);

   (d) 3 if `line` is a command or program with output redirection `>>` (append to output file) but without pipe;

   (e) 4 if line is a command or program with output redirection `>` (overwritten output file) but without pipe;

   (f) 5 if `line` is a pipe with two commands or programs without redirection;

   (g) 6 if `line` is a pipe with two commands or programs with only input redirection (without output redirection);

   (h) 7 if `line` is a pipe with two commands or programs with redirection(`>>`);

   (i) 8 if line is a pipe with two commands or programs with redirection(`>`);

(j) 9 if line is something else yet to be handled by our program but is handled by the shell perhaps.

The variables line, infile, and outfile are C-style strings and cmd1 and cmd2 are arrays of C-style strings (similar to argv in main), the last element in cmd1 and cmd2 should be a NULL pointer.

Notice that even though the returns of 3,4, and 7, 8 do not mention about input file redirection in the return code, it is possible the user input/command may contain input redirection. We may find out that indirectly if we pass an empty string `infile` to the function and upon return the value of `infile` is no longer empty, then we know we have input file redirection. Hence, the returns of 2 and 6 merely make the condition of input redirection explicit.

For example, if line is "`ls -a >> myfile`", then the return value of the function is 3; cmd1[0] = "ls", cmd1[1] ="-a", and cmd1[2] = NULL (pointer to char); outfile = "myfile", and cmd2 and infile are not affected.

Note that the function assumes that memory space (pointed to by those pointers in the formal parameters) has been allocated by the caller for the function, which may or may not be the most flexible design (Can you see why or why not? What would be a better design?).

2. Making the program interactive. That is if the program receives a command line argument, it will parse and execute the command line argument (string in the double quotes). Otherwise, it will output "`myshell-%`" and read the entire line, which the user types, from the console or terminal as a command. It then parse and execute the command, wait until the command finishes, and repeat the prompt for the next command. If the command is "quit", the program will finish.

3. Adding the code to execute a simple command (return 1 from `parse_command` function). There are several ways to implement it. We should not use library (man 3) functions such as `system` or `exec`, since these functions are implemented using system calls (our purpose is to gain some experience with system calls). Instead we must use kernel system calls (man 2). For us, we will use system call `fork`, `execvp`, and `wait`. We have discussed or will discuss these system calls in class. You are encouraged to use man or man 2 to look them up in turing or use the internet to gather more information about them. The sample program on page 118 should help to illustrate the logic to execute a simple command even though the command is hard coded there and a different system call `execlp` is used.

2 Purpose

Review C/C++ programming. Review using Linux/Unix. Review program development under Linux/Unix. Learn or review input parsing. Learn kernel system calls `fork, execvp, exit` and how to use them to implement a simple Shell. Learn the concept of process and programming ideas related to the process. Realize that we can start the execution of a program from within our program.

3 Design

See the design of `parse_command` function in the introduction. The behavior of the simple Shell is designed for us by the bash Shell that we want the simple Shell to mimic. As for the user input to our program, there are two ways: 1) statically where the input is provided as part of the command line that runs the simple Shell; 2) interactively from the terminal or console.

4 Implementation

Using C programming language (gcc compiler), array of characters (C style string) and pointers, C library function, and kernel system calls.

5 Test and evaluation

Try different combinations and scenarios to make sure our program works. Put all the tests you have used in a file, one per line.

6 Report and documentation

A short report about things observed and things learned and understood. The report should also describe the test cases used and the reasons for each test case selected. Properly document and indent the source code. The source code must include the author name and as well as a synopsis of the file.

7 Lab submission

To have a better organization, for each of our programming assignments we should have a directory (folder), under which we should have source code (.c .h files), report, and makefile if you use make, etc. This is the directory where we compile and test run our program. In **this directory**, type the following:

```
submit csce3613 wingninggrader assign2
```