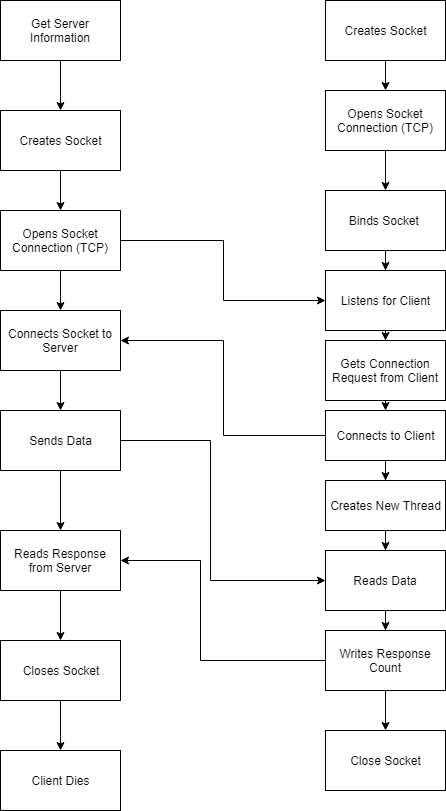
# Documentation:

This assignment is intended to be a multithreaded client server that will evaluate the reads and writes done by the system. The overall structure of the assignment is broken down into two file, Client.cpp and Server.cpp.

|  |  |
| --- | --- |
| The Server does the following:   1. Creates a TCP socket 2. Listens for connection request from the client 3. Creates a new thread 4. Connects to the client’s socket 5. Creates a new Thread 6. Reads Data from the Client 7. Writes an Acknowledgement to Client 8. Closes Socket | The Client does the following:   1. Gets Server Information 2. Creates new socket 3. Connect to Server 4. Send Data    1. Writes each buffer    2. Stores in an array and send entire thing    3. Send all data buffers at once 5. Waits for Acknowledgement 6. Closes Socket |

The Server file is primarily made up of four logical sections. The first is the main function that will take in the required number of arguments and convert them into integers. Should the wrong type be sent in, a segmentation fault will occur. After the conversions, it will pass these variables into the server constructor. In this constructor, the variables will be tested to ensure that they are not out of range or faulty. It will then be returned to the main method where it will run the server’s run method. This method will set up the sockets needed for connecting to clients and creating new thread to read the data. When one of these threads are created, the final method will run. In this method, it will read each byte of data from the client and count each read. After it is done, it will compute how long it took to receive and send the data and display it to the console. In addition, it will send an acknowledgement to the client. It will send a count of the number of reads that the server did on the client’s data.

The Client file is primarily made up of three logical sections. Like the server file, the client’s first method is the main method where it will gather the variables and ensure there are the correct amount. In addition it will convert all variables to integers, except the serverIP, which will remains as a string/char array. Once in the constructor, the variables will undergo a error checking process and will throw invalid\_argument errors if they are not correct. If all checks pass, it will initialize the global variables. Finally, it will return to the main method and run the client’s run method. In this method sockets will be created and connected to the server. After this, it will write the data to the network based on the type transfer selected. There are three types of transfers. The first is a buffer by buffer write that writes individual sections at a time. The second will store the buffers in an array and send the entire array on the network. The final mode will writeall buffers to the network at the same time. After this is done, it will compute the transfer and round trip times and print them to the console. In addition, it will print the acknowledgement from the server to the console.

**For a more detail explanation, see comments in code**

## Compiling and Running Files

**Client:**

Compile command: g++ Client.cpp -o client

run Command: ./client port repetition nbuf bufsize

example: ./client 1522 20000 15 100 127.0.0.1 1

**Server:**

Compile command: g++ Server.cpp -o server -pthreads

run Command: ./server port repetition

example: ./server 1522 20000

# Execution Output

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 1

Test1: data-sending time =:243571 usec Round-trip time = 257226 usec #reads = 21027

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 1

Test1: data-sending time =:241150 usec Round-trip time = 257379 usec #reads = 22368

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 1

Test1: data-sending time =:243309 usec Round-trip time = 257434 usec #reads = 22600

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 2

Test2: data-sending time =:242586 usec Round-trip time = 261222 usec #reads = 21039

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 2

Test2: data-sending time =:241810 usec Round-trip time = 261107 usec #reads = 21035

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 2

Test2: data-sending time =:247273 usec Round-trip time = 261269 usec #reads = 21036

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 3

Test3: data-sending time =:244411 usec Round-trip time = 260849 usec #reads = 21037

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 3

Test3: data-sending time =:240381 usec Round-trip time = 257485 usec #reads = 21025

paduac@uw1-320-07:~/CSS 432$ ./client 1522 20000 15 100 uw1-320-05.uwb.edu 3

Test3: data-sending time =:243753 usec Round-trip time = 259647 usec #reads = 21033

data-receiving time = 256853 usec

data-receiving time = 256759 usec

data-receiving time = 257165 usec

data-receiving time = 260727 usec

data-receiving time = 260597 usec

data-receiving time = 260887 usec

data-receiving time = 260306 usec

data-receiving time = 256933 usec

data-receiving time = 259227 usec

**This is sample output from the Client and Server for nbuf =15, bufsize =100, and types 1-3. For full terminal results, see “Results from Terminal.docx”.**

# Performance evaluation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| nbufs | bufsize | Type | data-sending time | round trip time | reads | data receiving time |  |
| 15 | 100 | 1 | 243571 | 257226 | 21027 | 256853 | **Port = 1522** |
| 15 | 100 | 1 | 241150 | 257379 | 22368 | 256759 | **Repetition =20000** |
| 15 | 100 | 1 | 243309 | 257434 | 22600 | 257165 |  |
| 15 | 100 | 2 | 242586 | 261222 | 21039 | 260727 |  |
| 15 | 100 | 2 | 241810 | 261107 | 21035 | 260597 |  |
| 15 | 100 | 2 | 247273 | 261269 | 21036 | 260887 |  |
| 15 | 100 | 3 | 244411 | 260849 | 21037 | 260306 |  |
| 15 | 100 | 3 | 240381 | 257485 | 21025 | 256933 |  |
| 15 | 100 | 3 | 243753 | 259647 | 21033 | 259227 |  |
| 30 | 50 | 1 | 247784 | 262816 | 21221 | 262462 |  |
| 30 | 50 | 1 | 254428 | 269554 | 22368 | 269204 |  |
| 30 | 50 | 1 | 250706 | 268526 | 21735 | 268164 |  |
| 30 | 50 | 2 | 238299 | 257687 | 21169 | 257339 |  |
| 30 | 50 | 2 | 241080 | 258150 | 21031 | 257835 |  |
| 30 | 50 | 2 | 243962 | 260386 | 21035 | 260051 |  |
| 30 | 50 | 3 | 243272 | 260563 | 21037 | 260030 |  |
| 30 | 50 | 3 | 241560 | 257434 | 21022 | 256932 |  |
| 30 | 50 | 3 | 242200 | 261064 | 21036 | 260550 |  |
| 60 | 25 | 1 | 370590 | 371730 | 23830 | 371214 |  |
| 60 | 25 | 1 | 363201 | 364062 | 23376 | 363500 |  |
| 60 | 25 | 1 | 357904 | 359042 | 23575 | 358491 |  |
| 60 | 25 | 2 | 242950 | 257409 | 21895 | 257005 |  |
| 60 | 25 | 2 | 241320 | 257913 | 21027 | 257596 |  |
| 60 | 25 | 2 | 240945 | 257466 | 22559 | 256959 |  |
| 60 | 25 | 3 | 241206 | 259884 | 21031 | 259329 |  |
| 60 | 25 | 3 | 244662 | 258328 | 21025 | 257884 |  |
| 60 | 25 | 3 | 242553 | 257618 | 21936 | 257076 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Averages |  |  |  |
| BufferSize | Data Sending | Round Trip | Reads |
| Typ 1 |  |  |  |
| 100 | 242676.6667 | 257346.3333 | 21998.33 |
| 50 | 250972.6667 | 266965.3333 | 21774.67 |
| 25 | 363898.3333 | 364944.6667 | 23593.67 |
|  |  |  |  |
| Type 2 |  |  |  |
| 100 | 243889.6667 | 261199.3333 | 21036.67 |
| 50 | 241113.6667 | 258741 | 21078.33 |
| 25 | 241738.3333 | 257596 | 21827 |
|  |  |  |  |
| Type 3 |  |  |  |
| 100 | 242848.3333 | 259327 | 21031.67 |
| 50 | 242344 | 259687 | 21031.67 |
| 25 | 242807 | 258610 | 21330.67 |

Type 2

Type 3

Type 1

Taking the data as a whole, the data collected from the client server seems to be inconclusive. But when you break the data down into the different types of data transfers, a pattern starts to emerge among the subgroups. This is especially apparent with the Type 1 data transfer. As the data buffer increases, the time it takes for the round-trip decreases. However, for data types 2 and 3, there was not much common pattern among the data. The only thing noticeable is that the average transfer times for type 2 and 3 are slightly faster than the averages for type 1.

# Discussion

## Multi-Writes

In my tests, the most differences were discovered when the buffer size decreased and the total transfer time increased. This was because of the consistent starting and stopping of writes to the network and the transmission of data. If the network speed was to decrease, this would increase the overhead of the transmission and could introduce queuing delays or packet loss. In addition, throughput would decrease because less data can get across the network and to the server.

## Writev

If the network was slowed down for writev, not much would change besides transmission time. This is because all the data is being transmitted as one whole. One thing that would change is the latency that the information arrives at the destination/server.

## Singlewrite

Much like writev, singlewrite will not change much because of the data being sent all at once. Like writev, it will suffer from latency issue.

## Other Considerations

One thing to keep in mind for all three types is the idea of error rate. Even if the error rate is small, it can have detrimental affects on performance. This is because if the data that arrives at the server is corrupted, they are going to have to re-request the information. By doing this, we have gone one full cycle between the client and server and will have to have for another half a cycle re-receive the data on the slow network.

One reason that you would want to service the connection in a thread is because this allows for you to accept multiple connections at the same time. If the main is doing all the work, the next connection would have to wait until the previous one finished. This would introduce a variety of issues with throughput, queuing delays, processing delays, and packet loss. In addition, using multiple threads will allow you to do one of two things. Spilt data retrieval into multiple parts, or allow for multiple clients to contact one server.