CMPUT 379 Assignment 1

Chengxuan Li

2022-02-02

1 Objectives

This programming assignment provides some hand on experience in using Unix system calls for accessing and utilizing system time values, process environment variables, process resource limits, and process management functions. Additionally, this assignment provides a brief idea of the underlying structure and mechanism of a shell program that can interact and communicate between user and OS.

2 Design Overview

- The simple shell program uses modular design. All functions related commands locate in a separate source code file (commands.c, function prototypes are in commands.h). Error display functions (warning and fatal) locate in a separate source code file as well (handler.c, function prototypes are in handler.h). Lastly, all functions that provides general functionality such as splitting string, allocate memory, calculate time, etc., are located in utils.c.
- Each command not only provides functionality to meet the program specification but has some degree of error checking such as invalid input, invalid request (stopping / continuing a terminated process), etc.

3 Project Status

The current status of the project is that the simple shell program meet all the program specifications mention in the assignment descriptions. However, it is possible for some edge cases to cause the shell program to behave differently or produce unexpected results. Also, the implementation in current status of the project is not yet optimal although it behave as what the specification describes. One of the reasons is that some programming concepts and strategies as well as refactor techniques are difficult to apply using C language.

For cdir command, if there are environmental variables presented in the path name, cdir command will try to replace all the environmental variables with its value if it is possible. Otherwise, it will simply ignore the '$' in path name.

For stop command, if it tries to stop a zombie process that is not yet marked as terminated, the program usually complains that it cannot stop that process. That zombie process will most likely be removed by the system after SIGSTOP is sent. Consequently, lstasks command will still list that zombie process although it is removed by the system, and terminate command will complain when calling kill() with SIGKILL.

For run command, if there is no specified program name provided, it will only fork a process that will immediately return 0 from main function and become a zombie process. Also, after a child process is created, and it launches its corresponding, the parent process will wait one second to see whether if the child process terminate immediately after launching due to internal error (execvp error, or error inside the program) or completion of the execution. If so, the parent process will automatically mark that task as terminated.

For check command, check command will display the process based on the target id as well as its descendant processes. If a process has too many child processes, check command cannot display all its child processes. The limit for the number of child processes is set to 132.

One known issue is that if a invalid task No is provided, it is still possible for this invalid task No to bypass the error checking since atoi() function returns 0 if it cannot convert the task No from string to integer but 0 one of the task No if there are already tasks created. This issue can be solved by creating a separate function for checking invalid character in the string. However, it's not yet implemented in the current status of the project.

There are no specific signal handlers installed at the current status of the project for signal such SIGQUIT, SIGINT, etc.

4 Testing and Results

To tested my implementation, the simple shell program was tested on two different machines: lab machine and ManjaroI3, which is based on Arch Linux. The obtained results was almost identically as the output in the three examples provided in the eClass. The only difference was the notification displayed in stdout / stderr after an execution of a command. This applied to run, stop, continue, terminate and check commands.

5 Acknowledgements

List of sources of assistance:

- Stevens, S. Rago, Advanced Programming in the Unix Environment, 3/E, Addison Wesley, 2013
- The Open Group Base Specifications Issue 7, 2018 edition IEEE Std 1003.1™-2017 (Revision of IEEE Std 1003.1-2008)
- Linux Man Pages