Jordan Sinoway

CSC345-02

Project 1 – User Interface

February 26th, 2021

The following is the outline of the program requirements for Project 1 – User Interface, and how and to what degree I completed them. As outlined in the assignment, the shell can create a child process and execute inputted commands in the child process, includes a history feature, input and output redirection, parent/child communication via a pipe, along with a few other simple features and error handling.

Program Requirements:

- Creating Child Process and Executing Command in Child

As outlined in the Project 1 description, a child process was created with the "fork()" command. This is handled by initializing process ids with the fork() process, and parent and child process can be created. If the pid > 0, this implies that the parent process is executing, and if pid == 0, the child process is executing. By checking the pid value, many of the other functions can be initialized, such as redirection and pipe operations, as well as another fork() for handling the pipe command.

- History Feature

The history feature in this shell is similar in concept to the history functionality in the base Linux shell, where hitting the up or down arrows on a keyboard will cycle through previously used commands. The history command "!!" calls the most recently used command from memory and runs it again. Whenever a command is run, it is stored to the "history" variable, so that it can be called again if the user inputs "!!". When the shell is first run, the history variable has a default value of 0, so that if "!!" is used before any other commands are entered, it returns "No commands in history" rather than segmentation faulting.

- Input and Output Redirection

In order to handle the redirection operators "<" and ">", the second to last argument in "args" is checked to see if an operator has been entered. If they are found, the "redirect_in" or "redirect_out" flags are set. In the child process, file descriptors are either declared and created for "redirect_out" or opened for "redirect_in" for the text file argument. Next, the dup2() function duplicates them either to the standard output or stand input. The text file arguments can then act as outputs or inputs for the command before the redirection operator. The in.txt file in the .zip contains an unsorted group of numbers to test the "<" operator, as well as the out.txt file to test the ">" operator.

- Parent/Child Communication via Pipe

Pipes allow for parent and child processes to communicate and for the output of a command to serve as the input for another. A pipe flag, “pipe_process” was declared to determine if the pipe operator “|” was used, along with a pipe file descriptor, “pipe_fd”. When the input arguments are tokenized, the input is checked for the “|” operator, and if it is found, the pipe flag is set to the value of the argument location. Inside the 1st child process, another child/parent process is forked, and the child uses dup2() to redirect the standard out to the pipe. After the child process is complete, the parent process executes the 2nd command, and the standard in is redirected to another element of the pipe. The arguments after the “|” are then executed, using the output from the first set of arguments as the input for the 2nd set.

- Other Features and Bugs

All standard commands are handled via the execvp() function in the child process, but there are a few commands that are specially handled. Using the standard change directory command, “cd” calls the chdir() function, and includes error handling for incorrect inputs.

The user can input “exit” to quit the shell and return to the terminal. This works in most cases, except for a bug where “exit” must be used an additional time for each command inputted. I was unable to debug this, as the exit(0) function I used should automatically exit the currently running program, and was not fixed by using other exiting functions, such as break(). Ctrl + C will still exit the shell, but this is not very elegant. I assume the source of this is runaway child processes, but I was unable to debug this.

Additionally, there is sometimes a bug when calling the “&” argument in the shell, as it will run the process in the background, but not fully return to the shell. I believe this is a similar problem to the previous bug, where the child process does not finish correctly.

I intend to continue working on the shell and attempting to debug it, as I have enjoyed working on the project.