Nicholas Everekyan, Kevin Gallagher

CS433: Operating Systems

Professor Zhang

October 2cnd, 2024

Programming Assignment 2 Report

Analysis of Functions:

parse_command: This function tokenizes a string input into its respective command elements, delimiting by spaces. It also checks for the & symbol, which would indicate a command that would run in the background. It returns the number of arguments in the command.

Execute_command: This function executes a command that is passed to it, as well as the flag for if the process should run in the background or not. Utilizes execvp() for process execution.

handle_history: Utilizes a memset() buffer for storage of history of commands. Simply outputs the last inputted history command.

execute_pipe_command: Forks two commands from one command input, given that that command has the pipe operator. Follows the same process for each fork, that being to duplicate the file descriptors using dup2(), and executing the command using execvp(). Waits for both processes to complete and terminates the function.

find_pipe: Finds the pipe operator. Also splits command into two separate arrays, one for each command.

execute_command_redirection: Executes commands with any redirection operators, such as > or <. Checks for either of those operators, and with > writes to the output, and with <, it reads from the input.

Main analysis:

The functionality of the entire program is contained within a while loop, which checks for a number of things. First, we obtain the command from user input, while checking to make sure it is not blank, null, or some strange command. We add that command to the history, and then parse the command from the input. If the command is to exit, we set the flag to exit, and terminate the loop. We then set the flag for if the command is to run in the background or not, and then finally, execute a pipe command, or a command with redirection. The redirection command still works for commands even without redirection in those commands.

Concluding Remarks:

We ran into an issue with our parse_command() function, in which a command with an & symbol was not correctly recognized. This was mainly due to incorrect bound checking, however it did lead us on a wild goose chase throughout the entirety of the code. All in all, this shell did end up being straightforward, albeit somewhat complex at the same time. Due to the high usage of commands that already exist within the UNIX running environment, we did not specifically test for complexity nor time usage, although any potential problems would arise from the length of commands, as that is what we iterate through to parse the command, set specific flags, and check for specific inputs.