**Design and Algorithm**

This program is centered around the functionality of the shell which Is to behave like a Linux command interpreter. The key functionalities of the shell are to 1) display a prompt to the console – indicating that the shell is ready to accept a command from the user. 2) read the line of input from the console. 3) spawn and have a new process execute the user’s command. The Shell works in unison with ThreadOS which is an operating system simulator. ThreadOS also contains a system library called SysLib which includes many system call functions that are used throughout the shell program.

The shell program consists of two key methods: run() and generateCmd().

The run() method starts by printing out the command prompt in the form “shell[line#]%” using the SysLib.cout() system call. SysLib.cin() is then used to get the user input. The line number in the command prompt increments with every line of input. The input is then converted to a string to pass to the SysLib stringToArgs utility function. This function takes each space delimited string of input and stores each one into a separate string array element.

We then take this string array and start traversing it. I check to see if the current index equals either ; or & or is the last element in the array. if it is, we start to generate the command using the generateCmd method.

The generateCmd method takes in an array of strings, a starting index, and an ending index. in this method I loop through the passed in array from the starting index to the ending index and then copy those elements into another string array which is then returned. This method is meant to extract the specific command from the larger input.

Now that I have got the command extracted. I start off by checking to see if command[0] equals the string “exit.” If it does, I then call SysLib.exit() to tell ThreadOS that we are now done. I also return to break out of the program. Otherwise, it continues to run. Now, if command[0] does not equal “exit” then we execute the command with SysLib.exec(). The exec system call takes the command and starts it as a child thread. I use a bool to check if exec was able to run without any errors. true = no errors and false = error.

Now we are left to check delimiters. A command followed by the delimiter & means that the command(s) would run concurrently. A command followed by the delimiter ; means that the command(s) would run sequentially. If the command is delimited by & then we don’t necessarily have to do anything as exec is already running. On the other hand, if the command is delimited by ; and exec did not have an error, then we need to call SysLib.join() which waits for the termination of one child thread before starting the next one

This loop is continued until there are no commands left to be parsed. We then go to the next line where the command prompt is printed and waiting for new user input.

**Flow Chart**

Chart, box and whisker chart

Description automatically generated

**Testing**

The program is started using the commands “javac Shell.java” and then is booted with the command “java Boot” to which you are brought into the ThreadOS Loader where Shell.java is invoked with the command “l Shell.” The command prompt is then printed to the console. Now we are ready for some input and testing. Once you see the command prompt, simply type in your command. If it is valid the output will be printed to the console and the prompt will be printed to the next line waiting for more input.

The shell can be tested with various PingPing commands which can be followed with any variation of delimiters.

Ex: PingPong abc 10 ; PingPong xyz 14 & PingPong eee 56 & PingPong www 29 ;

**Delimiters:**

* A command followed by the delimiter ; means that the command(s) would run sequentially.
  + In the example: PingPong abc 10 ; PingPong xyz 10 ;
  + A thread/process for abc would be created and abc would be outputted to the console. Then another thread/process would be created and xyz would be outputted to the console. Basically, the first command needs to finish executing before the second command can start executing
* A command followed by the delimiter & means that the command(s) would run concurrently
  + In the example PingPong abc 10 & PingPong xyz 10 &
  + Threads/processes for abc and xyz would be created at the same time. This basically means that abc will be executed and xyz will start in the background right away even if abc has not finished executing. Both abc and xyz would be outputted to the console at the same time.

If an invalid command is inputted, such as a delimiter by itself, an operator with no command, or only one command, the shell throws an exception but still moves onto the next line – waiting for more user input. In the case that a blank line is entered, the shell does not throw an exception and just moves to the next line – waiting for user input.

**Some examples of invalid commands**

* **PingPong not case sensitive:** pingpong abc 10 ;
* **Missing part of command:** PingPong abc &
* **Operator with no command:** ; or &

\*in all of these cases, an exception is thrown and the shell should move to the next line\*

**Outputs**

1 valid command: both concurrent (&) and sequential (;)

A picture containing text, close

Description automatically generated

2 and 3 valid commands run concurrently

A picture containing background pattern

Description automatically generated

2 and 3 valid commands run sequentially

A picture containing text, screenshot

Description automatically generated

A mixture of concurrent and sequential – all valid commands

Graphical user interface

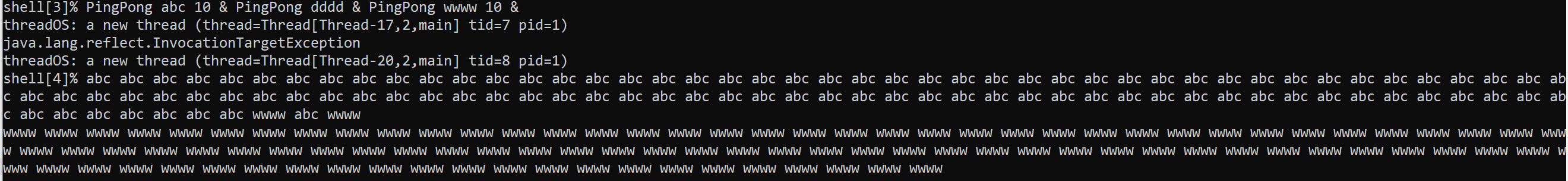
Description automatically generated

Mixture of valid and invalid commands : PingPong not case sensetive

A picture containing text, indoor, screenshot, several

Description automatically generated

Mixture of valid and invalid : not a proper command



Invalid commands – throws exception and moves to next line – waiting for more input

Text

Description automatically generated

An empty line – simply moves to the next line waiting for input

A black screen with white text

Description automatically generated with low confidence

Delimiter is entered by itself

Graphical user interface, application

Description automatically generated

Exit is entered – back into the loader. Type q to fully exit loader.

A screenshot of a computer

Description automatically generated with medium confidence