COEN346 - Programming Assignment #1

Team members:
Aryan Shirazi - 40119594
Karim Tabbara - 40157871
Julian Aloise - 40177178

Submitted to : Ferhat Khendek
Due: February 11, 2022

We certify that this submission is our original work and meets the Faculty's Expectations of Originality

COMMANDS AND STRUCTURE OF THE CLI

The goal of this assignment was to develop a **Command Line Interpreter (CLI)**. A command line interpreter interacts with the user through built in commands. A shell is a command line interpreter that one uses to access the services of an operating system. The CLI's shell consists of internal (built-in), included and external commands. The shell of the CLI can be invoked from the operating system's command lines. Internal commands are implemented under the shell thread (Shell.java). External and included commands are implemented under their own thread in another java class (Execute.java). Included commands are associated with the invocation of executables that are located in a directory path that is given to the CLI as an input from a text file (inputs.txt). The main function of the program reads all inputs from this text file and gives the readings to the shell. If the file that is requested for execution is not found in the path directories, the CLI will be able to execute those files using an external command which takes the absolute path of that file in the file system as input (AbsolutePath/Filename.exe).

USAGE OF THREADS

When the program starts its execution, it first runs the main functions that reads the input from "inputs.txt" and passes the content to the constructor of the shell. The shell object is then used to create a thread for the shell which is then joined. Joining the shell thread will put the execution of the main program on hold and will wait for the shell thread to finish executing before moving on to the rest of the main function. Once the shell thread is joined the program enters a continuous loop that will prompt the user for inputs. The loop will only stop when the user inputs the "exit" command. During the execution of the shell, the CLI may take any of the three types of commands. If the command given is not an internal command, the shell will create a new object of class Execute and will start a thread that will be joined if execution is happening in the foreground. In order to execute included/external commands in the background, the Execute thread must not be joined. This will go back to the top of the loop and prompt users for input while execution is happening in the Execute thread. The execute class constructor takes in two parameters; one is the name of the file and the other is the path directories. The path directories are only given for the sake of checking if the command is included or not. To elaborate, if the filename is found in one of the directories (by means of a linear search of an array containing all executables located in a directory) then the command will be of the included type. Otherwise, if the executable is invoked from it's absolute path in the file system, then the command is external.

The following constructs were created to allow users to interact with the shell program using the "echo" command. Knowing that there are three different variations of that command, we needed to implement a method for the system to figure out which variation to use ("EchoCommand()").

"If" statements are used to examine the user input:

- Case 1: if the input contains only the "echo" command, but not the "->" or "->>" commands.

The string the user wishes to display on the console is retrieved using a function "echo1()" and placed into a new variable. We then simply display that string using basic java printing methods (System.out.println)

- Case 2: if the input contains the "echo" command and only the "->" command.

First, the system finds the index of the tip of the arrow ">" using the indexOf() method of Strings. Using that index, the string the user wishes to write to a file of his/her choice is retrieved and placed into a new variable, along with the file name which is also placed into a new variable. Then, the implemented SingleArrow() method is invoked to write the information to the file using a simple FileWriter. The string is not displayed into the console.

- Case 3: if the input contains the “echo” command and only the “->>” command.

Similarly to the 2nd case, the index of the second tip of the arrow “>” is found using the lastIndexOf() method of Strings. Using that index, the string the user wishes to append to the file is retrieved and placed into a variable, along with the file name which is also placed into a new variable. Then, the implemented DoubleArrow() method is invoked to append the information to the file using a simple FileWriter. The string is not displayed into the console.

The final command is “command &”. Since all the commands are executed in the foreground by default, this command will execute the commands in the background. Running the command in the foreground requires the shell not taking any other commands until it is done executing the previous command. On the other hand, running the command in the background does not require the previous command to finish executing before prompting users for input.

The command “exit” is used to exit the shell program. The exit case is written in a while loop. Once the user inputs exit into the terminal, the while loop’s condition will change from true to false. This will exit the while loop and close the terminal.

The inputs read from the “inputs.txt” file are given to the constructor of the shell which produces a shell command line by displaying, “username@hostname$” and saves the comma-separated list of path directories in a variable PATH which will be passed to the constructor of the Execute.java if an included/external command is to be run. The user is then prompted to enter a command. The program will continue to prompt the user to enter another command until the user enters the “exit” command.

A thread, also known as a lightweight process, is a path of execution within a process. Threads are used to allow the processor to perform multiple tasks at the same time. Three threads are created in this assignment. A thread is created when running the main program. The process running the shell program creates a thread. For every invoked internal or external command, a thread is created.

DEMO TEST CASES

- Echo: echo “testing coen346 assignment 1”
- -> : echo “this is a “ -> testfile.txt
- ->> : echo “demo for our coen346 assignment !” ->> testfile.txt
- Included command: Filename.exe (where Filename.exe is located in one of the path directories)
- External command: AbsolutePath/Filename.exe(will specify location of executable in file system)
- Filname.exe & : invocation of executable with path directory in the foreground
- AbsolutePath/Filename.exe &: : invocation of executable in file system in the foreground
- NonExistentFile.exe : if file does not exist in path directories, “file not found” is displayed
- External command: AbsolutePath/NonExistentFile.exe if file does not exist in its specified location “file not found” is displayed

CONCLUSION

In light of the implementation of the CLI in this programming assignment, we have used multithreading to run the shell and provide the possibility of invoking executables in both the foreground and background. The executables are only invoked in the background when the thread of the Execute.java class is not joined and the shell goes back to prompt the user for inputs simultaneous to the invocation of executables. This gives us insight on multithreading in other areas of computer science where multiple threads may run different processes simultaneously and in parallel to each other.