CHINONSO EMMANUEL OKAFOR

OPERATING SYSTEMS

PROJECT #1- SIMPLE SHELL

Project Description

The project is one that creates my own shell scripts that work just like the standard shell in Unix. The project is divided into five tasks that are expected to be done and an additional task used for extra credit. Below are further explanations of what each task does

- I first print the shell prompt and add the built-in commands cd, env, pwd, setenv, echo, sand exit whereby cd changes the current working directory, pwd prints the current working directory, echo prints a message and the values of environment variables, exit terminates the shell, env prints the current values of the environment variables, and setenv sets an environment variable.
- I then incorporate other commands that aren't built-in commands so that they do what is expected when they are called in my shell. I do this with a child process and wait for the child process to complete before running the parent process.
- I then add a background process that runs in the background, without affecting the foreground process and immediately returns the prompt to the user to give a command for the foreground process.
- After this, I perform some signal handling that allows the user to exit the current running process and give the prompt anytime that Ctrl C is pressed by the user. At the same time, I ensure that the user leaves my shell and not the computer's shell when Ctrl C is pressed. To leave my shell to the computers shell, the built-in command, exist, as already explained is used
- Finally, I set a timer that allows at most 10 seconds for the process of each command given by the user. If the implementation of the command is completed in less than 10 seconds, we do not wait for the ten seconds to elapse, but immediately return the prompt to the user for another command. If the implementation time is more than 10 seconds, we exit the process at the tenth second and return the prompt to the user for the next command

Teammates

- Chinonso Okafor
- David Oluyomi-Lords

Design Choices for each Task

Task1

- After getting the command from the user, I split the command by space. When this is done, I check what the element on the 0th index is in the split array of arguments as it is at this 0th index that the built-in commands are. For each of the cases where the element at index 0 is a built-in command, I manually perform the function those built-in commands do

Task2

- If the element at the 0th index is not among the six built-in commands stated, I fork the parent process and if the current process is a child process, I use the execvp command to run the command as an executable. If it is a parent process, I wait for the child process to complete before continuing with the parent process.

Task3

- Once the prompt is shown to the user and the arguments are gotten, I get the last index of the array while appending each space-separated word to my argument array. I then check for what the element at that last index is. If it is &, then we are to perform this process in the background. For the background process, I use the execvp. If the last element in the argument array isn't &, I perform the actions already specified in tasks1 and tasks2 and update the parent process to only wait if the currently running process is a child process and not a background process

Task4

- For task 4, I use the SIGINT handler to know when the user had typed CTRL + C on the terminal. In the handler function, I kill the currently running process in order to handle long-running unconcluded processes and then call my main function in order to continue prompting the user for command and continue with my shell

Task5

- I use the SIGALRM handler to allow at most ten seconds for each process. I have an alarm for ten seconds and each time ten seconds have reached and the program hasn't terminated, I go to the SIGALRM handler function and kill the currently running process while calling my main function again to continue with my shell.