Concurrent Server Assignment

Chris Hoyt, Chris Deardeuff

CNT4505-Computer Networks & Distributed Processing, Professor Kelly

Table of Contents

[Introduction 3](#_Toc38829420)

[Client-Server Setup and Configuration 3](#_Toc38829421)

[Testing and Data Collection 4](#_Toc38829422)

[Data Analysis 6](#_Toc38829423)

[Conclusion 7](#_Toc38829424)

[Lessons Learned 7](#_Toc38829425)

# Introduction

The purpose of this project was to get an introduction into programing and limitations of a concurrent socket server. The goal of the assignment was to demonstrate how the response time of the concurrent server is lower than an iterative server as more clients send requests. This paper will outline the process we took in programing, the results we got from the server and other collected data, and the lessons that we learned from this project.

# Client-Server Setup and Configuration

The server-side program was designed using multithreading allowing the server to handle multiple clients at a time. At startup, the server prompts for an IP Address and a port number for the server to listen on. A socket is set up and waits for a client to connect, and when a client connects it opens a thread on the socket and begins waiting for an input from the client. If multiple clients try to connect to the server it will open multiple threads allowing the server to process the inputs from the clients more efficiently. When a request if received from a client the server will take it through a switch case and execute the corresponding action, returning the requested information to the client.

The purpose of the server-side program is to sit and listen for a client or multiple clients on a specified IP Address and port. When the server sees a client, a thread is open for the client allowing for more efficient return on sending requests back to multiple clients.

The client program takes the user input for how many, and what type of request, and then will create that many clients and start their threads. Each client thread connects to the server and receives its information from the request and then prints to the console. The client generator is coded so that it uses a do while loop to continuously run the program until the user says quit. Inside the do while loop, it prompts the user for the server address, the port that the server is listening on, the number of desired clients, and the type of request to send. After the user enters all that information the program will create that many clients using a client object that inherits the Thread class which allows them to run in parallel. After the client objects are all done running, the do while loop executes again. The only changes that were made to the client generator for this project was the adding of the decrease counter method if the running server thread returned an error. This allowed the client thread safety methods to signify that all clients were ended, and it could tally the total and average run times for all clients.

When making the multi-threaded server, not much changed in terms of how processing the client requests went. In terms of the multi-threading, a separate class was used to create the server threads. The Server Thread class holds the switch case that is used to process client requests from the original single threaded server.

# Testing and Data Collection

The Concurrent Server was tested by pushing one to one-thousand clients at it. Each client would record the time it took the server to return the process and then give the client-side console a total time and an average time that the server took to send the process it was asked for.

The data shows a quadratic growth with some larger increases during the higher number of requests, similar to the iterative server from the last project. However, this growth while large in a relative scale, is tiny compared to the growth the Iterative version of the server saw. It also did not occur until significantly more clients had been generated and sent requests. The following charts show the time taken for each request a client could send over an iterative server, versus the concurrent servers’ data. The charts on the left are Iterative servers data while the charts on the right hand side show the concurrent servers data.

# Data Analysis

While increase the number of clients on an iterative server drastically increases the time taken to return the requested data, it is on a much larger scale than the concurrent server. The charts above show that the times of the concurrent server with one thousand clients, is roughly the same as an iterative server with twenty-five clients. The reason the concurrent server can handle so many clients faster is that it is processing the request of the clients at the same time on different threads; whereas the iterative server is one by one, meaning that if one client is taking longer to process, the other clients will have an increase in their times.

# Conclusion

The data shows that the concurrent server can candle far many more clients than the iterative server, with only small changes needed to turn the server from iterative to concurrent. In conclusion, the concurrent server is not only faster for smaller amounts of clients, it drastically increases the number of clients that can be getting requests. This allows for any services being handled by the server to be faster and thus the clients are happier. If the question was ever to make an iterative server or a concurrent server, it is worth the bit of extra code to get faster server responses.

# Lessons Learned

Many lessons were learned in this project. The previous experience gained from multithreading the client generator made the server easy to change. However, the result gotten from the data was eye opening in how much faster a concurrent server really was. Another lesson learned was how important it is to keep code clean and well documented for future use. When going back to edit our code many times we had to completely reread about certain concepts to refamiliarize ourselves with what our code was doing. This could have been reduced or even negated if the code had better documentation and was easier to read. Overall we learned many things from these two projects.