**Testing and Data Collection**

Similar to the Iterative Socket Server, the Concurrent Socket Server was tested by running the program multiple times for each listed operation (Date and Time, Server Uptime, Memory Usage, Netstat, Current Users, and Running Processes) given 1 request, 5 requests, 10 requests, 15 requests, 20 requests, 25 requests, and 100 requests. For each individual request given, the response time was recorded in milliseconds. The Total Elapsed Time and the Average Elapsed Time was then calculated and displayed through line graphs.

The data collected compares the Total Elapsed Time and the Average Elapsed time for each operation. The Total Elapsed Time was calculated by taking the elapsed time for each individual request. The Average Elapsed time was collected by taking the Total Elapsed Time and dividing that by the number of requests given. Both the average and total were calculated for 1 request, 5 requests, 10 requests, 15 requests, 20 requests, 25 requests, and 100 requests. Additionally, the graphs provided for the Concurrent Socket Server will be overall compared to the graphs provided for the Iterative Socket Server.

\**Note*: each individual request time was not listed due to length purposes, however running the program will readily provide this information

\**Note*: for each graph of each operation shown below, the Concurrent data graphs will be placed above the Iterative data graphs.

Average Elapsed Time for each Operation (Concurrent and Iterative):

[INSERT GRAPHS HERE (concurrent on top of iterative)]

Total Elapsed Time for each Operation (Concurrent and Iterative:

[INSERT GRAPHS HERE (concurrent on top of iterative)]

# **Data Analysis**

Based on the Concurrent graphs shown above, for both the Average and Total Elapsed time, they all share a similar pattern – a very slight and steady increase (with some fluctuation) with each number of requests until it reaches 100 requests, and the graphs see a major time spike. However, the reason for this spike is because of the gap of the number of requests between 25 and 100. Based on these graph patterns, it is clear that as the number of requests go up, the time increases, but very slightly and steadily.

Increasing the number of clients gives little to no effect on the total turn-around time for individual clients. In the same respect, increasing the number of clients gives little to no effect on the average turn-around time for individual clients. The reason is because all of the client’s requests are attended to simultaneously through multiple threads from the server instead of having the wait one at a time, which increases the response time significantly. Although the time still increases with an increasing number of clients, the time increase is very minimal and it still does not affect the clients’ since their requests are taken care of simultaneously.

Compared to the average turn-around time of the Concurrent server, the Iterative server’s average turn-around time shows a linear increase, while the Concurrent server’s turn-around time shows a more steady and flat line. This shows that an Iterative server experiences increased time delays with more clients while the Concurrent server does not.

This shows that a Concurrent server is good in situations that need to handle many clients with many requests as the server has better efficiency and will not bog down over time. The Iterative server is good in situations that only need to handle one or a few clients as its efficiency is not the best and adding more client will only slow down the process further.

# **Conclusion**

Based on the data analysis, it is safe to conclude that having a multithreaded server is really the best option to have, whether working with a single or multiple client, due to its much better efficiency. Although a single-threaded server is still usable and works, just as said in the previous conclusion of the Iterative server, it has become obsolete with today’s standards as modern technology heavily supports multitasking, thus requiring the capabilities that multithreading has to offer.