Intro to Network Programming Documentation

Client.cpp

The client takes in 6 parameter arguments in the following order: [port] [repetition] [nbufs] [bufsize] [serverIp] [type]. Note: the nbufs and bufsize are multiplied and must be equal to 1500. Start the timer called “start” to begin the operation of writing and reading.

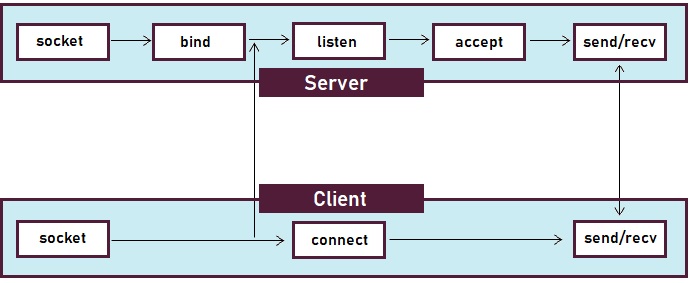
It is tested in 3 scenarios:

1. Multiple writes: invokes the write( ) system call for each data buffer, thus resulting in calling as many write( )s as the number of data buffers, (i.e., nbufs).
2. Writev: allocates an array of iovec data structures, each having its \*iov\_base field point to a different data buffer as well as storing the buffer size in its iov\_len field; and thereafter calls writev( ) to send all data buffers at once. It means that it will call as many writes for each data buffers.
3. Single write: allocates an nbufs-sized array of data buffers, and thereafter calls write( ) to send this array, (i.e., all data buffers) at once.

After this writing operation, get the time for lap to show the transfer of the data to server

Then, you have an end timer to indicate that all writing and reading operations were completed.

Illustration of the interactions of the server and client: Send/Recv are being replaced by write/read respectively. The operations of writing are being done in a pthread function called “func” and sent back to client.



Server.cpp – must be set up first before calling client to take in

Server takes in two parameter arguments in the following order: [port] [repetition] and that the port must be greater than 100 and must be less than the maximum port number, which is 65535. The repetition must also be same as the client’s repetition, lest it break the server.

It has a thread function routine that would be executed when the main() call the creation of the pthread, which it would pass in a struct that would contain the client’s socket and repetition from the server. The thread function routine would have a for loop that would begin reading all of the data (buffer) from the client’s sockets. Once it has finished reading all of the data, then it sends(writes) it back to the client, which was being tracked by an int called “count.” Then, it would calculate the data receiving time. It would return back to the main() while loop, which is infinite, and calls a pthread\_join to avoid wasting resources and server goes onto standby until it receives another client. Every time client is being called, it would create a new data and merge it back with thread\_join().