Iterative Socket Server

Lupe Ramos and Jinel Johnson

CNT4504 - Computer Networks & Distributed Processing

John Kelly

Introduction

The purpose of the iterative server project is to allow students to observe the impact of increases in requests on a server’s response time. The goal of the project is to develop a single-threaded Linux server that will take requests from a number of clients. In addition to the server, the team members must also create a multi-threaded client, which is used to connect to the server and send requests. The remainder of this paper serves to explain the methods used to achieve the projects goals, as well as show the resulting performance degradation relative to the number of client requests.

Client-Server Setup and Configuration

The server program listens to the specified port and processes requests sent by the client. It makes use of the java.io package to read the client’s input as well as return the result of the requests. The server also makes use of the java.net package to establish a connection to the specified port. The user has to input a port number that the server must listen to. The program checks the value that the user inputs to ensure that it is a valid port number, which ranges from 1025-4998. Once the user inputs an acceptable comm port, the server program attempts to create a server socket at the specified port with a limit of twenty-five requests, which was the maximum number of requests specified in the project description.

Once the server detects a request from the clients it will run the requested Linux command and return the result. The program retrieves the command through the “getClientChoice” method, which reads the client’s input and returns the input to the main function. The user’s choice is then passed to the “handleRequest” function. handleRequest uses a switch case to select the appropriate command string. This command string is passed into the “getCmdOutput” function, which runs the command and returns the command’s result to handleRequest, which returns the result to the main function. This result is then passed to the “printToClient” function, which returns the result of the request to the client. In addition to processing requests, the server also keeps a count of all requests it receives and displays this information every time it completes a request.

The client program generates multiple requests to the server based on arguments given by the user and calculates the turnaround time for each individual request, as well as for all the requests combined. It makes use of the java.time package to get the request turnaround time, java.io to get input from the user and pass requests to the server, and the java.net package to establish a connection to the appropriate port. The program begins by asking the user for the address and port number of the server. The user is then asked to insert a request choice, ranging from 1-6, and a number of clients. The port number, request choice and client numbers are all validated as they are inputted before sending the request.

The program then creates a number of threads equal to the number of clients requested and puts the clients into an array. The threads are created using the “RequestThread” class. This class also attempts to send the request to the server and times the turnaround time for the request. Each request’s turnaround time gets printed to the terminal and is also added to the total time using the “addToTurnSum” function. Once all the threads are created, each thread in the array waits using the “join” method. This waiting is necessary to ensure that the turnaround time is not artificially lengthened by having the thread wait for other threads to complete their processes. Once all requests have been processed, the program returns the total time and the average time per request.

Testing and Data collection

The server was tested by running each combination of clients and requests five times. First, five instances of date and time were run with one client, then five instances of date and time were run for five clients and so on until all six data sets were collected. The reason for using five trials was to prevent outliers from skewing the results as much, which is clearest on the single client testing. Once all the testing was complete, the five values for each client and request combination were averaged, which is the data present in the accompanying charts. The turnaround times for each of the following trials is measured in milliseconds.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Date and Time | 1 | 5 | 10 | 15 | 20 | 25 |
| Trial 1 | 16 | 5 | 5 | 11 | 6 | 11 |
| Trial 2 | 1 | 2 | 12 | 19 | 35 | 5 |
| Trial 3 | 1 | 4 | 9 | 11 | 8 | 12 |
| Trial 4 | 1 | 6 | 11 | 10 | 8 | 13 |
| Trial 5 | 1 | 8 | 8 | 14 | 15 | 15 |
| Average | 4 | 5 | 9 | 13 | 14.4 | 11.2 |

The date and time requests show the importance of running multiple trials, as the first trial of a single client was much higher than any of the other requests for a single client. This may have been because it was the first command ran by the server and it needed more time because of that. Each trial value is the return time in milliseconds with the column header values being the number of requests sent. While most increases in clients increased the average time, the trial of 25 clients had a lower average than both the 15 and 20 client trials.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Uptime | 1 | 5 | 10 | 15 | 20 | 25 |
| Trial 1 | 2 | 3 | 9 | 13 | 19 | 15 |
| Trial 2 | 1 | 3 | 14 | 8 | 19 | 17 |
| Trial 3 | 1 | 2 | 9 | 7 | 20 | 17 |
| Trial 4 | 1 | 3 | 3 | 13 | 21 | 18 |
| Trial 5 | 1 | 3 | 7 | 10 | 25 | 14 |
| Average | 1.2 | 2.8 | 8.4 | 10.2 | 20.8 | 16.2 |

Just as with the date and time trials, the 20 client trials had the highest average time. This set also showed much more erratic growth than date and time, as well as a larger range of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Memory Use | 1 | 5 | 10 | 15 | 20 | 25 |
| Trial 1 | 2 | 7 | 5 | 15 | 22 | 15 |
| Trial 2 | 1 | 8 | 7 | 13 | 19 | 16 |
| Trial 3 | 1 | 6 | 12 | 12 | 12 | 13 |
| Trial 4 | 1 | 7 | 9 | 14 | 7 | 13 |
| Trial 5 | 1 | 5 | 7 | 10 | 13 | 23 |
| Average | 1.2 | 6.6 | 8 | 12.8 | 14.6 | 16 |

The memory use trial is the first trial to have each subsequent set have a larger average turnaround time. This set also has two primary time increases: one at five clients, and another at fifteen clients.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Connections List | 1 | 5 | 10 | 15 | 20 | 25 |
| Trial 1 | 7 | 12 | 30 | 68 | 141 | 170 |
| Trial 2 | 3 | 13 | 26 | 74 | 154 | 182 |
| Trial 3 | 3 | 15 | 40 | 79 | 137 | 194 |
| Trial 4 | 4 | 17 | 47 | 119 | 172 | 263 |
| Trial 5 | 3 | 21 | 38 | 114 | 170 | 312 |
| Average | 4 | 15.6 | 36.2 | 90.8 | 154.8 | 224.2 |

The connections list has a significantly larger range of times than any other set of trials. This is likely due to the fact that each time the number of clients increases, it has to print exponentially more lines of output. It is also notable that this is one of the two requests where the single client trials are consistently higher than one millisecond.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Current users | 1 | 5 | 10 | 15 | 20 | 25 |
| Trial 1 | 1 | 7 | 8 | 5 | 18 | 26 |
| Trial 2 | 1 | 6 | 7 | 9 | 7 | 11 |
| Trial 3 | 1 | 2 | 4 | 13 | 13 | 7 |
| Trial 4 | 1 | 3 | 4 | 5 | 7 | 15 |
| Trial 5 | 1 | 3 | 14 | 13 | 18 | 13 |
| Average | 1 | 4.2 | 7.4 | 9 | 12.6 | 14.4 |

The current users trials again followed the expected trend of the higher client counts taking longer than the shorter counts. This set is unique in that it is the only set to have all single client times be the same. This could be attributed to random chance, or it could be because there was only one user logged in at the time. It is also likely that the average time of this request will change dramatically based on the number of users, but it should still maintain the same general trend.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Running Processes | 1 | 5 | 10 | 15 | 20 | 25 |
| Trial 1 | 5 | 15 | 24 | 38 | 44 | 51 |
| Trial 2 | 4 | 12 | 25 | 33 | 41 | 53 |
| Trial 3 | 4 | 13 | 23 | 27 | 43 | 53 |
| Trial 4 | 4 | 9 | 22 | 37 | 37 | 58 |
| Trial 5 | 4 | 16 | 22 | 33 | 44 | 57 |
| Average | 4.2 | 13 | 23.2 | 33.6 | 41.8 | 54.4 |

The running processes trials are the most linear set of values in of all he requests and also had the second largest time range of all trials. Additionally, it is the trial set where the single client average is the highest.

Data Analysis

The increases in clients do increase the average turnaround time for most instances. Because the average increases that means that each individual within the set has also increased. It is worth noting that, while the average time increases for each increase in requests, the increase is not direct. Five clients are not five times longer than one client, ten is not twice as slow as five, and so on. It is also noteworthy that the specific request had an impact on a couple of the trial sets.

Conclusion

It is evident from the data that an increase in clients is the primary cause of longer turnaround times. We can also conclude that some commands naturally take longer to complete than others. The data also shows that a single client request can be completed “instantly” in the sense that they almost always take a single millisecond to complete.

Lessons Learned

One important lesson learned is how to compile and execute java files in Linux. This project was also helpful in understanding uses of certain Java packages as well as learning how to execute Linux commands using Java. Lastly, it was interesting to learn that some commands take longer to complete than others.