CS3480 Principles of Secure Application Development
Programming Lab 1

Submission instructions are provided at the end of this document.

This is an individual assignment. Any plagiarized submissions will receive ZERO marks.

Important: DO NOT host your code publicly on GitHub or other public places during development phase. You are free (and encouraged) to use Git or other version controlling systems privately.

Understanding, Creating and Using Linux Processes

Goal: Goal of this programming assignment is for you to get a first-hand experience and understanding on using fork and exec family of system calls.

Part 1:

Task: Write a program **`lab1p1.c`** (in C language) to mimic the following functionality (when executed in Linux shell (eg. bash).

```
shell~:$ ./lab1p1 /bin/echo Linux is cool + /bin/echo But I am sleepy +
/bin/echo Going to sleep now + /bin/sleep 5 + /bin/echo Now I am awake
```

The effect of the above command should be running four processes in the shell as follows.

```
/bin/echo Linux is cool
/bin/echo But I am sleepy
/bin/echo Going to sleep now
/bin/sleep 5
/bin/echo Now I am awake
```

Notes:

1. If these binaries are located in different paths in your system, please mention the correct path. You code in **`lab1p1.c`** **must** use **`execve()`** to execute the binaries.
2. As you can see from the above example, input to lab1pl includes a set of arguments separated by '+'. Assume that one such argument could have a maximum length of 8 words and minimum length of 0 words.
3. To handle empty arguments, use **/bin/true** command. Look it up in man pages!
4. You should test your system thoroughly using different input arguments. For testing, you may write additional shell scripts. Make sure you give relevant executable permissions to shell scripts when running.
5. For debugging purposes, you may write to stderr:
   `fprintf(stderr, "error = %d\n", errcode);`

Part2:

Task: To develop a shell-like tool **`lab1p2.c`** to execute repetitive commands.

Example input 1:

```
shell~:$  ./lab1p2 gcc -c %

      a.c
      b.c
      c.c
```

In the above example, after the first line, there are three lines a.c, b.c, and c.c input through the stdin. The execution of the above input should mimic the behavior of separately running the following commands.

```
shell~:$ gcc -c a.c
shell~:$ gcc -c b.c
shell~:$ gcc -c c.c
```

% is a placeholder for the words that will be entered in the subsequent lines.

Example input 2:

```
shell~:$  ./lab1p2 gcc % % % % % %

      -c a.c
      -c b.c
      -c c.c
      -o a.o b.o c.o
```

In the second example, the command is expecting multiple words on the same consequent input line. **`lab1p2`** should facilitate any number of input words **less than or equal** to the indicated number of %. The maximum number of % is 8.

To indicate the end of input, you will need to type CTRL-D.

Note: For **`lab1p2`**, you must use either **`execvp()`** or **`execve()`** to execute commands.

Submission Instructions:

You will need to submit a .zip archive named with the registration number eg: 200111X.zip to the Moodle submission link. The zip archive should include the following.

1. The complete source code with instructions to build the code and use it, written in a readme file.
2. A small report (max two A4 pages) about how you approached this task, the challenges you faced, and how you resolved them.

Good luck and have fun!

If you have any questions, ask and discuss them in the **Lab1 Discussion Forum** on Moodle. If you notice any mistakes in the writeup, drop me an email.