CMSC 216 Exercise #5 Summer 2022

shell_jr ("Shellito") **Due: Fri Jul 22, 2022, 11:55 pm**

1 Objectives

To practice fork and exec* system calls by implementing a very simple shell.

2 Overview

The first thing you need to do is to copy the directory shell_jr we have left in the grace cluster under the exercises directory. Remember that you need that folder as it contains the .submit file that allows you to submit.

IMPORTANT: You can discuss this exercise with classmates, but you may not exchange code and may not write code together.

3 Grading Criteria

Your assignment grade will be determined as follows:

Results of public tests	28%
Results of release tests	72%

4 Academic integrity statement

Please **carefully read** the academic honesty section of the course syllabus. We take academic integrity matters seriously. Please do not post assignment solutions online (e.g., Chegg, github) where others can see your work. Posting code online can lead to an academic case where you will be reported to the Office of Student Conduct.

5 Specifications

For this exercise you will implement a simplified shell. A shell is a C program that executes commands. We have been using tcsh, but there are other shells like ksh, sh, bash, etc. For this exercise shell_jr will be the name of the shell you will implement.

There two types of commands a shell can process: linux commands and shell commands. Linux commands are programs that reside in directories like /usr/bin (e.g., /usr/bin/ls). To execute a linux command the shell forks itself and loads the program using an exec* system call (e.g., execvp). Shell commands (e.g., cd, exit) do not require a fork/exec call. They can be implemented by using system calls and other resources.

6 Shell Jr Functionality

Your shell will have a loop that reads command lines and process them. The prompt for your shell will be "shell_jr: ". The commands your shell must handle are:

1. **exit** - When the user enters the **exit** command the shell will stop executing by calling exit(). Before executing exit, the shell will print the message "See you".
2. **hastalavista** - Has the same functionality as **exit**.
3. **cd** - This command changes the working directory. You can assume the user will always provide a directory as an argument. Use chdir() to change the working directory.
4. A command with a maximum of one argument (e.g., **wc location.txt**). That means your shell should be able to handle commands like **pwd**, **date** or **wc location.txt**.

7 Requirements

1. You must NOT use an exec* function to implement the functionality associated with the commands exit, hastalavista, and cd. For other commands, you must create a child (via fork()) and use execvp() to execute the command.
2. If the user provides an invalid command, the message "Failed to execute " followed by the command name should be printed. In this case the child will exit returning the error code EX_OSERR. Use printf to display the message and flush the output buffer (e.g., fflush(stdout)). Note that the shell is not terminated by executing an invalid command.
3. You don't need to handle the case where the user just types enter (you can assume the user will always provide a command).
4. Make sure you use printf to print the shell prompt and that you flush the buffer.
5. It is your responsibility to verify that your program generates the expected results in the submit server.
6. You must use execvp (and no other exec* system call).
7. Your code must be written in the file shell_jr.c.
8. You may not use dup2, read, write, nor pipes.
9. You may not use system() in order to execute commands.
10. You can assume a line of input will have a maximum of 1024 characters.
11. Provide a makefile that builds an executable called shell_jr. Name the target that builds the executable shell_jr . Also add a target called clean that removes the shell_jr executable.
12. Your program should be written using good programming style as defined at

    `http://www.cs.umd.edu/~nelson/classes/resources/cstyleguide/`
13. Common error: If you get the submit server message "Execution error, exit code 126" execute "make clean" before submitting your code.
14. Common error: To forget to return the correct value (e.g., 0) in your code.
15. Your C program representing your shell does not take command line arguments. That is, the main function is defined as:

    ```
    int main() { }
    ```
16. When you see that execvp relies on argv that means it uses an array of strings (argv is not the parameter associated with command line arguments). Just initialize argv with an array of strings and pass it to execvp.
17. Your shell should exit when end of file is seen. This explains why public tests do not have exit nor hastalvista as the last command.
18. This exercise relies on Standard I/O, NOT Unix I/O.
19. shell_jr takes a maximum of two arguments (e.g., wc location.txt). You can ignore any other values provided after the second argument. For example, if someone enters **wc location.txt bla** bla will be ignored and the command will be processed successfully.

20. For exit and hastalavista you can ignore any values provided after the command (just exit the shell).

21. For cd you can ignore any values provided after the directory name. For example, cd /tmp bla will change the working directory to /tmp.

22. If an invalid directory is provided to the cd command, your shell should print an error message similar to:

    ```
    "Cannot change to directory INVALID_DIRECTORY_NAME"
    ```

    where INVALID_DIRECTORY_NAME will be replaced with the invalid directory name.

23. Do not use signals.

8 How to Start

You should start by creating a loop that reads lines and displays them. Then you should begin to process each command (starting with the exit and cd commands). You are free to process each line any way you want, however, reading a whole line using fgets and then processing the line using sscanf could make things easier. Keep in mind that if sscanf cannot read a value into a string variable, it will not change the variable. This could help you identify when a command has an argument or not.

9 Submitting your assignment (You need to do something extra)

Before submitting your work, execute ”make clean” otherwise you may get an error in the submit server. After ”make clean” submit your work using the submit command.