Day -14 LSP Assignment (Task -1)

1. TCP Server-Client Communication:

Problem Statement: Write a TCP server and client program in C++ where the server listens for incoming connections and echoes back any message it receives from the client. The client should be able to send a message to the server and display the echoed message.

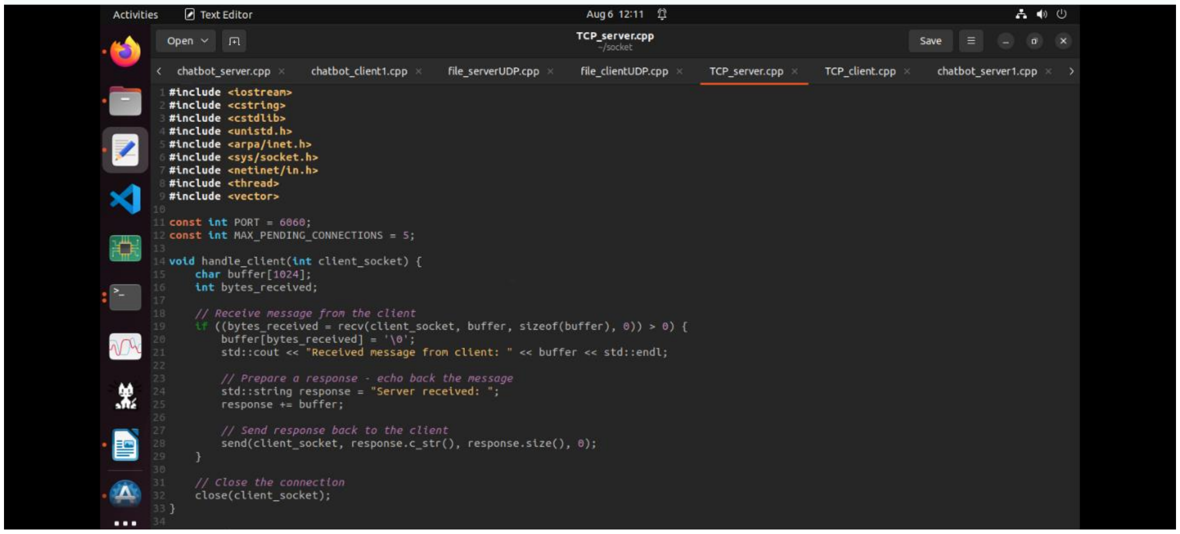
Requirements:

The server should run indefinitely, waiting for client connections.

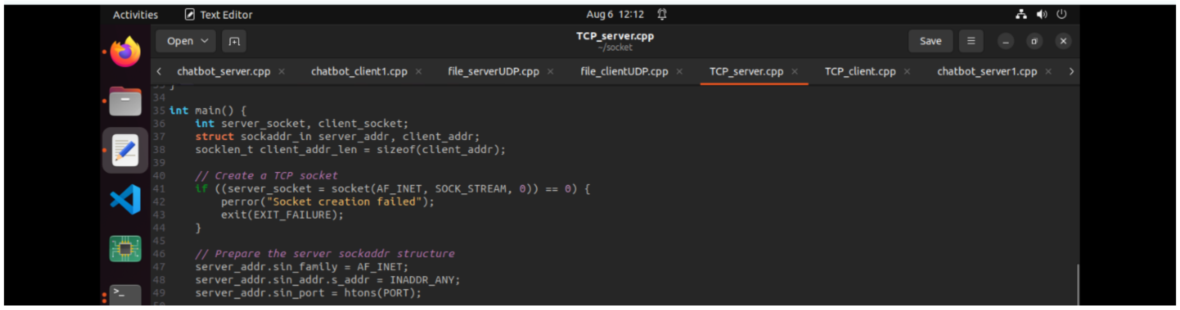
The client should take a message as input from the user, send it to the server, and display the response.

Implement proper error handling and cleanup (e.g., closing sockets).

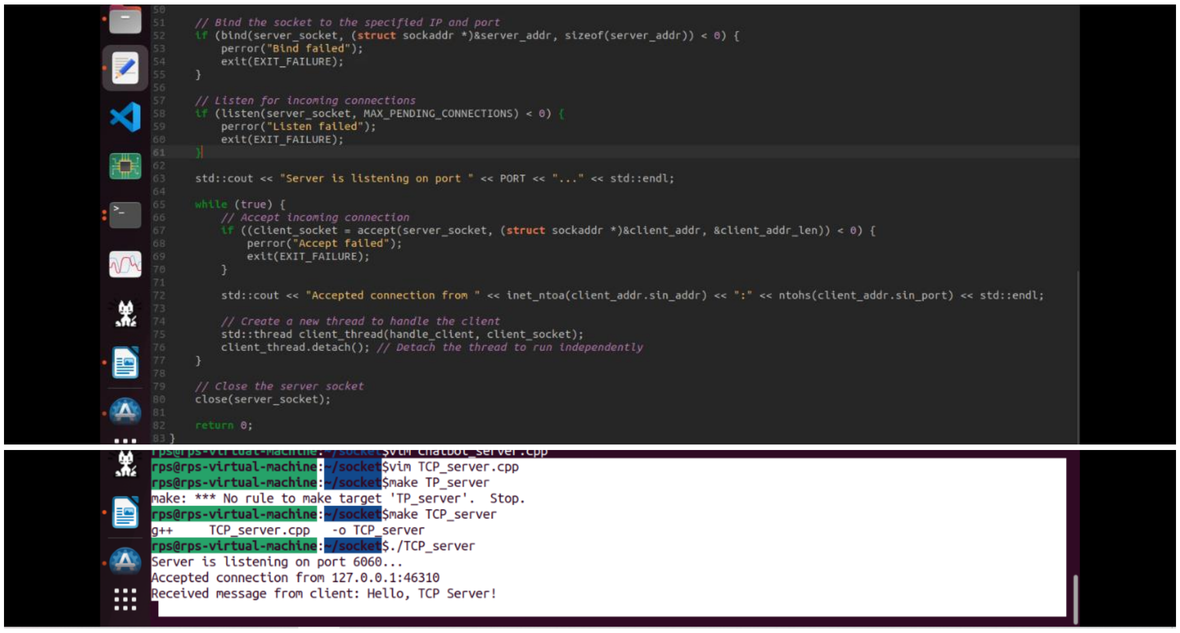
a. Server- side



```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <thread>
#include <vector>

const int PORT = 6060;
const int MAX_PENDING_CONNECTIONS = 5;

void handle_client(int client_socket) {
    char buffer[1024];
    int bytes_received;

    // Receive message from the client
    if ((bytes_received = recv(client_socket, buffer, sizeof(buffer), 0)) > 0) {
        buffer[bytes_received] = '\0';
        std::cout << "Received message from client: " << buffer << std::endl;

        // Prepare a response - echo back the message
        std::string response = "Server received: ";
        response += buffer;

        // Send response back to the client
        send(client_socket, response.c_str(), response.size(), 0);
    }

    // Close the connection
    close(client_socket);
}

```



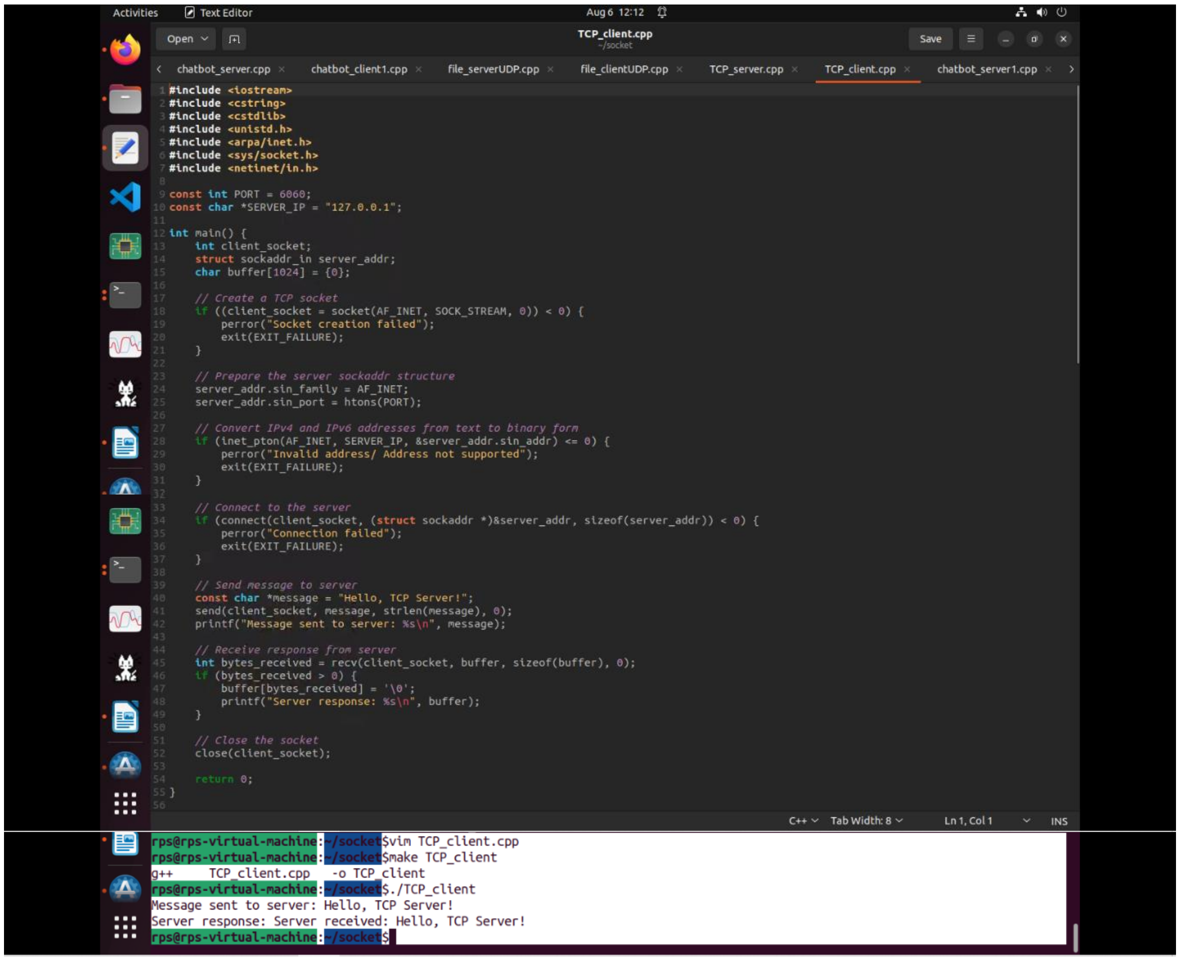
```cpp

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len = sizeof(client_addr);

    // Create a TCP socket
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Prepare the server sockaddr structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);
```



```cpp
    // Bind the socket to the specified IP and port
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_socket, MAX_PENDING_CONNECTIONS) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    std::cout << "Server is listening on port " << PORT << "..." << std::endl;

    while (true) {
        // Accept incoming connection
        if ((client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_addr_len)) < 0) {
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }

        std::cout << "Accepted connection from " << inet_ntoa(client_addr.sin_addr) << ":" << ntohs(client_addr.sin_port) << std::endl;

        // Create a new thread to handle the client
        std::thread client_thread(handle_client, client_socket);
        client_thread.detach(); // Detach the thread to run independently
    }

    // Close the server socket
    close(server_socket);

    return 0;
}
```

```
rps@rps-virtual-machine:~/socket$vim chatbot_server.cpp
rps@rps-virtual-machine:~/socket$vim TCP_server.cpp
rps@rps-virtual-machine:~/socket$make TP_server
make: *** No rule to make target 'TP_server'.  Stop.
rps@rps-virtual-machine:~/socket$make TCP_server
g++     TCP_server.cpp   -o TCP_server
rps@rps-virtual-machine:~/socket$./TCP_server
Server is listening on port 6060...
Accepted connection from 127.0.0.1:46310
Received message from client: Hello, TCP Server!
```

b. client - side



Activities | Text Editor | Aug 6 12:12

TCP_client.cpp ~/socket

chatbot_server.cpp | chatbot_client1.cpp | file_serverUDP.cpp | file_clientUDP.cpp | TCP_server.cpp | TCP_client.cpp | chatbot_server1.cpp

```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

const int PORT = 6060;
const char *SERVER_IP = "127.0.0.1";

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    char buffer[1024] = {0};

    // Create a TCP socket
    if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Prepare the server sockaddr structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to the server
    if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Send message to server
    const char *message = "Hello, TCP Server!";
    send(client_socket, message, strlen(message), 0);
    printf("Message sent to server: %s\n", message);

    // Receive response from server
    int bytes_received = recv(client_socket, buffer, sizeof(buffer), 0);
    if (bytes_received > 0) {
        buffer[bytes_received] = '\0';
        printf("Server response: %s\n", buffer);
    }

    // Close the socket
    close(client_socket);

    return 0;
}
```

C++ | Tab Width: 8 | Ln 1, Col 1 | INS

```
rps@rps-virtual-machine:~/socket$vim TCP_client.cpp
rps@rps-virtual-machine:~/socket$make TCP_client
g++     TCP_client.cpp   -o TCP_client
rps@rps-virtual-machine:~/socket$./TCP_client
Message sent to server: Hello, TCP Server!
Server response: Server received: Hello, TCP Server!
rps@rps-virtual-machine:~/socket$
```

2. UDP Server-Client Communication:

Problem Statement: Write a UDP server and client program in C++ where the server listens on a specific port and responds with "Hello, Client!" whenever it receives a message. The client should send a message to the server and print the response.

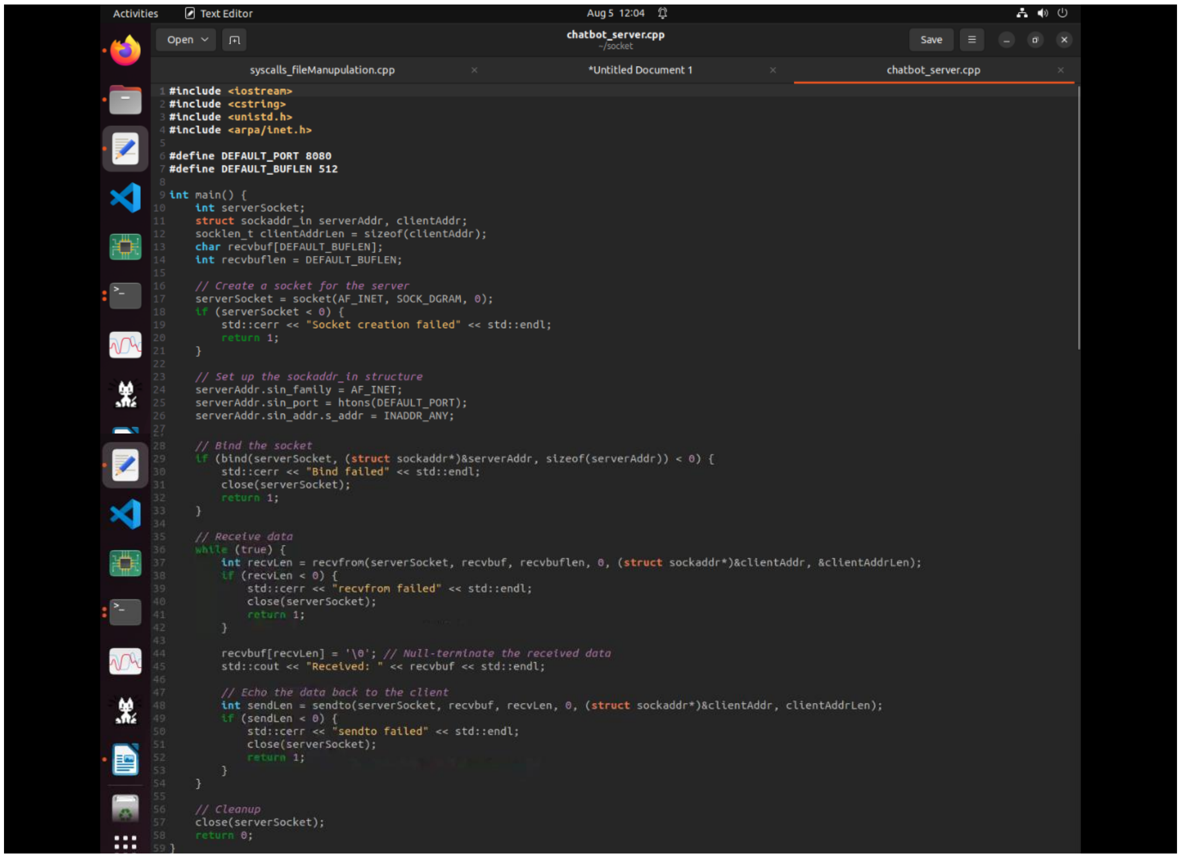
Requirements:

The server should run indefinitely, waiting for incoming messages.

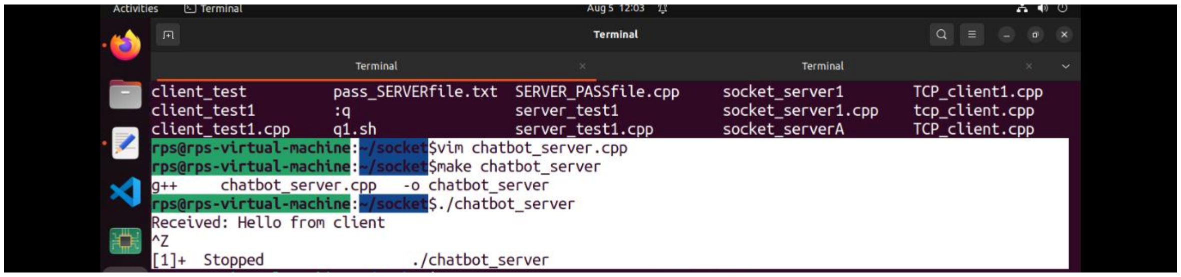
The client should send a predefined message (e.g., "Hello, Server!") and display the server's response.

Implement proper error handling.

a. Server side

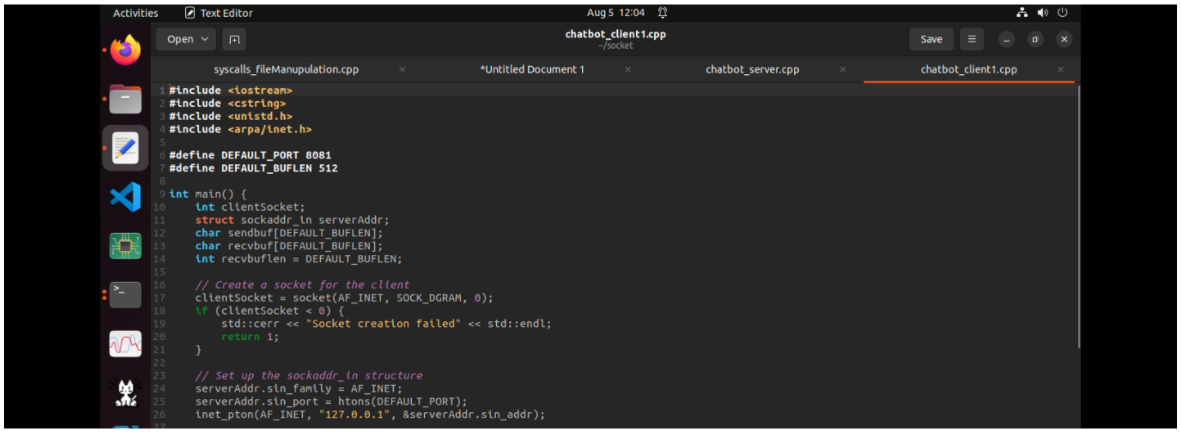


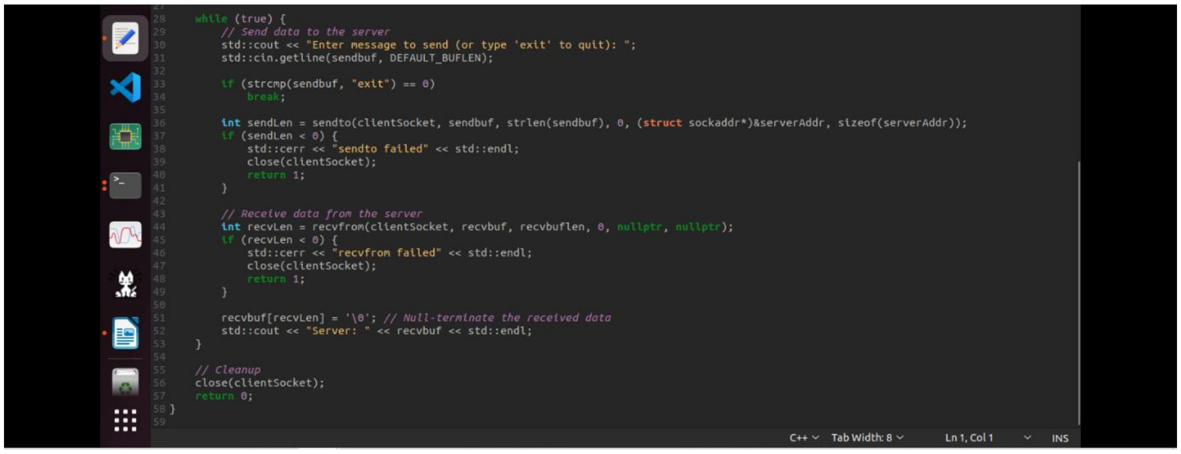
```cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8080
#define DEFAULT_BUFLEN 512

int main() {
    int serverSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t clientAddrLen = sizeof(clientAddr);
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the server
    serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (serverSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
        std::cerr << "Bind failed" << std::endl;
        close(serverSocket);
        return 1;
    }

    // Receive data
    while (true) {
        int recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
        if (recvLen < 0) {
            std::cerr << "recvfrom failed" << std::endl;
            close(serverSocket);
            return 1;
        }

        recvbuf[recvLen] = '\0'; // Null-terminate the received data
        std::cout << "Received: " << recvbuf << std::endl;

        // Echo the data back to the client
        int sendLen = sendto(serverSocket, recvbuf, recvLen, 0, (struct sockaddr*)&clientAddr, clientAddrLen);
        if (sendLen < 0) {
            std::cerr << "sendto failed" << std::endl;
            close(serverSocket);
            return 1;
        }
    }

    // Cleanup
    close(serverSocket);
    return 0;
}
```



```
client_test          pass_SERVERfile.txt  SERVER_PASSfile.cpp  socket_server1      TCP_client1.cpp
client_test1         :q                   server_test1         socket_server1.cpp  tcp_client.cpp
client_test1.cpp     q1.sh                server_test1.cpp     socket_serverA      TCP_client.cpp
rps@rps-virtual-machine:~/socket$vim chatbot_server.cpp
rps@rps-virtual-machine:~/socket$make chatbot_server
g++     chatbot_server.cpp    -o chatbot_server
rps@rps-virtual-machine:~/socket$./chatbot_server
Received: Hello from client
^Z
[1]+  Stopped                 ./chatbot_server
```

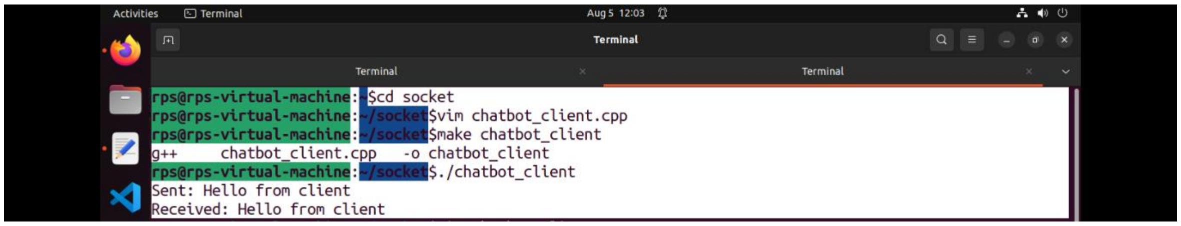
b. Client- side





chatbot_client1.cpp

```cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8081
#define DEFAULT_BUFLEN 512

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char sendbuf[DEFAULT_BUFLEN];
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the client
    clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (clientSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);

    while (true) {
        // Send data to the server
        std::cout << "Enter message to send (or type 'exit' to quit): ";
        std::cin.getline(sendbuf, DEFAULT_BUFLEN);

        if (strcmp(sendbuf, "exit") == 0)
            break;

        int sendLen = sendto(clientSocket, sendbuf, strlen(sendbuf), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
        if (sendLen < 0) {
            std::cerr << "sendto failed" << std::endl;
            close(clientSocket);
            return 1;
        }

        // Receive data from the server
        int recvLen = recvfrom(clientSocket, recvbuf, recvbuflen, 0, nullptr, nullptr);
        if (recvLen < 0) {
            std::cerr << "recvfrom failed" << std::endl;
            close(clientSocket);
            return 1;
        }

        recvbuf[recvLen] = '\0'; // Null-terminate the received data
        std::cout << "Server: " << recvbuf << std::endl;
    }

    // Cleanup
    close(clientSocket);
    return 0;
}
```



```
rps@rps-virtual-machine:~$cd socket
rps@rps-virtual-machine:~/socket$vim chatbot_client.cpp
rps@rps-virtual-machine:~/socket$make chatbot_client
g++     chatbot_client.cpp    -o chatbot_client
rps@rps-virtual-machine:~/socket$./chatbot_client
Sent: Hello from client
Received: Hello from client
```

3. File Transfer using TCP:

Problem Statement: Write a TCP server and client program in C++ to transfer a file from the client to the server. The server should save the received file with the same name, and the client should specify the file to be sent.

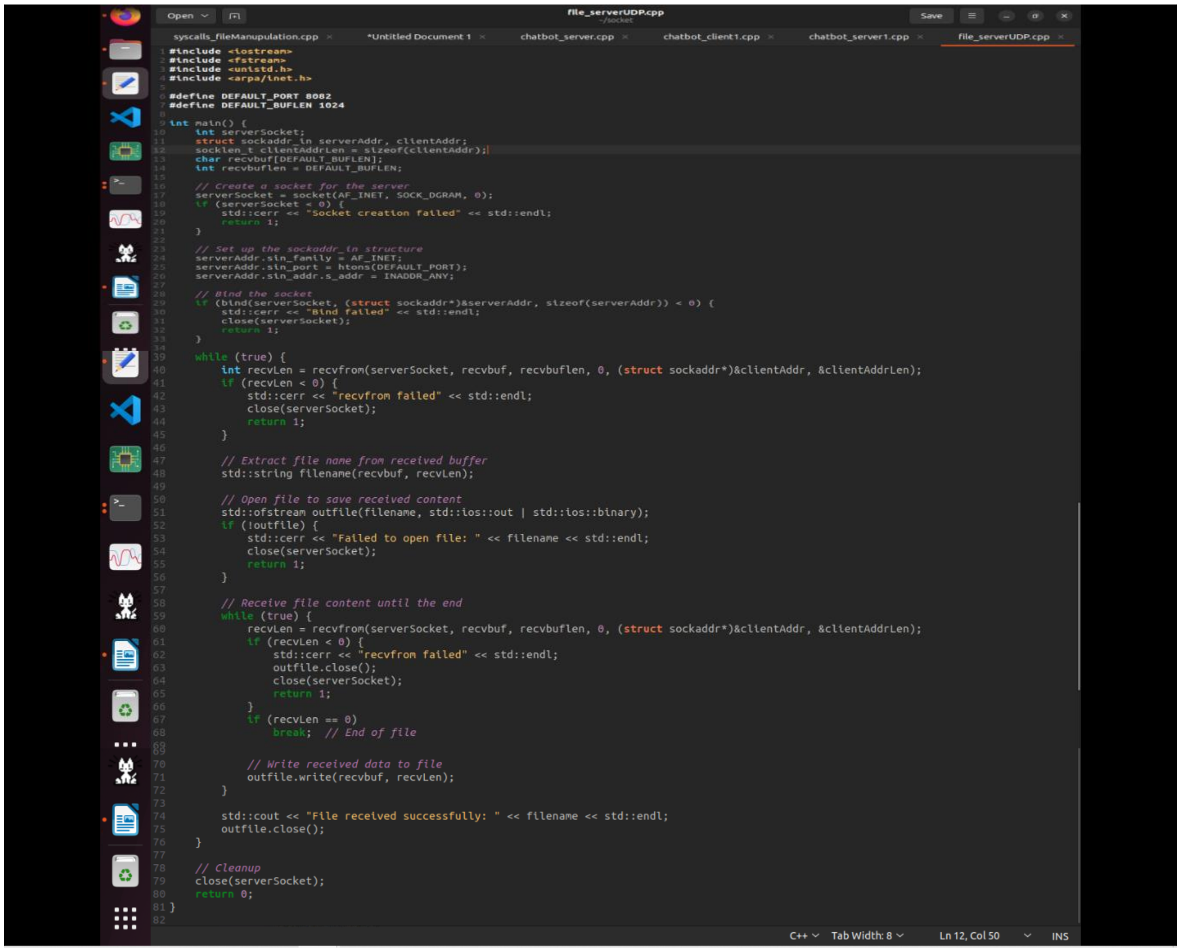
Requirements:

The server should run indefinitely, waiting for file transfer requests.

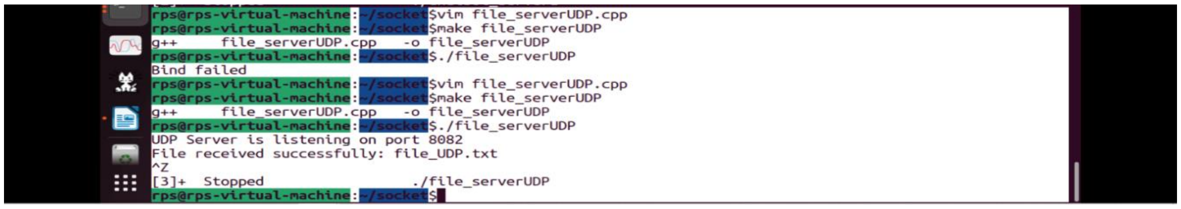
The client should prompt the user for a file path, read the file, and send its contents to the server.

Implement proper error handling and file operations.

a. Server side

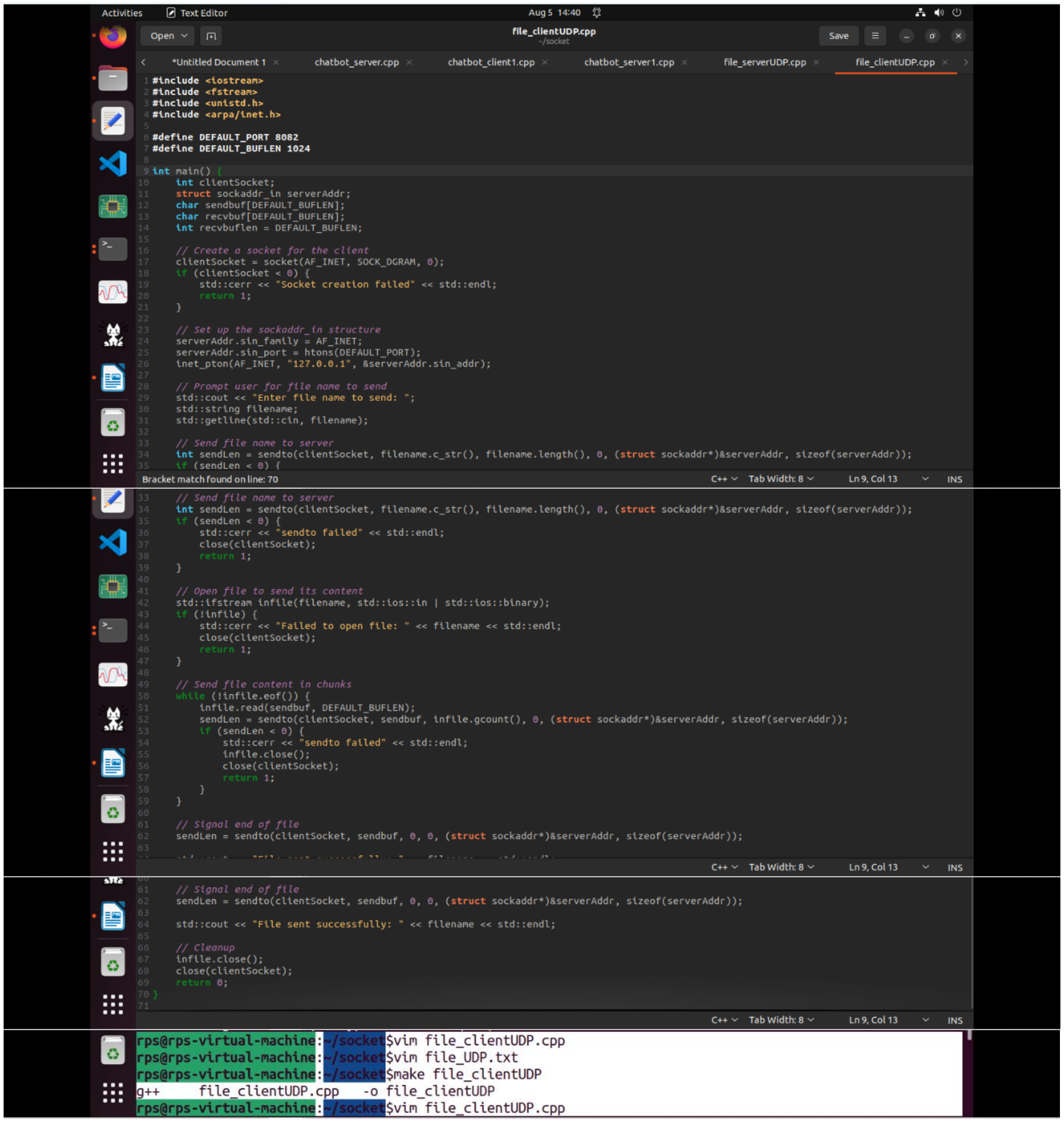


```cpp
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8082
#define DEFAULT_BUFLEN 1024

int main() {
    int serverSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t clientAddrLen = sizeof(clientAddr);
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the server
    serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (serverSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
        std::cerr << "Bind failed" << std::endl;
        close(serverSocket);
        return 1;
    }

    while (true) {
        int recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
        if (recvLen < 0) {
            std::cerr << "recvfrom failed" << std::endl;
            close(serverSocket);
            return 1;
        }

        // Extract file name from received buffer
        std::string filename(recvbuf, recvLen);

        // Open file to save received content
        std::ofstream outfile(filename, std::ios::out | std::ios::binary);
        if (!outfile) {
            std::cerr << "Failed to open file: " << filename << std::endl;
            close(serverSocket);
            return 1;
        }

        // Receive file content until the end
        while (true) {
            recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
            if (recvLen < 0) {
                std::cerr << "recvfrom failed" << std::endl;
                outfile.close();
                close(serverSocket);
                return 1;
            }
            if (recvLen == 0)
                break;  // End of file

            // Write received data to file
            outfile.write(recvbuf, recvLen);
        }

        std::cout << "File received successfully: " << filename << std::endl;
        outfile.close();
    }

    // Cleanup
    close(serverSocket);
    return 0;
}
```



```
rps@rps-virtual-machine:~/socket$vim file_serverUDP.cpp
rps@rps-virtual-machine:~/socket$make file_serverUDP
g++     file_serverUDP.cpp    -o file_serverUDP
rps@rps-virtual-machine:~/socket$./file_serverUDP
Bind failed
rps@rps-virtual-machine:~/socket$vim file_serverUDP.cpp
rps@rps-virtual-machine:~/socket$make file_serverUDP
g++     file_serverUDP.cpp    -o file_serverUDP
rps@rps-virtual-machine:~/socket$./file_serverUDP
UDP Server is listening on port 8082
File received successfully: file_UDP.txt
^Z
[3]+  Stopped                 ./file_serverUDP
rps@rps-virtual-machine:~/socket$
```

b. **Client side**



```cpp
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8082
#define DEFAULT_BUFLEN 1024

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char sendbuf[DEFAULT_BUFLEN];
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the client
    clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (clientSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);

    // Prompt user for file name to send
    std::cout << "Enter file name to send: ";
    std::string filename;
    std::getline(std::cin, filename);

    // Send file name to server
    int sendLen = sendto(clientSocket, filename.c_str(), filename.length(), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    if (sendLen < 0) {
        std::cerr << "sendto failed" << std::endl;
        close(clientSocket);
        return 1;
    }

    // Open file to send its content
    std::ifstream infile(filename, std::ios::in | std::ios::binary);
    if (!infile) {
        std::cerr << "Failed to open file: " << filename << std::endl;
        close(clientSocket);
        return 1;
    }

    // Send file content in chunks
    while (!infile.eof()) {
        infile.read(sendbuf, DEFAULT_BUFLEN);
        sendLen = sendto(clientSocket, sendbuf, infile.gcount(), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
        if (sendLen < 0) {
            std::cerr << "sendto failed" << std::endl;
            infile.close();
            close(clientSocket);
            return 1;
        }
    }

    // Signal end of file
    sendLen = sendto(clientSocket, sendbuf, 0, 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));

    std::cout << "File sent successfully: " << filename << std::endl;

    // Cleanup
    infile.close();
    close(clientSocket);
    return 0;
}
```

```
rps@rps-virtual-machine:~/socket$vim file_clientUDP.cpp
rps@rps-virtual-machine:~/socket$vim file_UDP.txt
rps@rps-virtual-machine:~/socket$make file_clientUDP
g++     file_clientUDP.cpp    -o file_clientUDP
rps@rps-virtual-machine:~/socket$vim file_clientUDP.cpp
```

4. Broadcast Messaging using UDP:

Problem Statement: Write a UDP server and client program in C++ to implement a simple broadcast messaging system. The server should broadcast a message to all clients in the network, and each client should display any broadcast messages it receives.

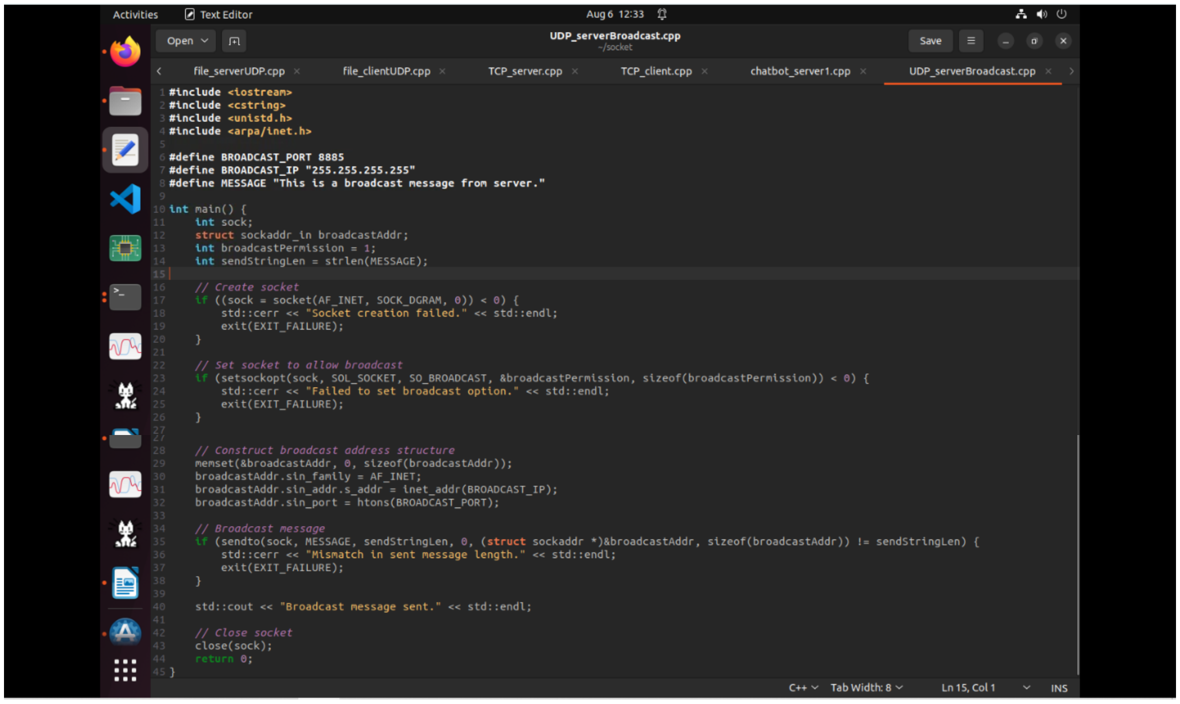
Requirements:

The server should send a broadcast message to a specific port.

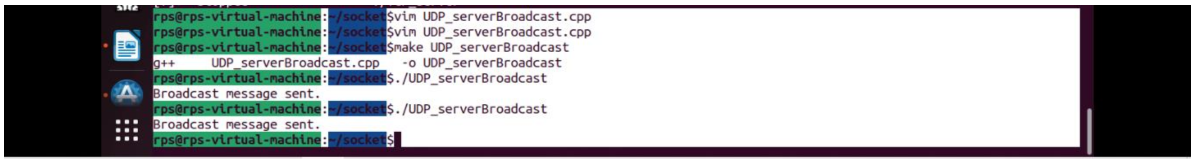
Each client should listen on the same port and display any messages it receives.

Implement proper error handling and use UDP broadcast mechanisms.

a. Server –side

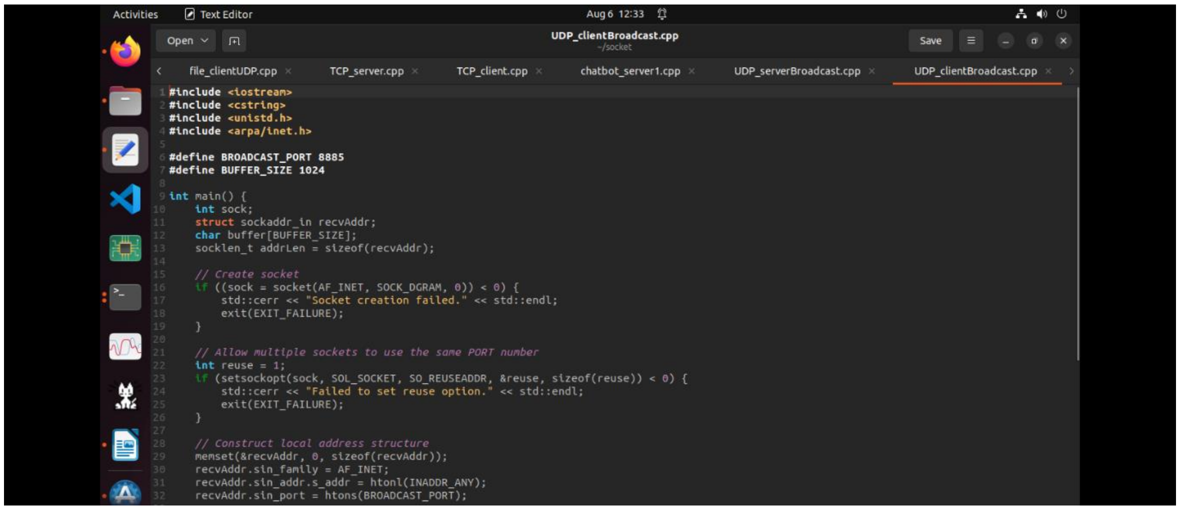


```cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define BROADCAST_PORT 8885
#define BROADCAST_IP "255.255.255.255"
#define MESSAGE "This is a broadcast message from server."

int main() {
    int sock;
    struct sockaddr_in broadcastAddr;
    int broadcastPermission = 1;
    int sendStringLen = strlen(MESSAGE);

    // Create socket
    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        std::cerr << "Socket creation failed." << std::endl;
        exit(EXIT_FAILURE);
    }

    // Set socket to allow broadcast
    if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &broadcastPermission, sizeof(broadcastPermission)) < 0) {
        std::cerr << "Failed to set broadcast option." << std::endl;
        exit(EXIT_FAILURE);
    }


    // Construct broadcast address structure
    memset(&broadcastAddr, 0, sizeof(broadcastAddr));
    broadcastAddr.sin_family = AF_INET;
    broadcastAddr.sin_addr.s_addr = inet_addr(BROADCAST_IP);
    broadcastAddr.sin_port = htons(BROADCAST_PORT);

    // Broadcast message
    if (sendto(sock, MESSAGE, sendStringLen, 0, (struct sockaddr *)&broadcastAddr, sizeof(broadcastAddr)) != sendStringLen) {
        std::cerr << "Mismatch in sent message length." << std::endl;
        exit(EXIT_FAILURE);
    }

    std::cout << "Broadcast message sent." << std::endl;

    // Close socket
    close(sock);
    return 0;
}
```

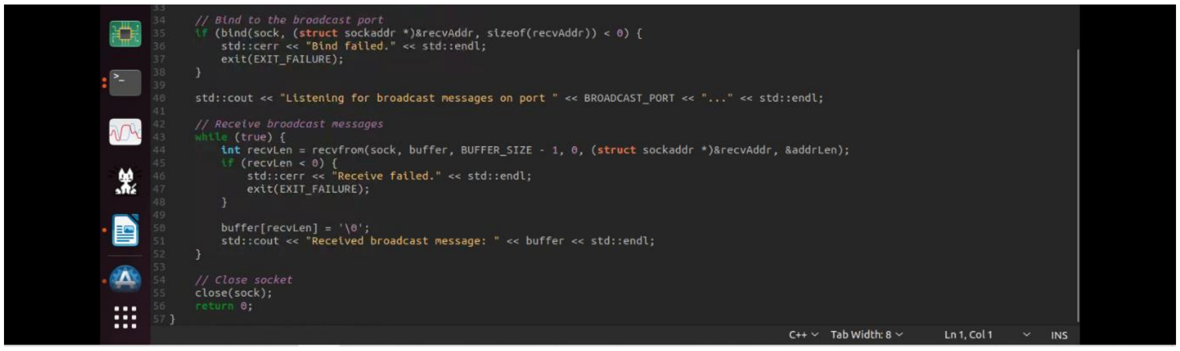


```
rps@rps-virtual-machine:~/socket$vim UDP_serverBroadcast.cpp
rps@rps-virtual-machine:~/socket$vim UDP_serverBroadcast.cpp
rps@rps-virtual-machine:~/socket$make UDP_serverBroadcast
g++     UDP_serverBroadcast.cpp    -o UDP_serverBroadcast
rps@rps-virtual-machine:~/socket$./UDP_serverBroadcast
Broadcast message sent.
rps@rps-virtual-machine:~/socket$./UDP_serverBroadcast
Broadcast message sent.
rps@rps-virtual-machine:~/socket$
```

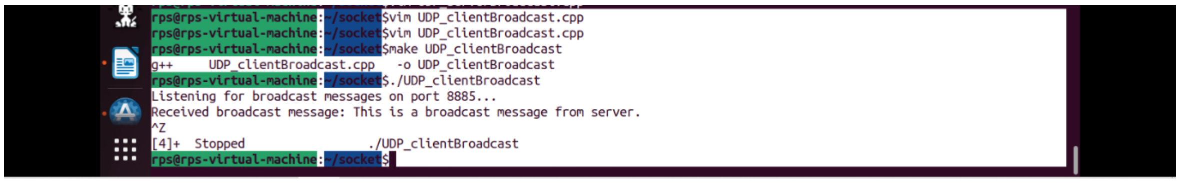
b. Client – side



```cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define BROADCAST_PORT 8885
#define BUFFER_SIZE 1024

int main() {
    int sock;
    struct sockaddr_in recvAddr;
    char buffer[BUFFER_SIZE];
    socklen_t addrLen = sizeof(recvAddr);

    // Create socket
    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        std::cerr << "Socket creation failed." << std::endl;
        exit(EXIT_FAILURE);
    }

    // Allow multiple sockets to use the same PORT number
    int reuse = 1;
    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse)) < 0) {
        std::cerr << "Failed to set reuse option." << std::endl;
        exit(EXIT_FAILURE);
    }

    // Construct local address structure
    memset(&recvAddr, 0, sizeof(recvAddr));
    recvAddr.sin_family = AF_INET;
    recvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    recvAddr.sin_port = htons(BROADCAST_PORT);
```



```cpp
    // Bind to the broadcast port
    if (bind(sock, (struct sockaddr *)&recvAddr, sizeof(recvAddr)) < 0) {
        std::cerr << "Bind failed." << std::endl;
        exit(EXIT_FAILURE);
    }

    std::cout << "Listening for broadcast messages on port " << BROADCAST_PORT << "..." << std::endl;

    // Receive broadcast messages
    while (true) {
        int recvLen = recvfrom(sock, buffer, BUFFER_SIZE - 1, 0, (struct sockaddr *)&recvAddr, &addrLen);
        if (recvLen < 0) {
            std::cerr << "Receive failed." << std::endl;
            exit(EXIT_FAILURE);
        }

        buffer[recvLen] = '\0';
        std::cout << "Received broadcast message: " << buffer << std::endl;
    }

    // Close socket
    close(sock);
    return 0;
}
```



```
rps@rps-virtual-machine:~/socket$vim UDP_clientBroadcast.cpp
rps@rps-virtual-machine:~/socket$vim UDP_clientBroadcast.cpp
rps@rps-virtual-machine:~/socket$make UDP_clientBroadcast
g++     UDP_clientBroadcast.cpp   -o UDP_clientBroadcast
rps@rps-virtual-machine:~/socket$./UDP_clientBroadcast
Listening for broadcast messages on port 8885...
Received broadcast message: This is a broadcast message from server.
^Z
[4]+  Stopped                 ./UDP_clientBroadcast
rps@rps-virtual-machine:~/socket$
```