Iterative Socket Server

Donnie White

Jordan Gallivan

CNT4504 – Computer Networks and Distributed Processing

Professor Kelly

**Introduction**

The purpose of the project is to create a project that allows for a client application to connect to a socket server. The client application should be able to send several different requests to the server which will then in turn return all necessary data that the client requested. The client should also be able to send multiple requests to the server at once in intervals of 1, 5, 10, 15, 20, 25 requests. These requests should include Date and Time, Uptime of the server, memory usage of the server, netstat, current users, and running processes. The goals of these two programs are for them to work seamlessly together, for the server to return accurate data to the client, and making the client as aesthetically pleasing as possible all while writing a clean, efficient source code. Throughout this paper, we will discuss the design and design decisions of both the client and the server programs, how both programs were tested, data analysis of the effectiveness of the server in relation to the number of clients connected, as well as a out conclusion and the lessons that we learned throughout this process.

**Client – Server Setup and Configuration**

The server consists of two classes which include the main server class as well as a class named Server Helper (SH). The SH class consists of two parts that aid in the execution of the main server class. These two parts are as follows:

Execute System Command: The Execute System Command method first takes a string named command in and creates a string builder object that is modified throughout the method. It then creates a process that is equal to Runtime.getRuntime().exec(command). This process passes the string into a command line terminal to get some necessary information about the machine the server is running on. The method also creates a Buffered Reader to read in the information from the command line terminal.

Send Message: The send message method passes in a String named, “messageToSend” and a buffered writer named, “writer”. We use the writer.write(messageToSend) command to send the message to the client.

The main server class takes input from the user to define what port the server should be running on and then waits for a client to connect using that port. It then creates a server helper object, buffered reader, and buffered writer. The server also creates a string constant named messageToSend. The server class uses the buffered reader to read the input from the client and then checks that input in a switch case for each expected input and performs the following operations for the respected input:

Date and Time: This case uses the java.util.date to return the current date and time and sets the messageToSend string to the value returned.

Uptime: This case uses the execute system command method from the SH class to execute the Linux command, “uptime” and sets the value of messageToSend to the value returned by this.

Memory Use: This case returns Runtime.getRuntime().totalMemory() – Runtime.getRuntime.freeMemory() and sets the value of messageToSend as the value returned from this.

Netstat: This case uses the execute system command method from the SH class to execute the Linux command, “netstat” and sets messageToSend to the value returned.

Current Users: This case uses the execute system command to execute the Linux command, “who” and assigns the value returned to the messageToSend string.

Running Processes: This case uses the execute system command to execute the Linux command, “ps” and sets messageToSend to the value returned by the command.

Default: As a redundancy, we set the default to send a message “NO MATCHES FOUND FOR CLIENT INPUT” back to the client as well as prints it out to the server terminal. While the client should catch all unexpected inputs, we opted to implement this redundancy to further prevent unexpected inputs into the server.

Finally, if the messageToSend is null, we again print and send an error message to both the server and client. We implemented this as a redundancy as well to prevent null values from being sent back to the client which could ultimately cause the client to crash. Otherwise, if the messageToSend is not null, we send it using the SH sendMessage method passing in messageToSend and the buffered writer as arguments. We then send an, “END” message in the same fashion as about to signal to the client that the server is done transmitting data and is ready for another request.

The Client

Testing and Data Collection

Data Analysis

Conclusion

Lessons Learned

One lesson learned was how to execute Linux commands through the program. We initially were trying to call socket commands to get the information needed. Once we learned of executing Linux commands through the program, we spent some time familiarizing ourselves with not only the commands themselves, but also how the runtime commands worked. After some troubleshooting and bug fixing, we were able to create a function to execute the commands.

Some other problems we had to overcome were differences between the Java version local to our computers, and the version that was running on the course server. Some code worked locally but not on the course server, so we had to update portions of our code to the correct Java version.