**Program1 Report**

Aika Usui

# Implementation

Client.cpp Server.cpp

A close up of a map

Description automatically generatedA close up of a logo

Description automatically generated

## Client.cpp

Client takes six arguments. argv[1] = serverName, argv[2] = IP port number used by server (last 5 of student ID), argv[3] = Repetition of sending a set of data buffers, argv[4] = Number of data buffers, argv[5] = Size of each data buffer (in bytes) and argv[6] = type of transfer scenario: 1, 2, or 3. The program verifies the arguments first and if any of them are invalid values, it exits the program. Establish a connection to a server using getaddrinfo() and socket(). Once a connection is success, it will send the repetition value by using write(). After that, it performs the corresponded test for appropriate number of iterations. During this process, the program measures the time it takes to send by using chrono time library. A client receive a message from a server by using read(). Finally, the program prints out the information about the test in the format of “Test (1,2, or 3): time = xx usec, #reads = yy, throughput zz Gbps” and close the socket.

## Server.cpp

Server take one argument, port number which is store in argv[1]. The program verifies the input and if it include invalid values, it exits the program. Creates a socket for a server and binds the socket to the local address. The socket listens to up to 5 connections and it will accept to establish a connection and get a new socket descriptor. After that, main calls the servicing thread to read the message from the client and write back to the client.

# Performance Evaluation

Performed 10 times for each for four different combination of nbus\*bufsize on each type of test and calculates the average for each.

**Repetition = 20000, nbufs \* bufsize** = 15 \* 100

|  |  |  |  |
| --- | --- | --- | --- |
| Type | RTT (usec) | #Reads | Throughput (Gbps) |
| 1 | 275342 | 22906.6 | 0.8250208 |
| 2 | 45541.8 | 20249.6 | 5.015952 |
| 3 | 43425.2 | 20158 | 5.885052 |

**Repetition = 20000, nbufs \* bufsize** = 30 \* 50

|  |  |  |  |
| --- | --- | --- | --- |
| Type | RTT (usec) | #Reads | Throughput (Gbps) |
| 1 | 586141.6 | 24621.2 | 0.2979302 |
| 2 | 71428 | 25735.2 | 3.311844 |
| 3 | 55830.2 | 24379 | 5.855086 |

**Repetition = 20000, nbufs \* bufsize** = 60 \* 25

|  |  |  |  |
| --- | --- | --- | --- |
| Type | RTT (usec) | #Reads | Throughput (Gbps) |
| 1 | 852264.6 | 29437.6 | 0.2300648 |
| 2 | 128474 | 22207.4 | 2.082242 |
| 3 | 43875.4 | 20193.8 | 6.524414 |

**Repetition = 20000, nbufs \* bufsize** = 100 \* 15

|  |  |  |  |
| --- | --- | --- | --- |
| Type | RTT (usec) | #Reads | Throughput (Gbps) |
| 1 | 1743822 | 34322.4 | 0.1321038 |
| 2 | 178868.8 | 22973.2 | 1.337104 |
| 3 | 41643.8 | 20156.2 | 5.218578 |

# Discussion

### comparing your actual throughputs to the underlying bandwidth

Since I could not find the bandwidth of the Linux lab, I decided to use the average bandwidth value in the United States. According to speedtest.net, the average bandwidth in the United States is 132.55 Mbps, which is 0.13255 Gbps. Every throughput that I got from the performance test are higher than the bandwidth. Throughput of Type 1 scenario is close to the bandwidth value although it took the longest time to process. Again, since I could not get the actual bandwidth value of the Linux lab, I can’t conclude that throughput can stay close to bandwidth even in the worst scenario, but as far as I tested, it is possible.

References: <https://www.speedtest.net/global-index>

### comparisons of the performance of multi-writes, writev, and single-write performance

* Number of reads for single-write is lowest for any nbuf and bfsize values.
* Single-write is the fastest regarding to RTT and throughput is the highest of all.
* Multi-writes is the slowest of all regarding to RTT and throughput is the lowest of all.

Approximately, the more it performs the reading, it gets slower for the RTT based on the relationship between #Reads and RTT for each type of performance. Reference to the code, the multiple writes requires a for loop to write all the data, which indicates that the server performs the same amount of reading as the iteration of loop. On the other hand, single write writes only once and the server reads once. Similar to this, type 2 also requires to write only once. This gives me the idea that it is faster to perform the one read with big data than multiple reads with small data. The bigger the bufsize is, the higher the throughput is. This makes sense because the more data is processed if the bufsize is bigger.

### comparison of the different buffer size / number buffers combinations

* As nbufs increases, the RTT increases for type 1 and 2.
* As nbufs increases, the number of reads for type 1 increases.
* Throughput is highest for 15\*100 for type 1, 2, and 3.

As I discussed above, multi-writes has a for loop to go over all the data to be written. Since nbufs decides how many iterations will performs, the number of reads for type 1 increases as nbufs increases. Similarly, since type 2 required a for loop that is based on nbufs, its RTT increases as the nbufs increases. However, type 2 is still faster than type 1 because type 2 only calls one write call. Type 3 don’t show the significant changes over the different values for buffer size and number buffers because it only calls one write call and the process is not affected by the nbufs or bufsize.