Kyla Ramos
2/18/22

Project 1 Report

Child process creation and executing commands

I modified the main() function in Figure 3.32 in order to make a child process that forks and executes the command specified by the user. The main function will loop continuously unless exit is typed by a user. User input is taken being the command line, and is then parsed to check for specified commands, by storing the separate pieces of text or tokens (ex - cd folder) into different elements in an array, with args[0] storng the cd commnd, and args[1] storing “folder”. These are then later passed to the execvp() function, after parsing each separate token and storing them in the array. Also, it checks if an & is used, so that the parent process will wait for the child to exit.

Command history management

If a user wants to execute the last command entered, they are able to type !!. If no command was typed, the program will display a basic error message. Otherwise, it will store the last command entered into history, and proceed to display what it was and execute it again.

Add support of input and output redirection

In parsing the tokens, I check for either > or <. When the program finds one, it will set a flag variable for either in or out to 1. Depending on the flag, either input or output redirection will then execute. I utilized the dup2 function in order to handle both ways of redirection.

Allow the parent and child processes to communicate via a pipe

I implemented a pipe with the use of pipe(pid) and creating processes to handle the pipes. The | character is parsed into a token. If there is usage of the | character, then the parent process creates another child process that will execute the first argument before the |. The child also creates another child process that will execute the second argument after the |. It establishes a pipe between itself and the child process it creates. The dup2() is used as well.

Keep working directory information (change directory by [cd] command)

The cd command is implemented by first parsing the input to see if cd was typed in args[0]. If it was, then the folder that will be moved inside to via the cd command is used as args[1] for the input of the chdir function.

Up-to-date display the current directory in the prompt. For example, if you are in /home/user/my, then your prompt should show as osc:/home/user/my>instead of osc>

In order to keep track of the current directory, I used the getcwd function to keep track of the current working directory information. It is then displayed/updated with a print statement every time the loop of the shell runs.