Salma Rashed, Taya Wongsalong
CS433 Operating Systems
Dr. Xiaoyu Zhang
October 10, 2022

Programming Assignment 2 Report

Overview:

Our program uses UNIX commands like *fork()*, *wait()*, *execvp()*, *dup2()*, and in addition, we have created more commands and system calls to execute the program in the processes. The shell will wait for the user to input UNIX commands then the program will execute or terminate accordingly. The forking process occurs when the parent process first branches and reads the user input then the commands will be executed from the child process. The child process has to finish executing before the parent process can exit.

Class and Functions:

prog.cpp:

In our prog.cpp, it essentially acts as the main/driver of the program with functions including *redirectIN()*, *redirectOUT()*, *parse_command()*, *main()*, *userCommand()*, *saveCommand()*, *displayHistory()*. The shell keeps taking in the user's input until the user enters "exit" which then terminates the process.

- redirectIN(): this function opens the files and duplicates commands that the user enters from the file to the UNIX shell.
- redirectOUT(): this function writes to the files and also duplicates the output
- parse_command(): to parse the string from the user's input with spaces

- user_command(): this function has all of the UNIX commands that the user will enter; exit, !!, <, >
- saveCommand(): all of the commands are saved into the history
- displayHistory(): this displays all of the histories in the shell

Implementation:

As per the given starter code from the CS433 GitHub from the professor, as well as chapter 3 from the textbook. The main() method loops through the program using a while loop, waiting and listening to the user input. The while loop will keep looping as long as the user enters 1, otherwise, the program will terminate if the input is 0. In addition, the program first checks for the input of "!!" or *exit*. The program then either displays the last command that the user inputs or will exit the program. We parsed the string and check for the redirection operators (< or >). For our program, we were having issues with our input/output redirection so that function is not working properly. For the forking process, the program attempts the fork process using the pid that we assigned. We also used the *execvp()* function for the fork process, and it gets invoked when the child process has been forked.

Bugs in the program:

Our program does not pass the UNIX pipe test, both redirection of out > and input <, and lastly the running command with & test cases on gradescope. We have compiled it on replit, not the best choice, and it seems like it compiles, but on gradescope it fails the test case. I would say our program is 60% done, and it's not completed.

References:

cstring - strtok

C-programming - strcmp