Ben Foltz-Miranda

Dr. Zhang

CS 433-02

29 February 2024

# Assignment 2 Program Report

## Problem Description

The problem that I am solving is creating a simple UNIX shell that accepts user commands and then executes each command in a separate process. My implementation supports input and output redirection, as well as pipes as a form of inter-process communication (IPC) between a pair of commands. The UNIX shell also has a history of commands entered and allows the user to use the “!!” command to repeat the previously input command. This project uses the UNIX fork(), exec(), wait(), dup2(), and pipe() system calls and supports any Linux, UNIX, or macOS system.

## Program Design

In the main function, there is a loop that runs until the user inputs the command “exit”. Inside the loop, it prompts the user for input and then parses out the command and arguments from the input command by using the function parse\_command(). The parse\_command() function takes in a char array called command[] and a char pointer array called \*args[]. The parse\_command() function then uses the strtok() function to get each argument in the command[] array, this is done in a loop until the end of the command[] array is reached. In the loop, the args[] array gets allocated more memory so that the strcpy() function can be used to copy each argument into the array and ensure that args[] array is large enough to hold the command that is given by the user. At the end of the parse\_command() function the number of arguments is returned to the main function. After receiving the number of arguments (int num\_args) the main function checks if num\_args is greater than 0, if it is then it proceeds with parsing the command for special commands like “exit”, which exits the UNIX shell, “history”, which prints the history of commands, “!!”, which executes the last command by checking the history linked list and executing the that command and “&”, which allows the child and parent process to run concurrently. After checking for those special commands, the shell will use the fork() function to fork a child process, assuming that the fork did not fail, then the child process will begin to loop through the arguments to check for 3 special characters, “<”, which allows for input redirection, “>”, which allows for output redirection, and “|”, which allows inter-process communication. Assuming that none of these special characters are in the input, the shell executes the command by using the executeCommand() function, this function, uses the execvp() function. If the execvp() function returns a “-1” then the executeCommand() function prints “Command not found”. The shell then continues onto the parent process which waits for the child process to finish unless the command contains “&” at the end of the command, then it will run prompt the user for another input. The loop then begins again and prompts the user for input until the user enters “exit” to close the shell. I chose to keep the history of commands in a string vector since I have more experience working with strings as opposed to char arrays.

## System Implementation

I did run into some problems during my implementation, one of them was how to execute the last command in the history. Since I chose to store the history of commands as vector of strings, I couldn’t just pass the last element of the vector to execvp() since the execvp() function takes an array of pointers to a null-terminated string. I was able to solve the issue by creating a temporary char array that would hold the command from the history and then using strcpy() to copy the string into the char array and then running the parse\_command() function with that temporary char array as the command parameter and then having the rest of the loop run as normal so that the command could be run as normal. Another problem that I ran into was figuring out how to perform input and output redirection. I don’t think it was that hard to actually implement, but I had never used those functions and it was not the most intuitive experience trying to figure out how to get it to work but once I read more about the functions how they were supposed to be implemented in the code.

## Results

I believe that all the features that were required are included in my submission. I think that something that I would like to improve on would be implementing a way to allow for the use of an input and output operator to be used in the same command like in the textbook. I had not known about input or output operators prior to starting this project so I’m not entirely sure what would be something useful for me to use them for but I think it would just be interesting to have more functionality in this UNIX shell that I built. My UNIX shell passed all the test on gradescope, including the pipe operator test.

## Conclusion

I was able to solve the intend problem successfully. The program is able to successfully take in commands and execute them. A lesson that I’ve learned from this assignment is that I should read more about the functions that I am using so that I can figure out the solution to more problem quicker. I liked that this project was laid out pretty well in the book, so we just had to figure out a way to implement the basic ideas from the book and then make sure that it worked properly when ran. I think the only thing that I had a bit of trouble with was figuring out how to properly use the functions that the book told us to use, but in the end I was able to figure it out and get it to a point that I am satisfied with.