ECGR 3123: Data Communication and Networking

Project 1: Client-Server Chat

By: Kevin Granados, Wallace Obey

Objective:

The objective of Project 1 is to design and implement a communication system. The required deliverable is to have a functioning single client to single server communication system. If this was achieved, then a client to server to client, and even the backend of a group chat application (multiple clients talking to each other through one server in real time) could be achieved for more points.

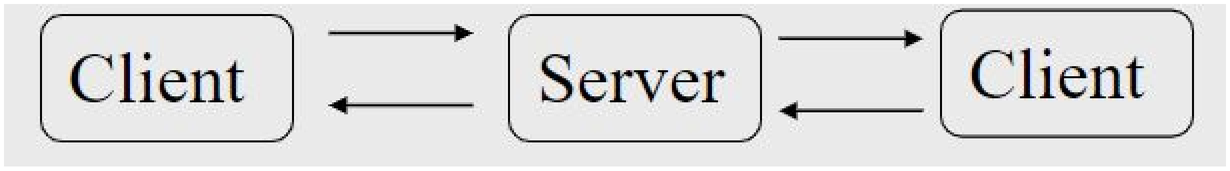
Design Specifications:

Client Server Communication:
- Design should be done in C++(Windows, Linux, Mac) using socket programming
- Build the client and server structs from scratch make them communicate continuously.
- The sockets should be one TCP sockets and one UDP socket.

Additional Points: Client to Server to Client Communication
- Have to get clients to talking to each by client to server the client communication
- Real time constraint (requires the use of multithreading)

Additional Points: Group Chat functionality
- Have multiple clients talking to each other through a single server
- Extensive Real time constraints (requires the use of multithreading)

Client to Server:



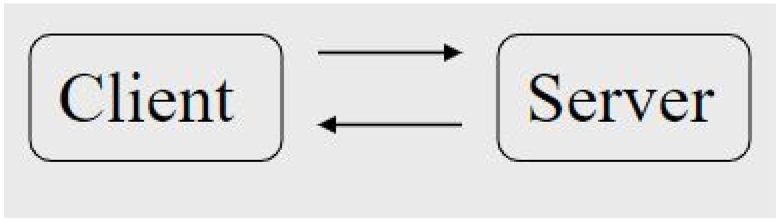


Figure 1: Client to Server Communication [1]

Client to Server to Client:

Figure 2: Client to Server to Client Communication [1]

Group Chat:

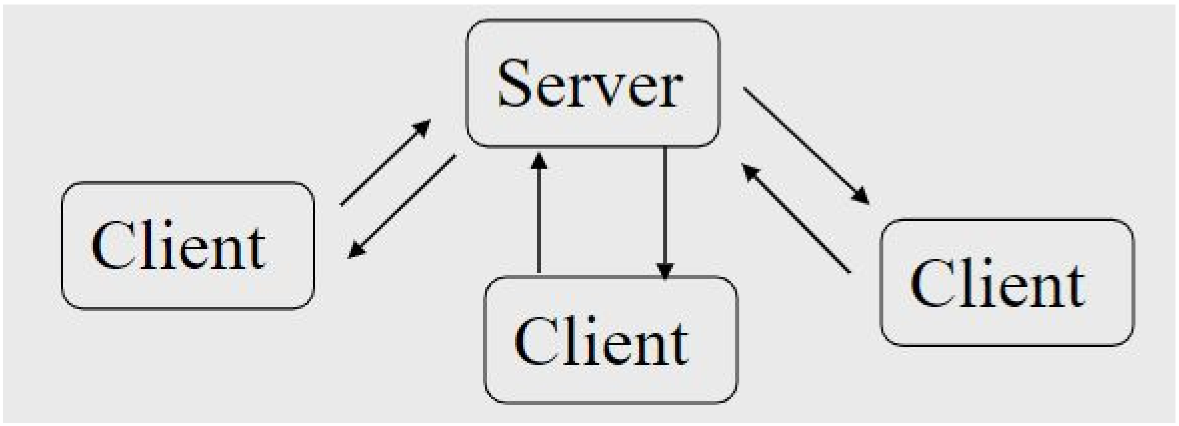
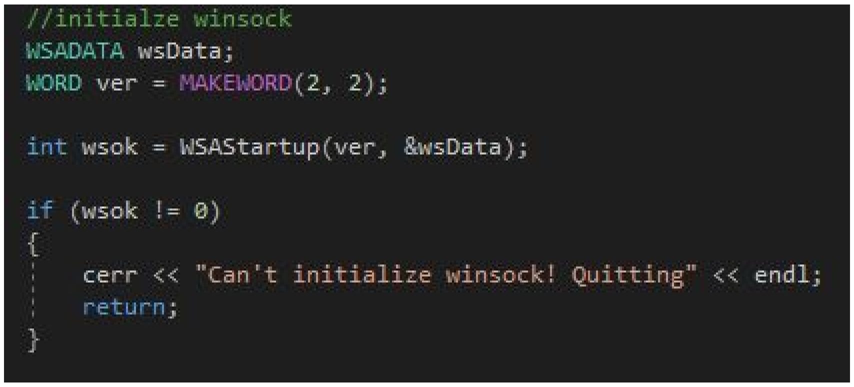


Figure 3: Group Chat Communication [1]

Code Explanation:

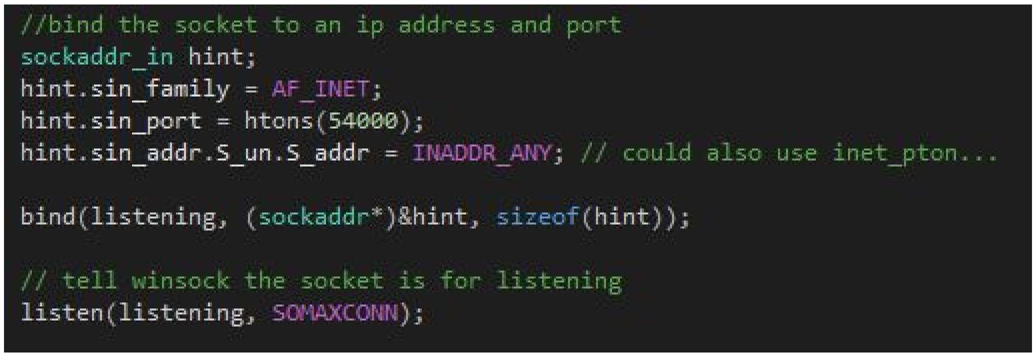
Client Server Communication:

- Server code: Winsock is a programming interface and the supporting program that handles input/output requests for Internet applications in a Windows OS in order to use this service in our code the "ws2_32.lib" and “ws2tcpip.h” libraries were imported. Firstly we initialize winsock and check if winsock starts up okay, with the WSAStartup function, which returns an integer value which we checked in our first if statement to output a error that winsock could not be initialized.



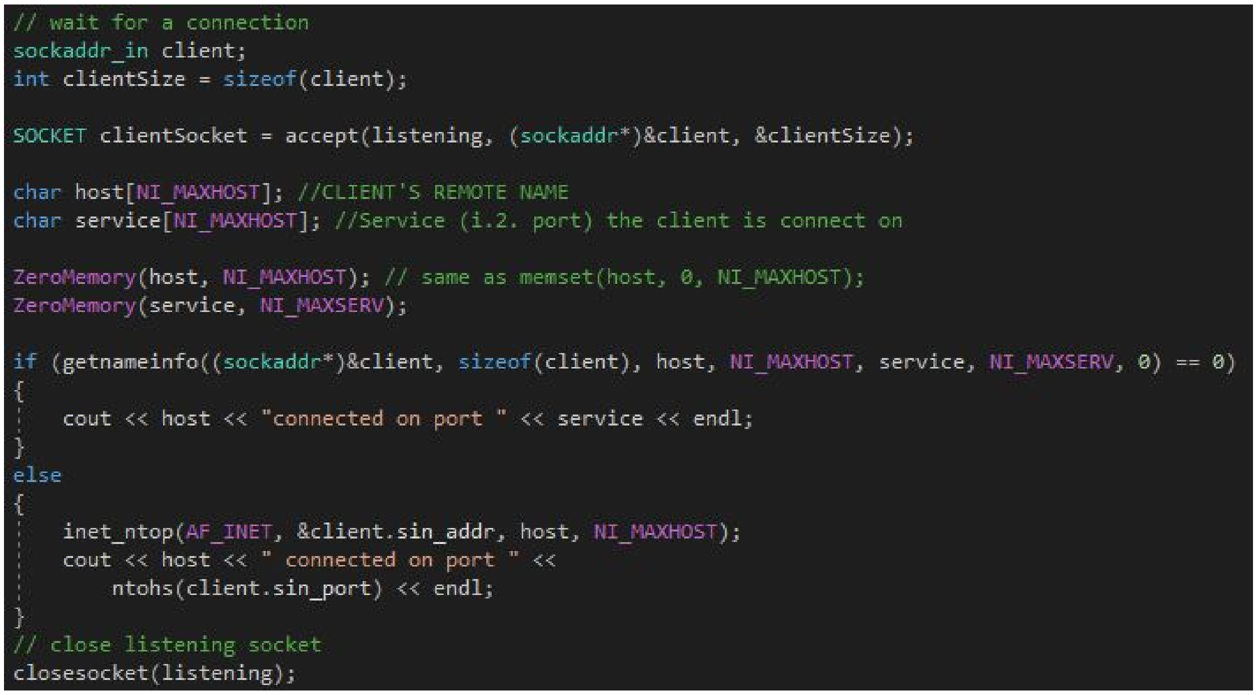
```
//initialze winsock
WSADATA wsData;
WORD ver = MAKEWORD(2, 2);

int wsok = WSAStartup(ver, &wsData);

if (wsok != 0)
{
    cerr << "Can't initialize winsock! Quitting" << endl;
    return;
}
```

Next step was to create the socket or endpoint for our server and tell winsock the socket is for listening as well as bind the socket to an IP address and port to 54000.



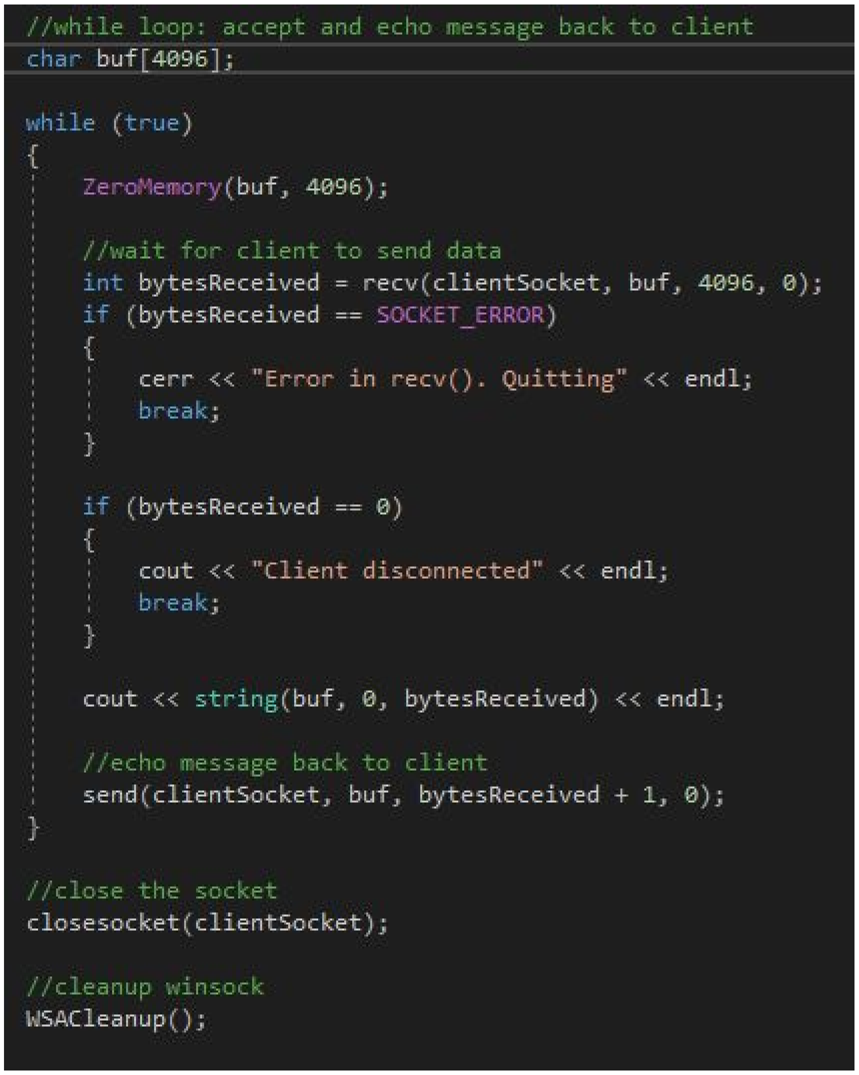
```
//bind the socket to an ip address and port
sockaddr_in hint;
hint.sin_family = AF_INET;
hint.sin_port = htons(54000);
hint.sin_addr.S_un.S_addr = INADDR_ANY; // could also use inet_pton...

bind(listening, (sockaddr*)&hint, sizeof(hint));

// tell winsock the socket is for listening
listen(listening, SOMAXCONN);
```

Now, wait for a connection and make an event handler to get info of of who connected.



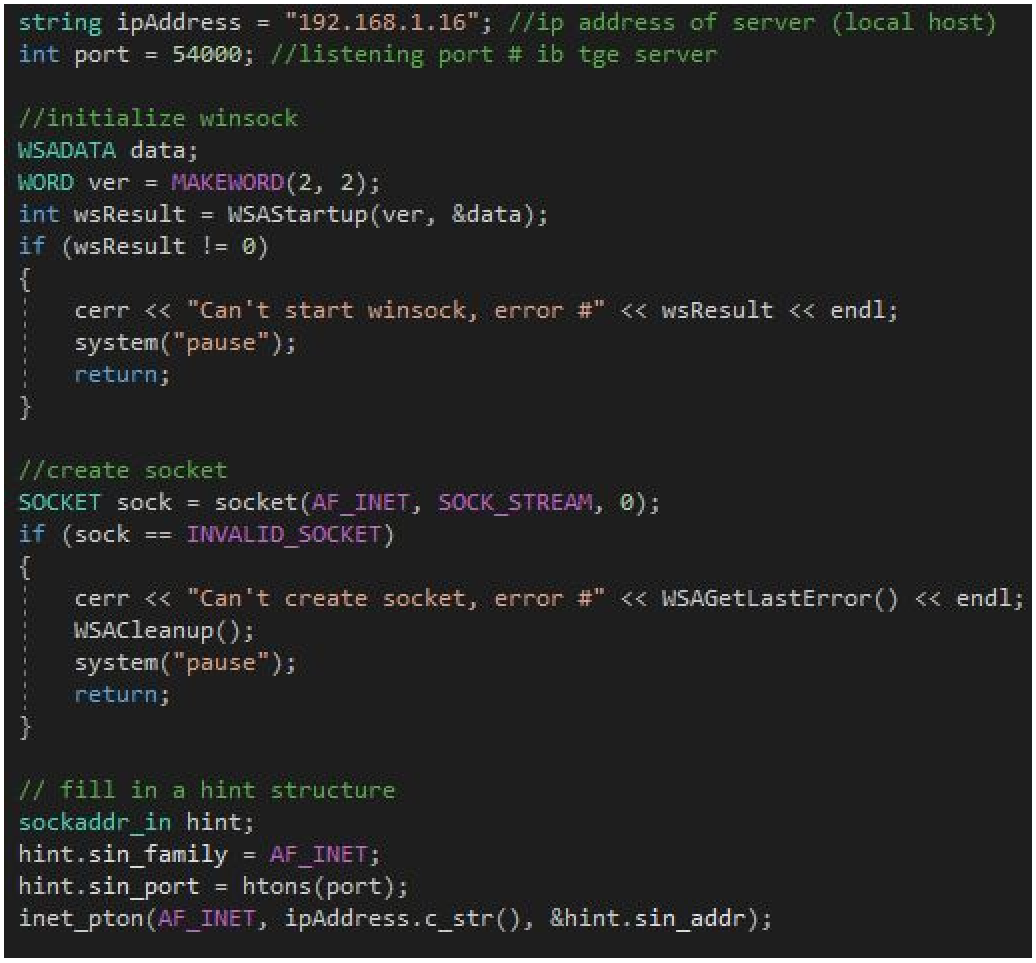
```
// wait for a connection
sockaddr_in client;
int clientSize = sizeof(client);

SOCKET clientSocket = accept(listening, (sockaddr*)&client, &clientSize);

char host[NI_MAXHOST]; //CLIENT'S REMOTE NAME
char service[NI_MAXHOST]; //Service (i.2. port) the client is connect on

ZeroMemory(host, NI_MAXHOST); // same as memset(host, 0, NI_MAXHOST);
ZeroMemory(service, NI_MAXSERV);

if (getnameinfo((sockaddr*)&client, sizeof(client), host, NI_MAXHOST, service, NI_MAXSERV, 0) == 0)
{
    cout << host << "connected on port " << service << endl;
}
else
{
    inet_ntop(AF_INET, &client.sin_addr, host, NI_MAXHOST);
    cout << host << " connected on port " <<
        ntohs(client.sin_port) << endl;
}
// close listening socket
closesocket(listening);
```

Once connected wait for a message from the client using “recv()” function and send the message by using the “send()” function.


```
//while loop: accept and echo message back to client
char buf[4096];

while (true)
{
    ZeroMemory(buf, 4096);

    //wait for client to send data
    int bytesReceived = recv(clientSocket, buf, 4096, 0);
    if (bytesReceived == SOCKET_ERROR)
    {
        cerr << "Error in recv(). Quitting" << endl;
        break;
    }

    if (bytesReceived == 0)
    {
        cout << "Client disconnected" << endl;
        break;
    }

    cout << string(buf, 0, bytesReceived) << endl;

    //echo message back to client
    send(clientSocket, buf, bytesReceived + 1, 0);
}

//close the socket
closesocket(clientSocket);

//cleanup winsock
WSACleanup();
```

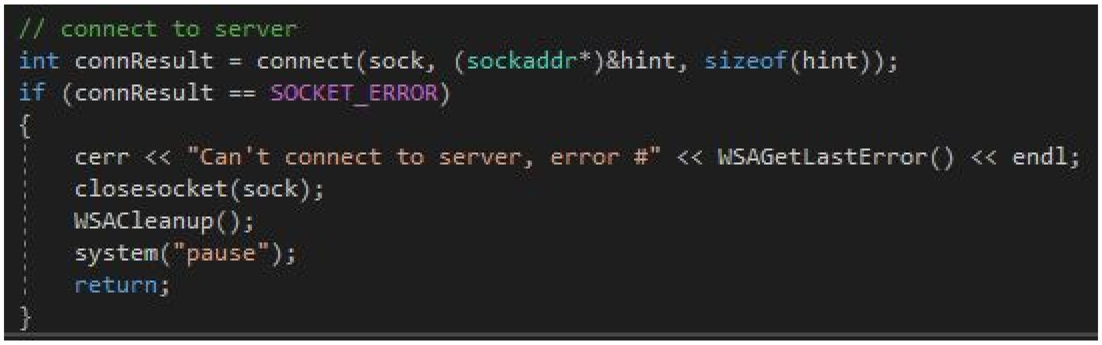
Lastly we close the connection of the socket using the “closesocket()” function and cleanup the winsock using the “WSACleanup” to close our server program.

- Client code: The IP address (local IP address) is set to a variable string and the port is set to 54000 to connect to the server. Then similar to the server code we have to initialize winsock and create a socket.



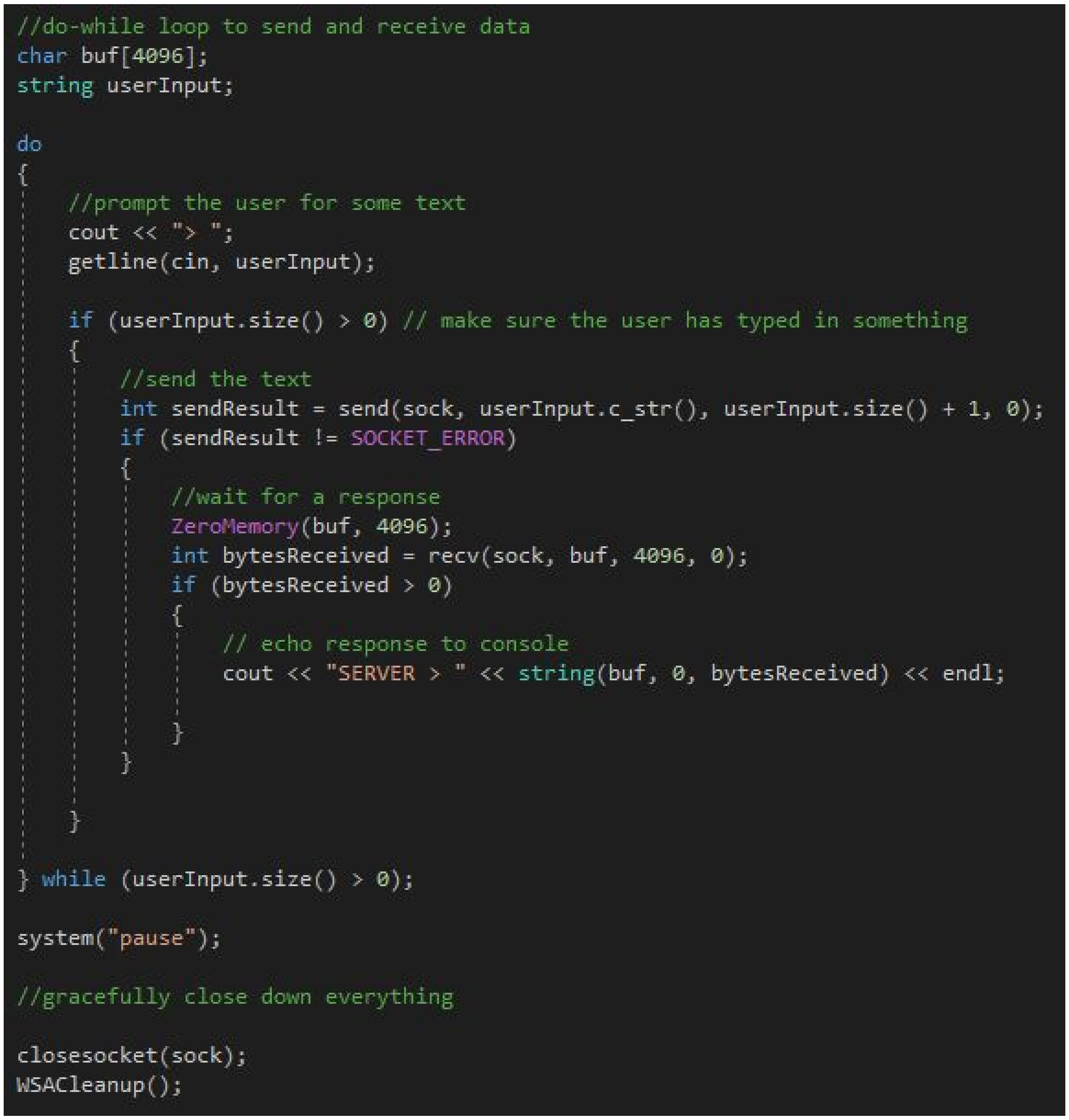
```
string ipAddress = "192.168.1.16"; //ip address of server (local host)
int port = 54000; //listening port # ib tge server

//initialize winsock
WSADATA data;
WORD ver = MAKEWORD(2, 2);
int wsResult = WSAStartup(ver, &data);
if (wsResult != 0)
{
    cerr << "Can't start winsock, error #" << wsResult << endl;
    system("pause");
    return;
}

//create socket
SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock == INVALID_SOCKET)
{
    cerr << "Can't create socket, error #" << WSAGetLastError() << endl;
    WSACleanup();
    system("pause");
    return;
}

// fill in a hint structure
sockaddr_in hint;
hint.sin_family = AF_INET;
hint.sin_port = htons(port);
inet_pton(AF_INET, ipAddress.c_str(), &hint.sin_addr);
```

Now, connect to the server using the “connect ()” function



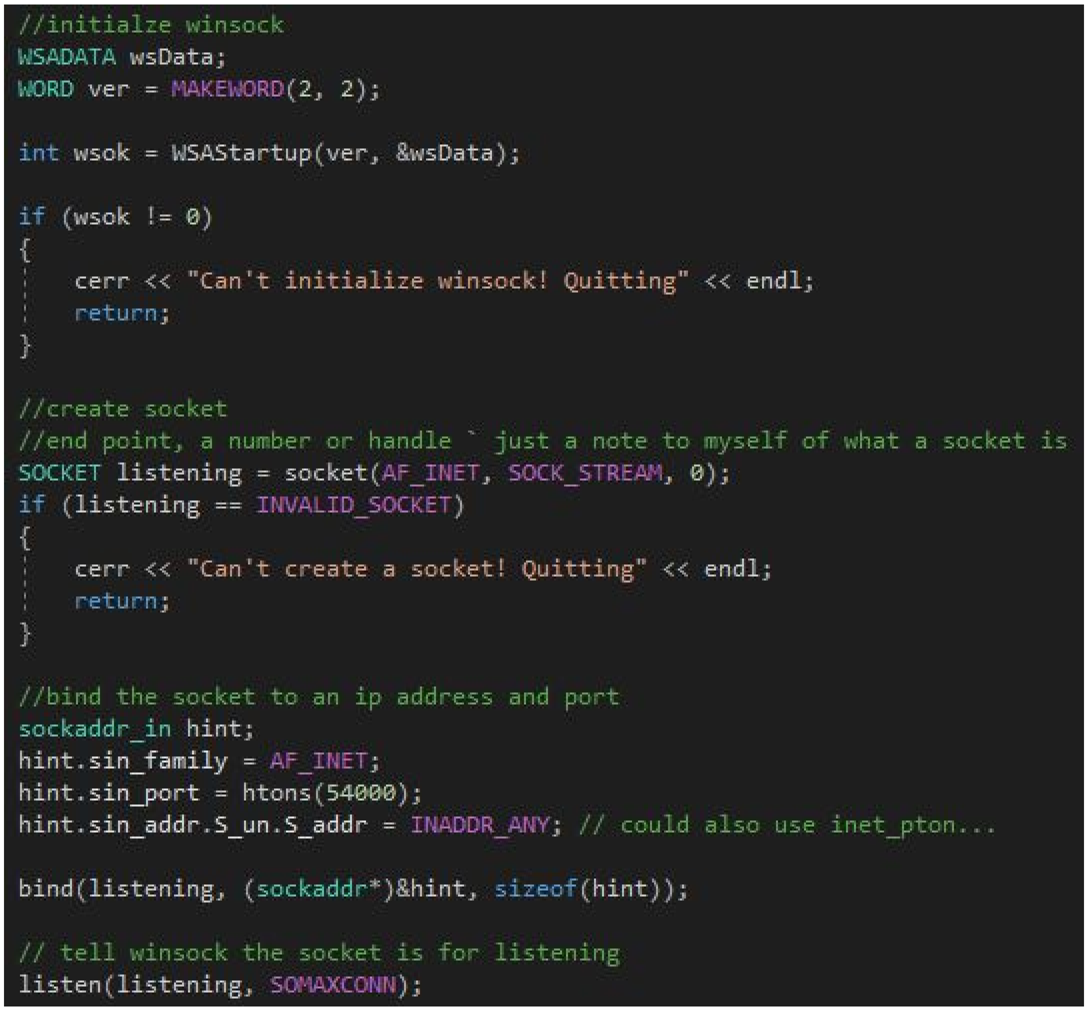
```
// connect to server
int connResult = connect(sock, (sockaddr*)&hint, sizeof(hint));
if (connResult == SOCKET_ERROR)
{
    cerr << "Can't connect to server, error #" << WSAGetLastError() << endl;
    closesocket(sock);
    WSACleanup();
    system("pause");
    return;
}
```

Next start a do-while loop to send and receive data. Included in our do-while loop is to prompt the user for some text and wait until the input is entered using the “getline()” function. Once the user has entered something the “send()” function is used to send to the server. Once the server send the message back to the client the if statement for the variable bytes Received is greater than 0 the server echo is printed out on the console. The do while loop will continue to run and echo the what the user has sent until the user input is less than 0. Then similar to the server code the use of “closesocket” and “WSACleanup()” functions to close our Client code.


```
//do-while loop to send and receive data
char buf[4096];
string userInput;

do
{
    //prompt the user for some text
    cout << "> ";
    getline(cin, userInput);

    if (userInput.size() > 0) // make sure the user has typed in something
    {
        //send the text
        int sendResult = send(sock, userInput.c_str(), userInput.size() + 1, 0);
        if (sendResult != SOCKET_ERROR)
        {
            //wait for a response
            ZeroMemory(buf, 4096);
            int bytesReceived = recv(sock, buf, 4096, 0);
            if (bytesReceived > 0)
            {
                // echo response to console
                cout << "SERVER > " << string(buf, 0, bytesReceived) << endl;

            }
        }

    }

} while (userInput.size() > 0);

system("pause");

//gracefully close down everything

closesocket(sock);
WSACleanup();
```

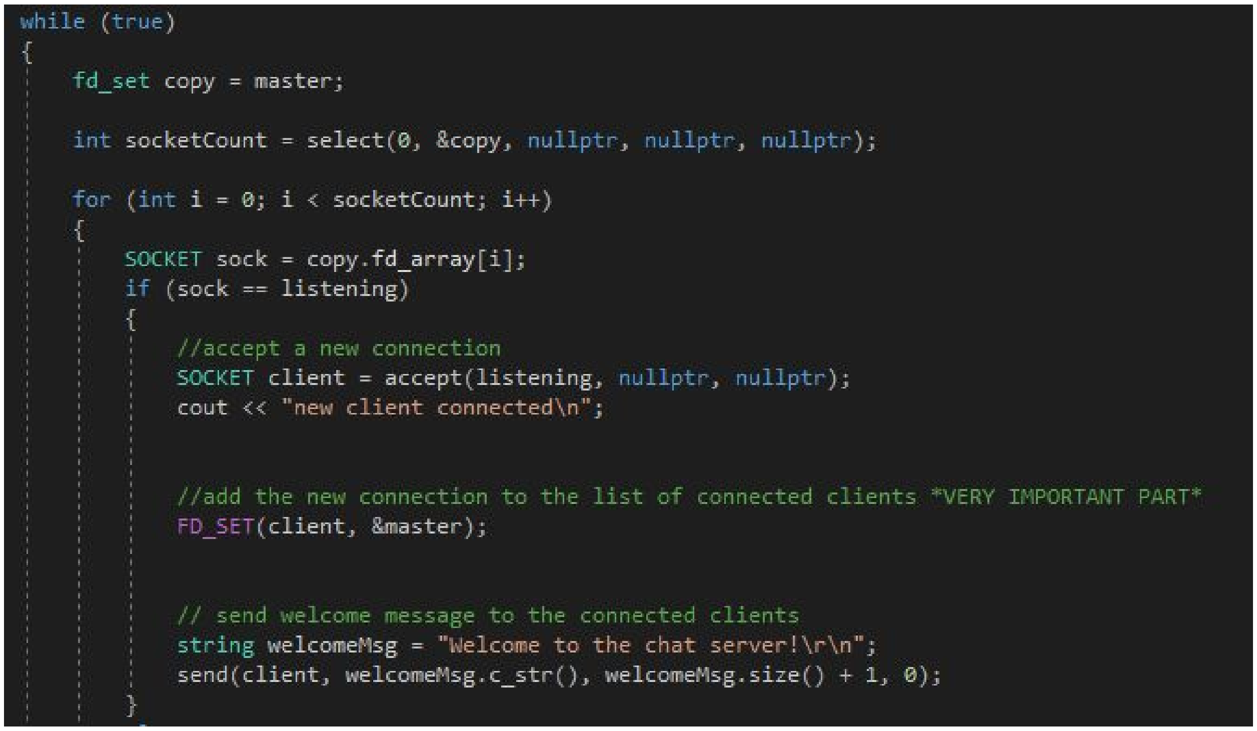
Multi-Client Communication:

- Server code: The "ws2_32.lib" and “ws2tcpip.h” libraries were imported in order to use Winsock. Similar to the single client server code initialization of winsock and creation of socket and binding of the socket to the IP address and port are created then tell winsock to listen using the “listen()” function.



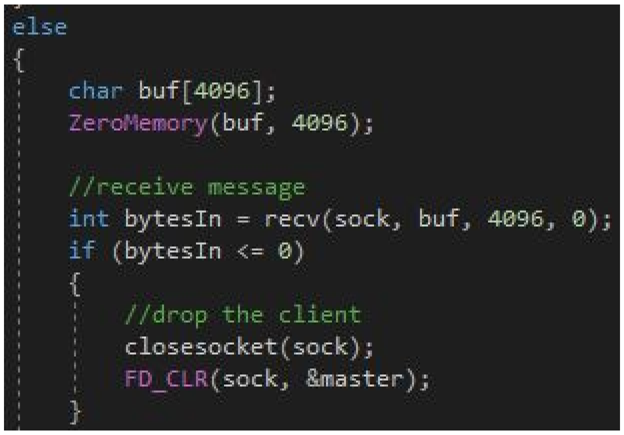
```
//initialze winsock
WSADATA wsData;
WORD ver = MAKEWORD(2, 2);

int wsok = WSAStartup(ver, &wsData);

if (wsok != 0)
{
    cerr << "Can't initialize winsock! Quitting" << endl;
    return;
}

//create socket
//end point, a number or handle ` just a note to myself of what a socket is
SOCKET listening = socket(AF_INET, SOCK_STREAM, 0);
if (listening == INVALID_SOCKET)
{
    cerr << "Can't create a socket! Quitting" << endl;
    return;
}

//bind the socket to an ip address and port
sockaddr_in hint;
hint.sin_family = AF_INET;
hint.sin_port = htons(54000);
hint.sin_addr.S_un.S_addr = INADDR_ANY; // could also use inet_pton...

bind(listening, (sockaddr*)&hint, sizeof(hint));

// tell winsock the socket is for listening
listen(listening, SOMAXCONN);
```

The fd_set structure is used by various Windows Sockets functions and service providers, such as the select function, to place sockets into a "set" for various purposes, such as testing a given socket for readability using the readfds parameter of the select function. A while loop is created to continue to check for new connections. If the number of sockets are less then “i” which is keeping count of our client users then in our case we use the fd_set functions to check the socket count and keep track of how many clients have been connected and store them into an array. If the socket is equal to the listening socket then the connection is accepted by the server and a new client is added to the array and as well as a welcome message is sent to the connected user.



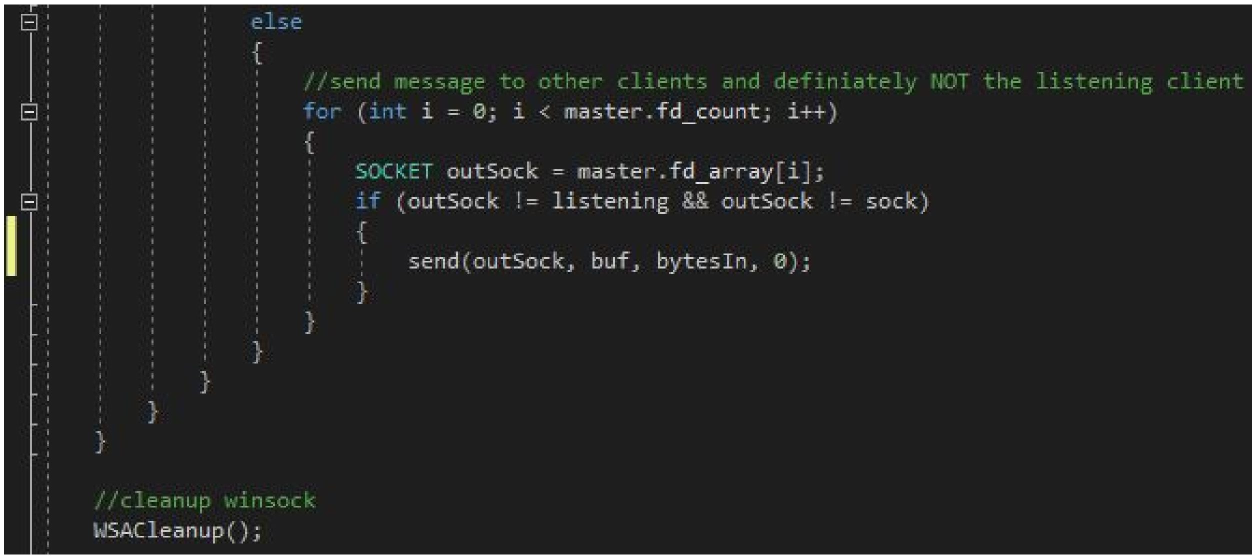
```
while (true)
{
    fd_set copy = master;

    int socketCount = select(0, &copy, nullptr, nullptr, nullptr);

    for (int i = 0; i < socketCount; i++)
    {
        SOCKET sock = copy.fd_array[i];
        if (sock == listening)
        {
            //accept a new connection
            SOCKET client = accept(listening, nullptr, nullptr);
            cout << "new client connected\n";


            //add the new connection to the list of connected clients *VERY IMPORTANT PART*
            FD_SET(client, &master);


            // send welcome message to the connected clients
            string welcomeMsg = "Welcome to the chat server!\r\n";
            send(client, welcomeMsg.c_str(), welcomeMsg.size() + 1, 0);
        }
```

Otherwise check for a message received from one of the clients if the client enters nothing in their message to the server then the client will be dropped from the server.



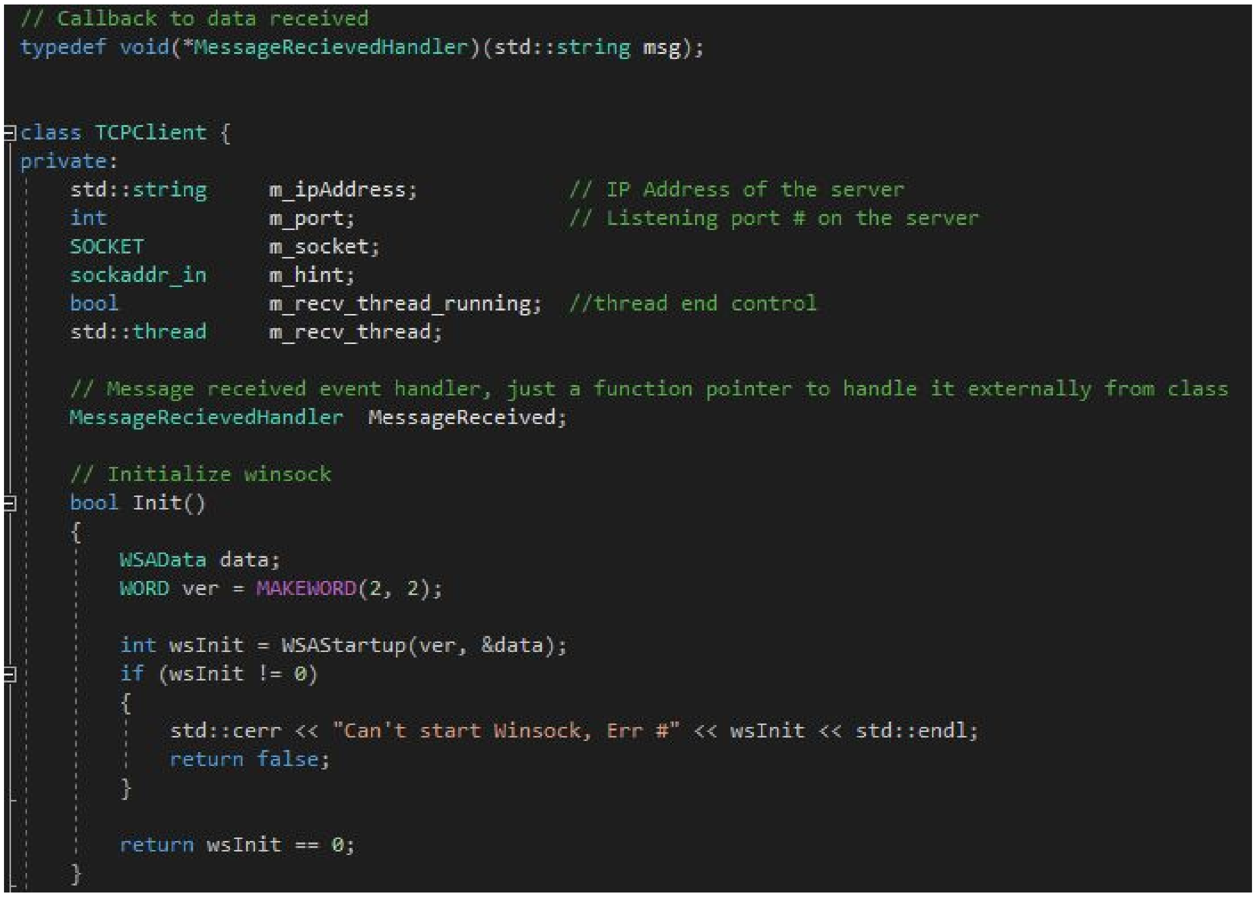
```
else
{
    char buf[4096];
    ZeroMemory(buf, 4096);

    //receive message
    int bytesIn = recv(sock, buf, 4096, 0);
    if (bytesIn <= 0)
    {
        //drop the client
        closesocket(sock);
        FD_CLR(sock, &master);
    }
```

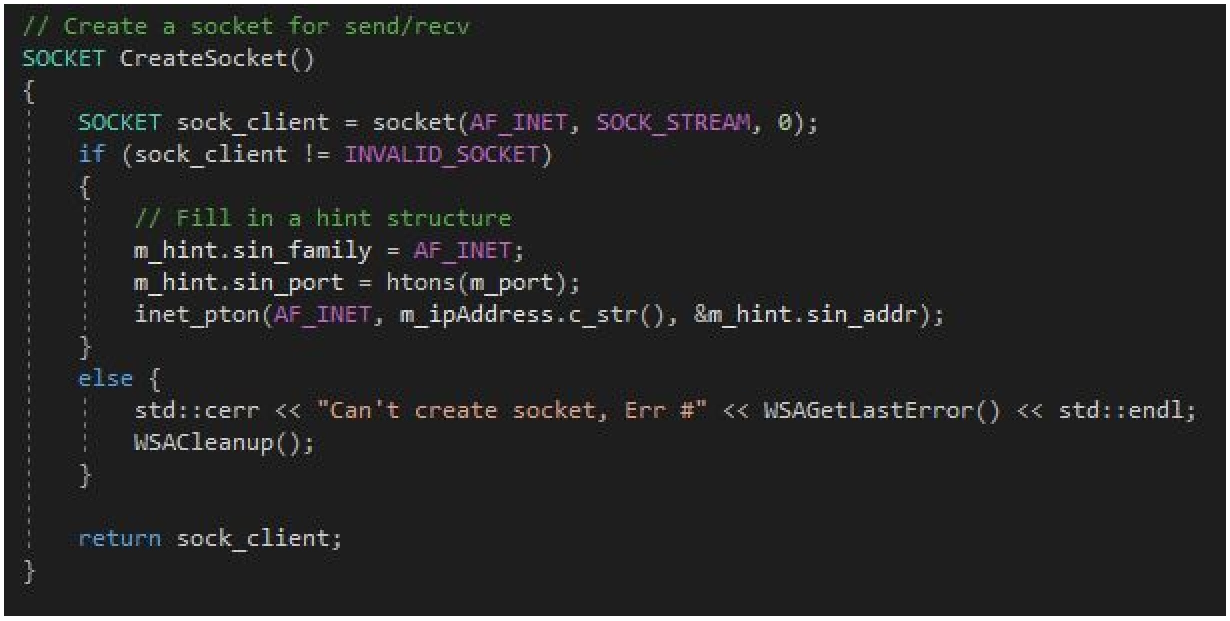
If something is actually sent to the server then it will be sent to the clients; however, not the one that sent the message initially using a for loop and the “send()” function.


```
                else
                {
                    //send message to other clients and definiately NOT the listening client
                    for (int i = 0; i < master.fd_count; i++)
                    {
                        SOCKET outSock = master.fd_array[i];
                        if (outSock != listening && outSock != sock)
                        {
                            send(outSock, buf, bytesIn, 0);
                        }
                    }
                }
            }
        }
    }

    //cleanup winsock
    WSACleanup();
```

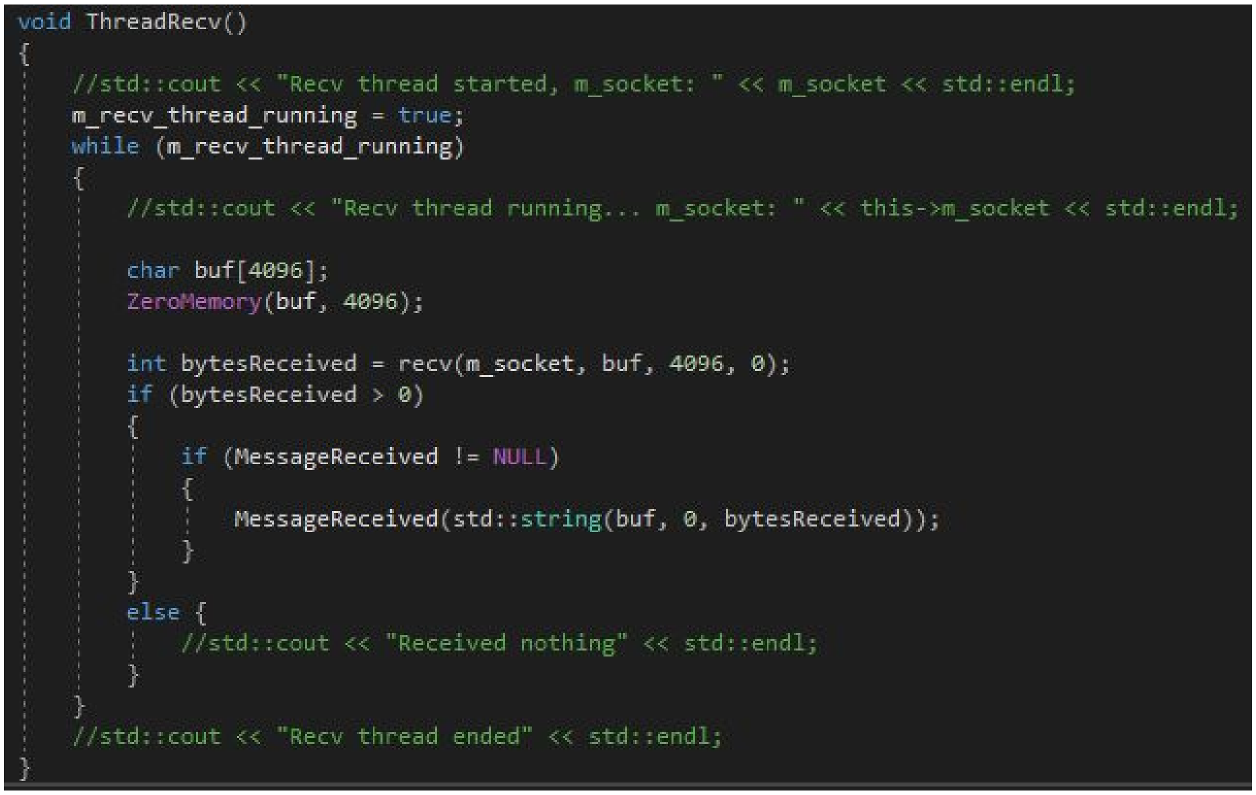
Lastly cleanup winsock using “WSACleanup()”.

- Client code: The client code was based on multi-threading, this was done with the “client.h” which took care of initializing winsock, creating the client, connecting the client to the server and checking the for messages that are received from other clients using a message received handler as well as a “TCPClient” class.

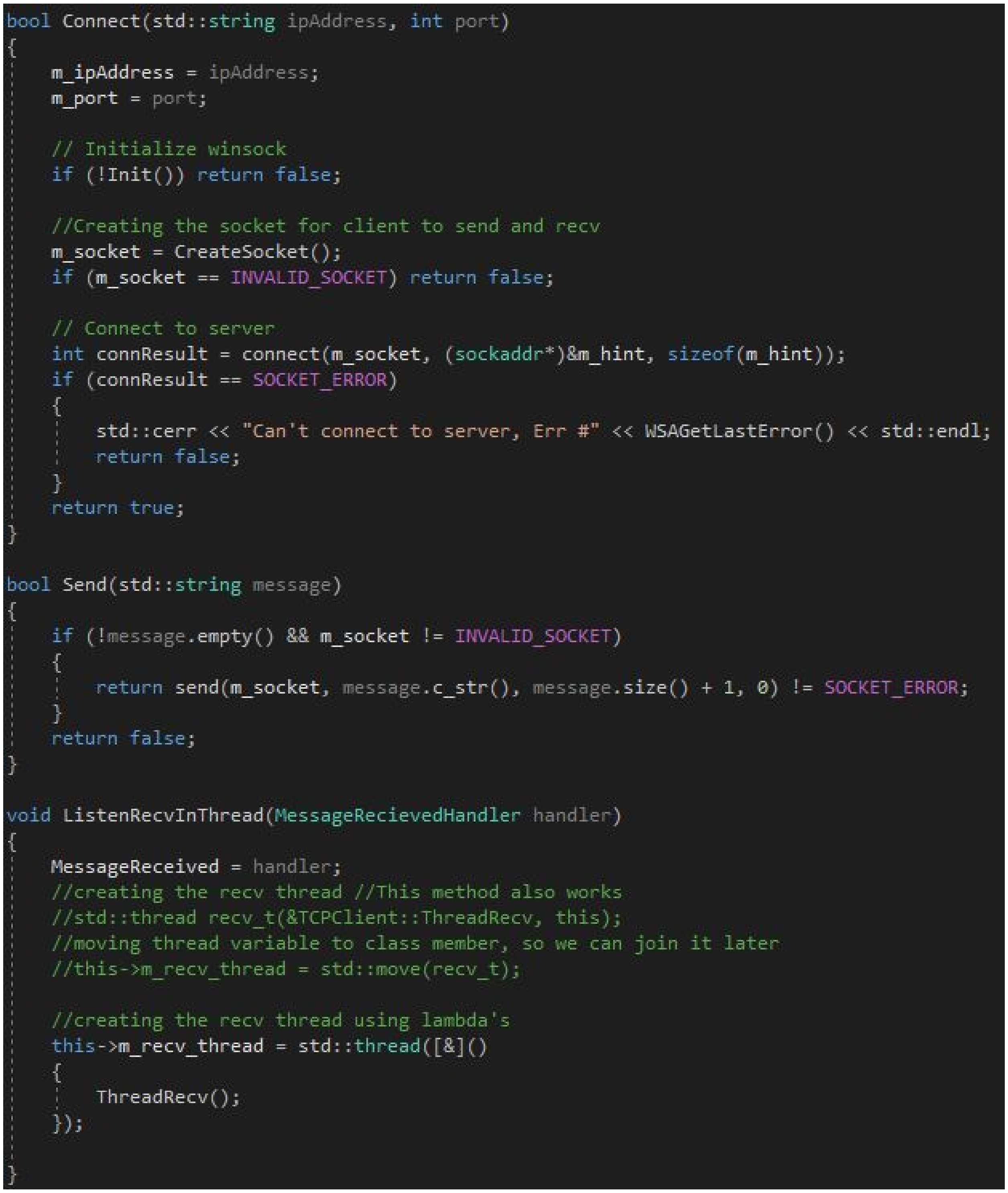

```
// Callback to data received
typedef void(*MessageRecievedHandler)(std::string msg);


class TCPClient {
private:
    std::string     m_ipAddress;            // IP Address of the server
    int             m_port;                 // Listening port # on the server
    SOCKET          m_socket;
    sockaddr_in     m_hint;
    bool            m_recv_thread_running;  //thread end control
    std::thread     m_recv_thread;

    // Message received event handler, just a function pointer to handle it externally from class
    MessageRecievedHandler  MessageReceived;

    // Initialize winsock
    bool Init()
    {
        WSAData data;
        WORD ver = MAKEWORD(2, 2);

        int wsInit = WSAStartup(ver, &data);
        if (wsInit != 0)
        {
            std::cerr << "Can't start Winsock, Err #" << wsInit << std::endl;
            return false;
        }

        return wsInit == 0;
    }
```

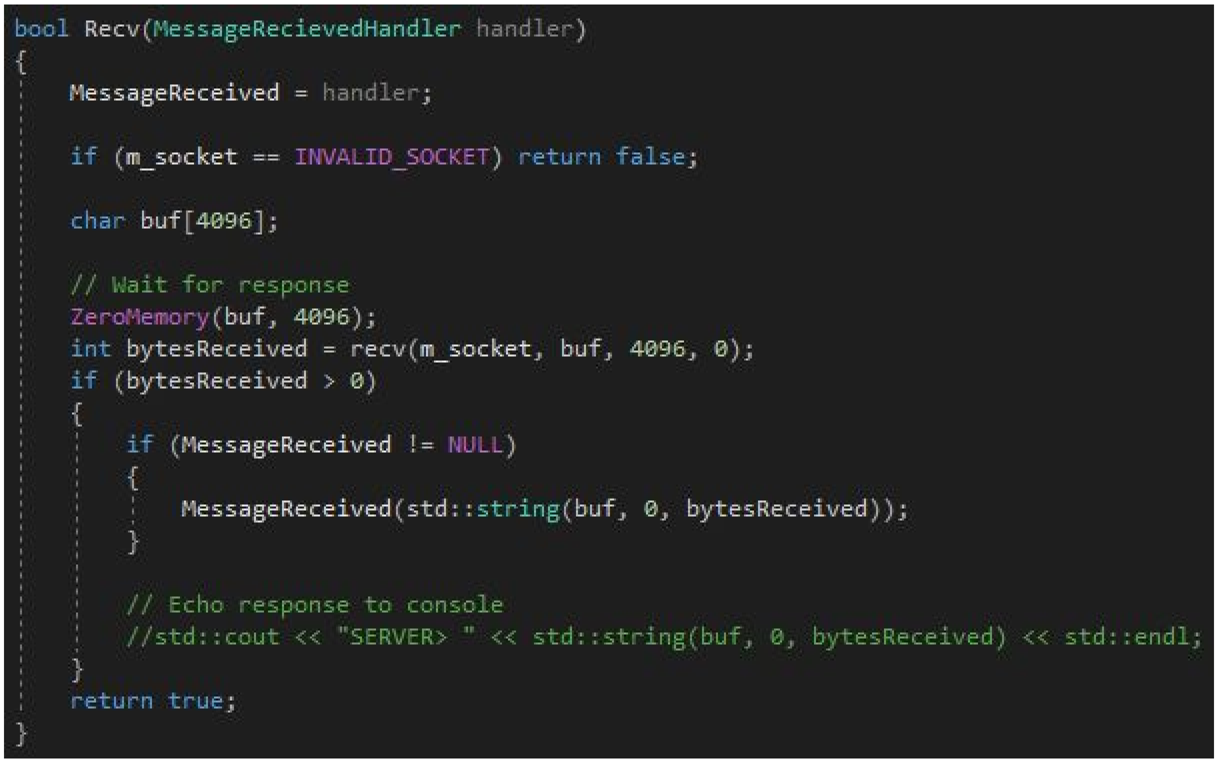


```cpp
// Create a socket for send/recv
SOCKET CreateSocket()
{
    SOCKET sock_client = socket(AF_INET, SOCK_STREAM, 0);
    if (sock_client != INVALID_SOCKET)
    {
        // Fill in a hint structure
        m_hint.sin_family = AF_INET;
        m_hint.sin_port = htons(m_port);
        inet_pton(AF_INET, m_ipAddress.c_str(), &m_hint.sin_addr);
    }
    else {
        std::cerr << "Can't create socket, Err #" << WSAGetLastError() << std::endl;
        WSACleanup();
    }

    return sock_client;
}
```



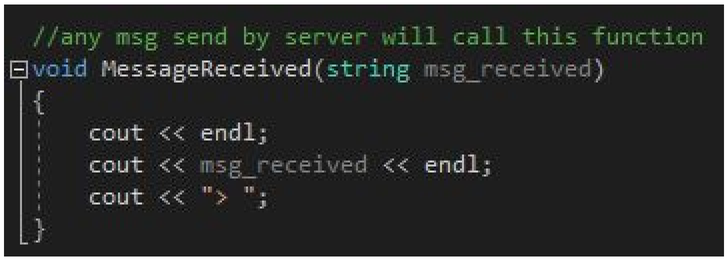
```cpp
void ThreadRecv()
{
    //std::cout << "Recv thread started, m_socket: " << m_socket << std::endl;
    m_recv_thread_running = true;
    while (m_recv_thread_running)
    {
        //std::cout << "Recv thread running... m_socket: " << this->m_socket << std::endl;

        char buf[4096];
        ZeroMemory(buf, 4096);

        int bytesReceived = recv(m_socket, buf, 4096, 0);
        if (bytesReceived > 0)
        {
            if (MessageReceived != NULL)
            {
                MessageReceived(std::string(buf, 0, bytesReceived));
            }
        }
        else {
            //std::cout << "Received nothing" << std::endl;
        }
    }
    //std::cout << "Recv thread ended" << std::endl;
}
```


```cpp
bool Connect(std::string ipAddress, int port)
{
    m_ipAddress = ipAddress;
    m_port = port;

    // Initialize winsock
    if (!Init()) return false;

    //Creating the socket for client to send and recv
    m_socket = CreateSocket();
    if (m_socket == INVALID_SOCKET) return false;

    // Connect to server
    int connResult = connect(m_socket, (sockaddr*)&m_hint, sizeof(m_hint));
    if (connResult == SOCKET_ERROR)
    {
        std::cerr << "Can't connect to server, Err #" << WSAGetLastError() << std::endl;
        return false;
    }
    return true;
}

bool Send(std::string message)
{
    if (!message.empty() && m_socket != INVALID_SOCKET)
    {
        return send(m_socket, message.c_str(), message.size() + 1, 0) != SOCKET_ERROR;
    }
    return false;
}

void ListenRecvInThread(MessageRecievedHandler handler)
{
    MessageReceived = handler;
    //creating the recv thread //This method also works
    //std::thread recv_t(&TCPClient::ThreadRecv, this);
    //moving thread variable to class member, so we can join it later
    //this->m_recv_thread = std::move(recv_t);

    //creating the recv thread using lambda's
    this->m_recv_thread = std::thread([&]()
    {
        ThreadRecv();
    });

}
```



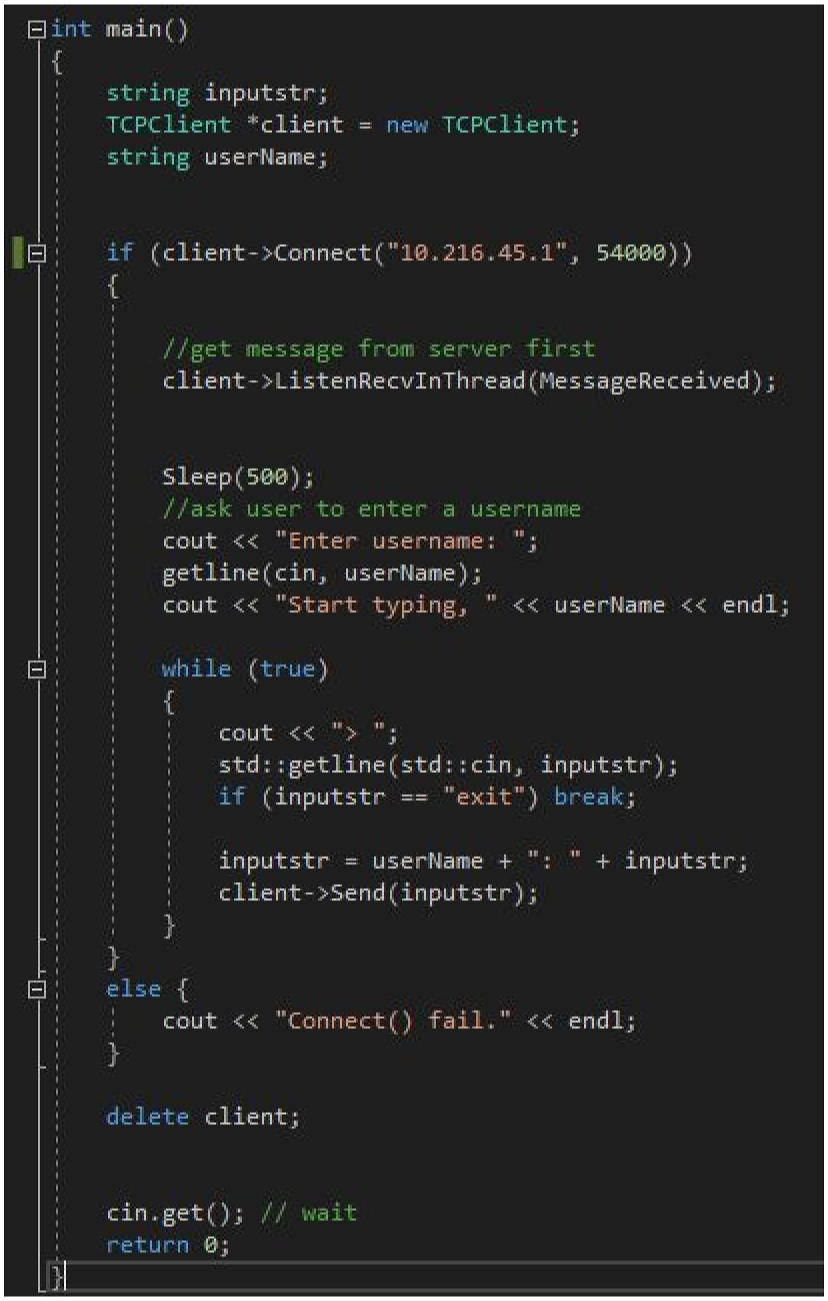
```
bool Recv(MessageRecievedHandler handler)
{
    MessageReceived = handler;

    if (m_socket == INVALID_SOCKET) return false;

    char buf[4096];

    // Wait for response
    ZeroMemory(buf, 4096);
    int bytesReceived = recv(m_socket, buf, 4096, 0);
    if (bytesReceived > 0)
    {
        if (MessageReceived != NULL)
        {
            MessageReceived(std::string(buf, 0, bytesReceived));
        }

        // Echo response to console
        //std::cout << "SERVER> " << std::string(buf, 0, bytesReceived) << std::endl;
    }
    return true;
}
```

From using this header file the "main.cpp" was a lot simpler to code. The first void function will output and message that is received from the clients that are sent to the server.



```
//any msg send by server will call this function
void MessageReceived(string msg_received)
{
    cout << endl;
    cout << msg_received << endl;
    cout << "> ";
}
```

In the main function we check if the client is connected to the local server host and use the function "connect()" with two parameters which are IP address and port. These values will change depending on what the local IP address the server is running the port set in the server code. If need to run on another client all they would have to do is change the IP address. The "sleep()" function is only used for a delay so the welcome message from the server is formatted to show up first and then ask the user for a username that will be used to distinguish who sent the message in our while loop. This while loop is used to get the message from the client using the "getline()" function. Once the user inputs something the "send()" function is used and sent to the server which then sends it to the other clients that are connected.

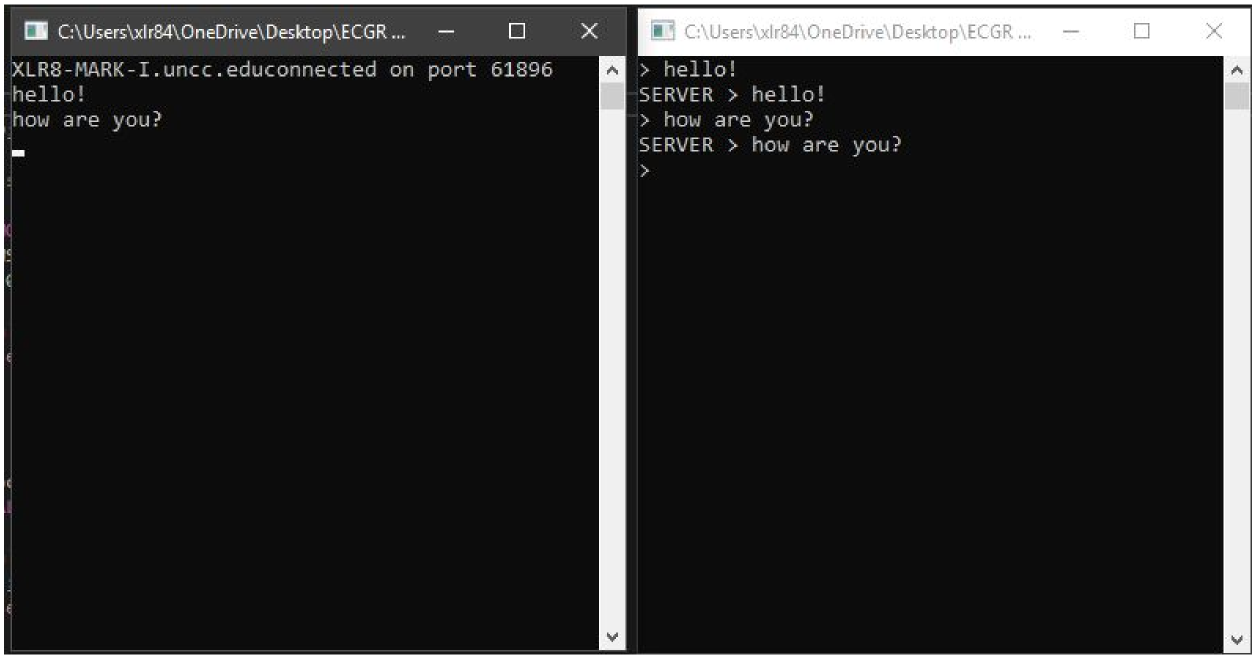


```cpp
int main()
{
    string inputstr;
    TCPClient *client = new TCPClient;
    string userName;


    if (client->Connect("10.216.45.1", 54000))
    {

        //get message from server first
        client->ListenRecvInThread(MessageReceived);


        Sleep(500);
        //ask user to enter a username
        cout << "Enter username: ";
        getline(cin, userName);
        cout << "Start typing, " << userName << endl;

        while (true)
        {
            cout << "> ";
            std::getline(std::cin, inputstr);
            if (inputstr == "exit") break;

            inputstr = userName + ": " + inputstr;
            client->Send(inputstr);
        }
    }
    else {
        cout << "Connect() fail." << endl;
    }

    delete client;


    cin.get(); // wait
    return 0;
}
```

Lastly delete the client from the server and exit the program.

Code Example Pictures:

- Client Server Communication:



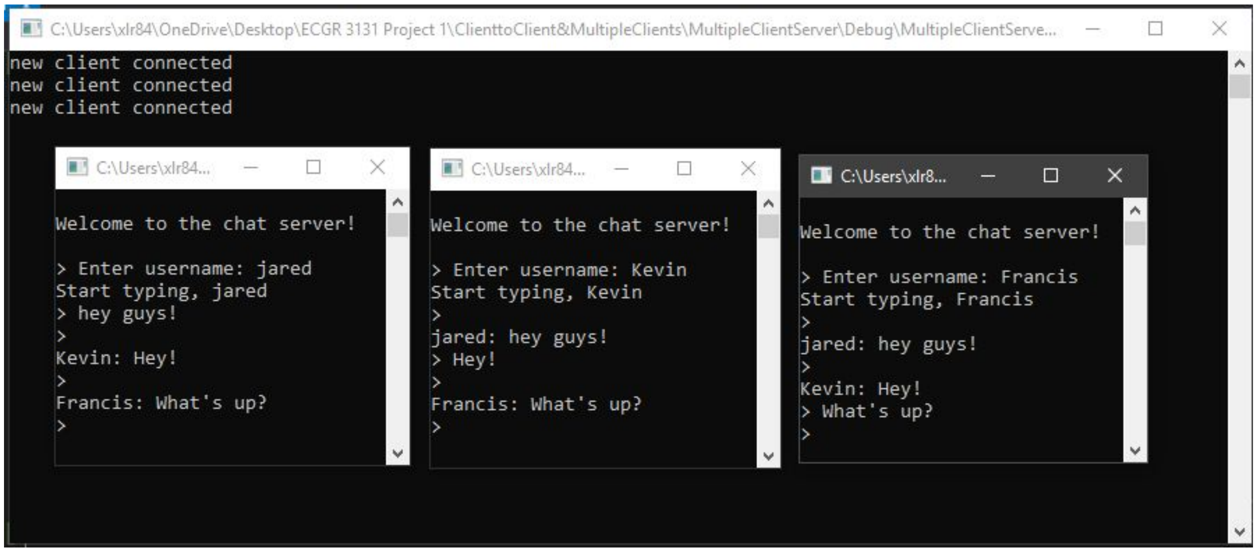
C:\Users\xlr84\OneDrive\Desktop\ECGR ...

```
XLR8-MARK-I.uncc.educonnected on port 61896
hello!
how are you?
```

C:\Users\xlr84\OneDrive\Desktop\ECGR ...

```
> hello!
SERVER > hello!
> how are you?
SERVER > how are you?
>
```

- Multi-Client Communication:



C:\Users\xlr84\OneDrive\Desktop\ECGR 3131 Project 1\ClienttoClient&MultipleClients\MultipleClientServer\Debug\MultipleClientServe...

```
new client connected
new client connected
new client connected
```

C:\Users\xlr84...

```
Welcome to the chat server!

> Enter username: jared
Start typing, jared
> hey guys!
>
Kevin: Hey!
>
Francis: What's up?
>
```

C:\Users\xlr84...

```
Welcome to the chat server!

> Enter username: Kevin
Start typing, Kevin
>
jared: hey guys!
> Hey!
>
Francis: What's up?
>
```

C:\Users\xlr8...

```
Welcome to the chat server!

> Enter username: Francis
Start typing, Francis
>
jared: hey guys!
>
Kevin: Hey!
> What's up?
>
```

References:

1. Han, Tao. "Lecture_5_release.", https://uncc.instructure.com/courses/92249/files/5302057?module_item_id=1730222
2. "GeeksforGeeks" website,

https://www.geeksforgeeks.org/socket-programming-cc/

3. "Soloan Kelly Youtube Channel" website, https://www.youtube.com/results?search_query=soloan+kelly