CS307 PA2 REPORT

Uğur Günal 30558

Overview:

My C++ program serves as a shell-like application that reads commands from a file, parses them, and executes them. The program has two main parts: Parsing and Execution of commands. In order to achieve concurrency, multiple processes and threads are used in the program.

Parsing:

For ease of use, I stored all lines in "commands.txt" in a struct called CommandLine. This struct stores all the properties that will be printed into "parse.txt," and it also contains some extra information like filename and argument length, which will be used later in the execution of commands part. I have used ifstream to parse the file. First, I got each line and then I pushed each word in that line into a string vector called commandinfo. Then, I created a CommandLine variable x and initialized all of its properties with the appropriate strings in the commandinfo vector. Finally, I pushed x into a CommandLine vector called alldata.

Execution of Commands:

All of the commands are stored in the CommandLine vector alldata, so I created a loop that iterates through every element in the alldata vector and does the proper implementation according to command type. I have separated command types into three main sections: wait command, commands including ">", and others.

- **Wait command:** The shell process iterates through the Threads vector and uses pthread_join to wait for all of the threads. Then, the shell process iterates through the Processes vector and uses waitpid to wait for all background processes.

- -**Commands including ">":** Since this command's output will be written into a file, there is no need for using threads. First, the shell process creates a fork. Then, the child process closes the standard output and opens the specified file. With this implementation, the output is automatically redirected to the file. After that, the child process executes the command with execvp. If this process is a background process, the shell process (parent) pushes the child process's ID into the Processes vector. If it is not, the shell process (parent) waits for the child process.

- **Other commands:** These commands' outputs will be printed into the terminal. To achieve the desired outputs and concurrency, threads will be used. The thread function takes an int pointer to a file descriptor as a parameter, reads from it, and prints the content into the terminal. The critical section of printing is protected by a mutex to prevent undesirable outputs due to multithreading.

  The shell process first creates a pipe in the heap, allowing communication between the shell (parent) and child processes. Then, the shell process creates a fork. If there is a "<" file redirection symbol in the command, the child process closes standard input and opens the specified file. With this implementation, the input is automatically redirected to the file. With the dup2 command, the child process also redirects its output into the write end of the pipe channel and executes the command with execvp. Then, the shell (parent) process creates a thread and pushes that thread's ID into the Threads vector. This thread takes the read end of the pipe as a parameter and prints all of the content into the terminal. As a summary shell process (parent) has taken the output of it's child process with pipe and then created a thread to print that into terminal. Lastly, if this process is a background process, the shell process (parent) pushes the child process's ID into the Processes vector. If it is not, the shell process (parent) waits for the child process.

After the shell process iterates all of the commands in the alldata vector, as in the wait command, the shell process waits for all of the threads and processes to terminate.