Iterative Socket Server

Furio Gerwitz and Shane Peterman

CNT 4504 - Computer Networks

Professor Scott Kelly

Introduction:

The purpose of the Iterative Socket Server Project was to develop a client-server configuration with a group of fellow students. Our group of two divided the project into two parts, those being the server and the client. Trey Cox developed the single-threaded server and John Lowery developed the multi-threaded client. When each of them finished their individual development processes, they tested their code together to see if there were any errors between the server and the client. The purpose of this project was not only to develop the client-server configuration, but also to learn to work together to accomplish a large and difficult programming goal. Computing professionals, regardless of the field they are in, will have to work with other computing professionals at some point during their careers. Learning how to split up coding work and implement code that can be easily understood by others are two important skills that each computing professional must have.

The Iterative Socket Server Project consisted of several goals. The primary goal of the project was to develop a client-server configuration. As this goal was quite large, our group divided it into several smaller, but easier to manage, goals. Our first goal was to split the work between the two of us and develop our sections separately, so the whole team would be contributing equally to the project. After separate development was complete, our next goal was to combine our individual work to create a complete client-server configuration. This included troubleshooting both parts of the project in order to create one cohesive configuration. Once we have confirmed that the client-server configuration was fully developed and did not have any bugs, we were able to complete our last goal, which was to write a report of the project as a whole.

The rest of this paper consists of several sections. The first section will go into the specifics of the client-server Setup and how it was configured. The second section will go into how the client-server configuration was tested and how data was collected. The third section will explain the data that was previously collected. The fourth section will list any conclusions that were made from our data. The fifth and final section will discuss the lessons that we learned as a group throughout this project.

Client-Server Setup and Configuration:

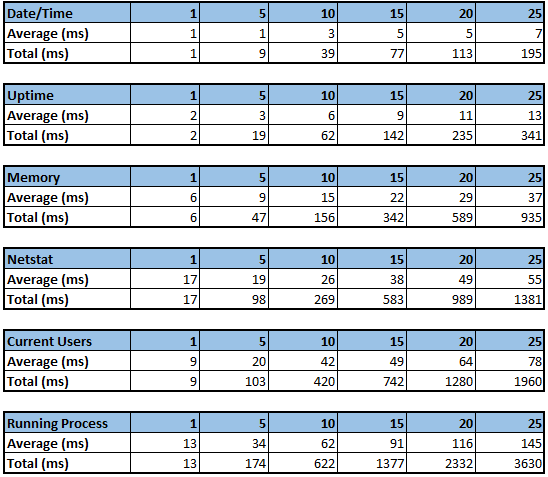
The setup for the Client-Server uses a single-threaded server that runs on a single port and a multithreaded client-side that can send a chosen number of threads in each iteration. The server side of the project starts by allowing the user to call the program on the Linux machine and declare a port that the server-side application will run on. Once the serverside application is running it waits for a client to connect. Once a client has connected to the server-side application the server echos a command that notes the connection. Once the connection to the client-side application has been made and the client starts to run different threads the server-side application opens a new socket for each thread and then runs then listens for the information the client-side is sending over. It takes the received information and runs a given function based on the information given. It then writes the information requisition back to the client-side application and closes the socket, waiting for the next thread.

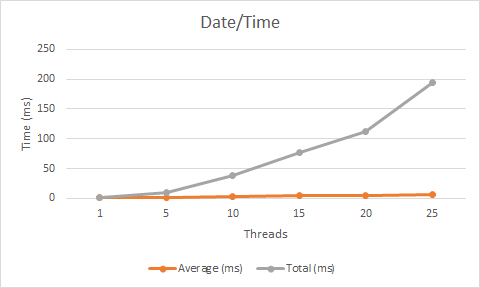
The other side of the project is the client-side application. The client-side of the application is started after the server side has been started and the client-side of the application needs to be given the server IP and the port number the server is running on to connect. Once the connection has been successful the client-side application asks the user to enter a number 1-7 to send a request to the server for specific information or to exit the application. If the user enters a number 1-6 the client-side application asks the users how many threads they want to make. Then the application calls a constructor class that creates the threads and hold a run function and it starts a loop, based on the number of threads the user wants, and in that loop, it starts a timer, writes to the server the number for the information it wants, listens for the information, prints the received information for the user to see, stops the timer, and ends the thread. After that loop runs the average time is given back to the user and then the menu of choices is prompted again for the user. Once the user chooses to exit the client-side application ends its connection and the server-side application shuts off.

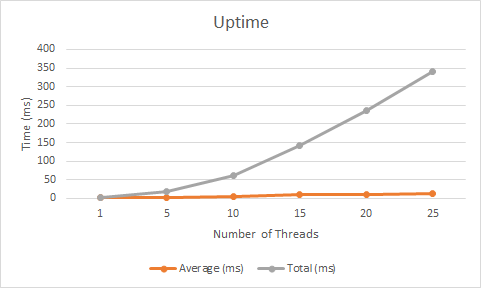
In the process, we chose to only send strings of the number the user chose from the client-side application and then have the server-side application change those into integers and choose the function because of those integers. This choice was made to simplify the reading process of the passed information back and forth as both sides of the application would only pass strings to the other side.

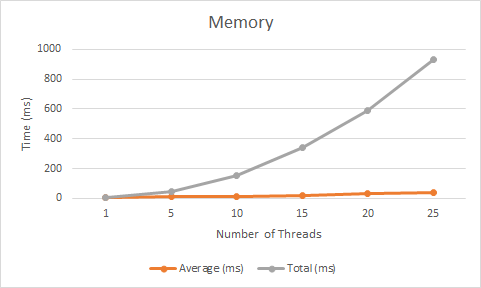
Testing and Data Collection:

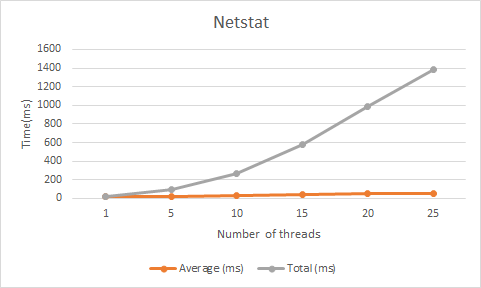
The Iterative Server was tested after it was complete and any bugs were fixed. Testing occurred in such a way as to make sure all of the project guidelines were met. Each different type of client request (Date and Time, Uptime, Memory Use, Netstat, Current Users, and Running Processes) was tested with all of the potential client request use cases as well (1, 5, 10, 15, 20, and 25). All of the results regarding turn-around time from these tests were then recorded.

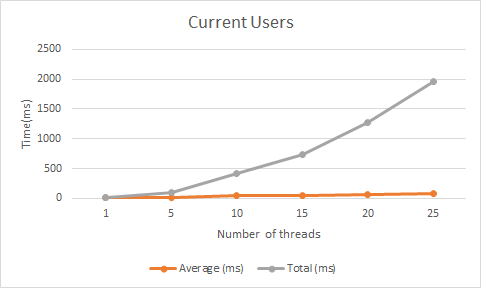


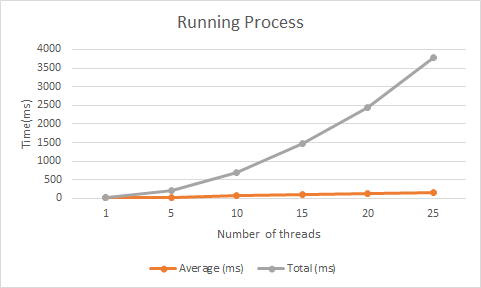












Data Analysis:

Increasing the number of clients will increase the turn-around time for individual clients and the average turn-around time for clients as a whole. The average turn-around time consistently increases as the number of clients increases. This could only mean that the turn-around time for individual clients is increasing as well.

The primary cause of the increase in turn-around time and average turn-around time is due to the increasing number of requests to the server. The server will take more time to process requests if there are more requests on the queue. This effect will steadily increase, as threads have to wait for other threads to finish before they can start, causing the whole thing to slow down gradually.

Conclusion:

Based on the data that was analyzed previously, there were a few clear trends. As mentioned in the previous “Data Analysis” section, an increase in client requests results in a higher individual turn-around time for clients, as well as a higher average turn-around time for clients. Another notable trend was the increase in times based on the varied client request types. In order of lowest times to highest times, it went Date and Time, Uptime, Memory Use, Netstat, Current Users, and Running Processes. This increase in times across different client requests is primarily due to the difficulty of each of the different requests. It is significantly easier for the server to get the date and time, compared to the server checking all of the processes that were run.

Lessons Learned:

There were several lessons learned in the process of this project. The first lesson was learning how to build client-side and server side applications in java. We also learned how to debug a two-part java application. One of the major problems during this assignment was getting the parts to communicate properly while building them separately. After several rewrites, we were finally able to make both parts communicate and send the correct information. On the server-side, we learned how to use Linux commands in a java application, and on the client-side, we learned how to use timers and the run function in the constructor class.