Program 1 Report

# Design

The design of this program follows the implementation of a simple TCP client-server architecture.

Once the server instance starts up, it begins waiting for a client to connect. Upon connection, a new thread will be created which will read the data sent by the client and keep track of the number of bytes read. Once it has finished reading the data, the server returns the number of bytes read to the client.

When a client instance starts up, it attempts to connect to the server. If the client was successful in connecting to the server, it beings writing data to the server. There are three ways for the client to write to the server: multiple writes, writev, and single write.

# Results

Table 1: Server Receiving Times (in microseconds)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Scenario 1 (multiple writes)** | **Scenario 2 (writev)** | **Scenario 3 (single write)** |
| 15 nbufs \* 100 bufsize | 256991usec | 256721usec | 256955usec |
| 30 nbufs \* 50 bufsize | 257432usec | 257136usec | 256849usec |
| 60 nbufs \* 25 bufsize | 280714usec | 257100usec | 256830usec |

Table 2: Client Data Sending Times (in microseconds)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Scenario 1 (multiple writes)** | **Scenario 2 (writev)** | **Scenario 3 (single write)** |
| 15 nbufs \* 100 bufsize | 240119usec | 241399usec | 239894usec |
| 30 nbufs \* 50 bufsize | 242624usec | 244359usec | 241700usec |
| 60 nbufs \* 25 bufsize | 280656usec | 241431usec | 238526usec |

Table 3: Client Round Trip Times (in microseconds)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Scenario 1 (multiple writes)** | **Scenario 2 (writev)** | **Scenario 3 (single write)** |
| 15 nbufs \* 100 bufsize | 257825usec | 257486usec | 257628usec |
| 30 nbufs \* 50 bufsize | 258278usec | 257879usec | 257639usec |
| 60 nbufs \* 25 bufsize | 281618usec | 257785usec | 257641usec |

# Performance Evaluation

When looking at the performance for how data is split when sending, we see faster times in all scenarios when the buffer is split into smaller numbers and bigger buffer size (15 nbufs \* 100 bufsize). As the buffer size gets smaller and the number of bytes increase, we see that the server receiving times, the data sending times, and round-trip times are slower.

When we look at the different scenarios of sending data, we find that on average, single writes are faster than multiple writes and writev. This is true for all the different ways the buffer is split up.

# Discussion

It makes sense for a single write to be faster than multiple writes because when calling a write, it makes a system call. So, when we have to write for each data buffer, the times add up to be faster than writing using a single write. An advantage writev has over a single write is that the data buffer can be split. This can be useful for a slower connection where there is a higher potential for a packet loss if sending a large buffer rather than splitting it up. However, if connection speeds are normal, it would be better to send the data buffer over a single write.

When a server creates a thread to handle incoming connections, it allows for the server to handle the data while also allowing other connections to come in. This would not be possible if done in the main method since the server would have to process the whole buffer before being able to accommodate other incoming connections.