write a Code where when user put ctrl c

Message delivered to server :

Client:

```
rps@rps-virtual-machine:~$ vim client2.cpp
rps@rps-virtual-machine:~$ make client2
g++     client2.cpp   -o client2
rps@rps-virtual-machine:~$ ./client2
Running... Press Ctrl+C to interrupt.
^CCtrl+C pressed! Sending message to server...
Message sent.Exiting.
Server response: Message Recieved
```

```
rps@rps-virtual-machine:~$ cat client2.cpp
#include <iostream>
#include <csignal>
#include <unistd.h>
#include <cstring>
#include <arpa/inet.h>
using namespace std;
void signalHandler(int signum) {
        cout << "Ctrl+C pressed! Sending message to server..." << endl;
        int sock = 0;
        struct sockaddr_in serv_addr;
        const char* message = "User pressed Ctrl+C";
        char buffer[1024] = {0};
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                cerr << "Socket creation error" << endl;
                exit(EXIT_FAILURE);
        }
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(8080);
        if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
                cerr << "Invalid address/ Address not supported" << endl;
                exit(EXIT_FAILURE);
        }
        if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
                cerr << "Connection Failed" << endl;
                exit(EXIT_FAILURE);
        }
        send(sock, message, strlen(message), 0);
        cout << "Message sent.Exiting." << endl;
        read(sock, buffer, 1024);
        cout << "Server response: " << buffer << endl;
        close(sock);
        exit(signum);
}
```

```
        int main() {
                signal(SIGINT, signalHandler);
                cout << "Running... Press Ctrl+C to interrupt." << endl;
                while (true) {
                        pause();
                }
                return 0;
        }
```

Server:

```
rps@rps-virtual-machine:~$ vim server2.cpp
rps@rps-virtual-machine:~$ make server2
g++      server2.cpp    -o server2
rps@rps-virtual-machine:~$ ./server2
Server listening on port 8080
Recieved message:  User pressed Ctrl+C
^C
```

```
rps@rps-virtual-machine:~$ cat server2.cpp
#include <iostream>
#include <unistd.h>
#include <cstring>
#include <arpa/inet.h>
using namespace std;
int main() {
        int server_fd, new_socket;
        struct sockaddr_in address;
        int opt = 1;
        int addrlen = sizeof(address);
        char buffer[1024] = {0};
        const char *response = "Message Recieved";
        //Creating socket file descriptor
        if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
                cerr << "Socket creation error" << endl;
                exit(EXIT_FAILURE);
        }
        if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEPORT, &opt, sizeof(opt))) {
                cerr << "setsocketopt error" << endl;
                exit(EXIT_FAILURE);
        }
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons(8080);
        //Forcefully attaching socket to the port 8080
        if (bind (server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
                cerr << "bind failed" << endl;
                exit(EXIT_FAILURE);
        }
         if (listen(server_fd, 3) < 0 ) {
                cerr << "listen error" << endl;
                exit(EXIT_FAILURE);
        }
         cout << "Server listening on port 8080" << endl;
```

```
        while (true) {
                if ((new_socket = accept(server_fd, (struct sockaddr *) &address, (socklen_t *)&addrlen)) < 0) {
                        cerr << "Accept error" << endl;
                        exit(EXIT_FAILURE);
        }
        read(new_socket, buffer, 1024);
        cout << "Recieved message:  " << buffer << endl;
        send(new_socket, response, strlen(response), 0);
        close(new_socket);
         }
        return 0;
}
```

write a Code where when user put ctrl z

Message delivered to server :

Server:

```
rps@rps-virtual-machine:~$ vim server3.cpp
rps@rps-virtual-machine:~$ make server3
g++      server3.cpp    -o server3
rps@rps-virtual-machine:~$ ./server3
Server listening on port
Recieved message:  User pressed Ctrl+C
^Z
[3]+  Stopped                 ./server3
```

```
rps@rps-virtual-machine:~$ cat server3.cpp
#include <iostream>
#include <unistd.h>
#include <cstring>
#include <arpa/inet.h>
#define PORT 8080
using namespace std;
int main() {
        int server_fd, new_socket;
        struct sockaddr_in address;
        int opt = 1;
        int addrlen = sizeof(address);
        char buffer[1024] = {0};
        const char *response = "Message Recieved";
        //Creating socket file descriptor
        if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
                cerr << "Socket creation error" << endl;
                exit(EXIT_FAILURE);
        }
        if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEPORT, &opt, sizeof(opt))) {
                cerr << "setsocketopt error" << endl;
                exit(EXIT_FAILURE);
        }
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons(PORT);
        //Forcefully attaching socket to the port 8080
        if (bind (server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
                cerr << "bind failed" << endl;
                exit(EXIT_FAILURE);
        }
         if (listen(server_fd, 3) < 0 ) {
                cerr << "listen error" << endl;
                exit(EXIT_FAILURE);
        }
         cout << "Server listening on port " << endl;
```

```
         while (true) {
                if ((new_socket = accept(server_fd, (struct sockaddr *) &address, (socklen_t *)&addrlen)) < 0) {
                        cerr << "Accept error" << endl;
                        exit(EXIT_FAILURE);
       }
       read(new_socket, buffer, 1024);
       cout << "Recieved message:  " << buffer << endl;
       send(new_socket, response, strlen(response), 0);
       close(new_socket);
        }
       return 0;
}
```

Client:

```
rps@rps-virtual-machine:~$ vim client3.cpp
rps@rps-virtual-machine:~$ make client3
g++      client3.cpp    -o client3
rps@rps-virtual-machine:~$ ./client3
Running... Press Ctrl+Z to interrupt.
^ZCtrl+C pressed! Sending message to server...
Message sent.Exiting.
Server response: Message Recieved

[5]+  Stopped                 ./client3
```

```
rps@rps-virtual-machine:~$ cat client3.cpp
#include <iostream>
#include <csignal>
#include <unistd.h>
#include <cstring>
#include <arpa/inet.h>
#define PORT 8080
using namespace std;
void signalHandler(int signum) {
        if(signum == SIGTSTP) {
        cout << "Ctrl+C pressed! Sending message to server..." << endl;
        int sock = 0;
        struct sockaddr_in serv_addr;
        const char* message = "User pressed Ctrl+C";
        char buffer[1024] = {0};
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                cerr << "Socket creation error" << endl;
                exit(EXIT_FAILURE);
        }
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(PORT);
        if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
                cerr << "Invalid address/ Address not supported" << endl;
                exit(EXIT_FAILURE);
        }
        if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
                cerr << "Connection Failed" << endl;
                exit(EXIT_FAILURE);
        }
        send(sock, message, strlen(message), 0);
        cout << "Message sent.Exiting." << endl;
        read(sock, buffer, 1024);
        cout << "Server response: " << buffer << endl;
        close(sock);
        raise(SIGSTOP);
        }
}
```

```
int main() {
        signal (SIGTSTP, signalHandler);
        cout << "Running... Press Ctrl+Z to interrupt." << endl;
        while (true) {
                pause();
        }
        return 0;
}


rps@rps-virtual-machine:~$
```

Objective:

Create a C++ application that combines signal handling and socket programming to manage network communication while gracefully handling interruptions (e.g., SIGINT for program termination). The application should be capable of sending and receiving messages over a network while responding appropriately to system signals.

Requirements:

Socket Programming:

Implement a TCP server that listens for incoming connections on a specified port.

Implement a TCP client that connects to the server and exchanges messages.

Signal Handling:

Implement signal handlers for SIGINT (Ctrl+C) and SIGTERM to gracefully shut down the server and client.

Ensure that the program can handle interruptions without crashing or leaving resources unfreed.

Data Exchange:

The client should be able to send a message to the server.

The server should echo the received message back to the client.

Graceful Shutdown:

When the server receives a SIGINT or SIGTERM signal, it should close all active connections and free resources before terminating.

When the client receives a SIGINT or SIGTERM signal, it should inform the server before terminating.

Server:

```
rps@rps-virtual-machine:~$ vim server4.cpp
rps@rps-virtual-machine:~$ make server4
g++      server4.cpp    -o server4
rps@rps-virtual-machine:~$ ./server4
Server listening on port 8080
^C
Signal (2) received. Shutting down server gracefully...
```

```
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cstring>
#include <csignal>

#define PORT 8080

int server_fd, new_socket;
bool running = true;

void signalHandler(int signum) {
    std::cout << "\nSignal (" << signum << ") received. Shutting down server gracefully...\n";
    running = false;
    if (new_socket >= 0) {
        close(new_socket);
    }
    if (server_fd >= 0) {
        close(server_fd);
    }
    exit(signum);
}
```

```
int main() {
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    const char *hello = "Hello from server";

    signal(SIGINT, signalHandler);
    signal(SIGTERM, signalHandler);

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Binding socket to the port 8080
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }
```

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

    while (running) {
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
            if (running) {
                perror("accept");
            }
            continue;
        }
        read(new_socket, buffer, 1024);
        std::cout << "Message from client: " << buffer << std::endl;
        send(new_socket, hello, strlen(hello), 0);
        std::cout << "Hello message sent\n";
        close(new_socket);
        new_socket = -1;
    }

    close(server_fd);
    std::cout << "Server shutdown complete.\n";
    return 0;
}
rps@rps-virtual-machine:~$
```

Client:

```
rps@rps-virtual-machine:~$ vim client4.cpp
rps@rps-virtual-machine:~$ make client4
g++      client4.cpp    -o client4
rps@rps-virtual-machine:~$ ./client4
Message sent
^C
Signal (2) received. Shutting down client gracefully...
```

```
rps@rps-virtual-machine:~$ cat client4.cpp
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>
#include <csignal>

#define PORT 8080

int sock = 0;
bool running = true;

void signalHandler(int signum) {
    std::cout << "\nSignal (" << signum << ") received. Shutting down client gracefully...\n";
    running = false;
    if (sock >= 0) {
        send(sock, "Client shutting down", 20, 0);
        close(sock);
    }
    exit(signum);
}

int main() {
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    const char *message = "Hello from client";

    signal(SIGINT, signalHandler);
    signal(SIGTERM, signalHandler);

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "\nSocket creation error\n";
        return -1;
    }
```

```
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        std::cerr << "\nInvalid address/ Address not supported\n";
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "\nConnection Failed\n";
        return -1;
    }

    send(sock, message, strlen(message), 0);
    std::cout << "Message sent\n";
    read(sock, buffer, 1024);
    std::cout << "Message from server: " << buffer << std::endl;

    while (running) {
        // Keep the client running to handle the signal
        pause();
    }

    close(sock);
    std::cout << "Client shutdown complete.\n";
    return 0;
}
rps@rps-virtual-machine:~$
```

CP Echo Server:

Implement a TCP server that:

Binds to port 9090.

Listens for incoming connections.

Accepts a connection from a client.

Receives a message from the client and echoes the same message back to the client.

Closes the connection and terminates.

Server:

```
rps@rps-virtual-machine:~$ vim server5.cpp
rps@rps-virtual-machine:~$ make server5
g++      server5.cpp    -o server5
rps@rps-virtual-machine:~$ ./server5
Waiting for connections on port 9090...
Client connected
Received message: Hello, Server!
Client connection closed
Server terminated
```

```
rps@rps-virtual-machine:~$ cat server5.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cerrno>

int main() {
    int server_fd, client_fd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    char buffer[1024] = {0};

    // Create socket
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Set up server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(9090);

    // Bind socket
    if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("Bind failed");
        close(server_fd);
        return 1;
    }
```

```
    // Listen for incoming connections
    if (listen(server_fd, 1) < 0) {
        perror("Listen failed");
        close(server_fd);
        return 1;
    }

    std::cout << "Waiting for connections on port 9090...\n";

    // Accept a connection
    client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &addr_len);
    if (client_fd < 0) {
        perror("Accept failed");
        close(server_fd);
        return 1;
    }

    std::cout << "Client connected\n";

    // Receive message from client
    ssize_t valread = read(client_fd, buffer, sizeof(buffer) - 1);
    if (valread < 0) {
        perror("Read failed");
        close(client_fd);
        close(server_fd);
        return 1;
    }

    // Null-terminate the buffer
    buffer[valread] = '\0';
    std::cout << "Received message: " << buffer << std::endl;
```

```
    // Echo the message back to the client
    if (send(client_fd, buffer, valread, 0) < 0) {
        perror("Send failed");
        close(client_fd);
        close(server_fd);
        return 1;
    }

    // Close the client connection
    close(client_fd);
    std::cout << "Client connection closed\n";

    // Close the server socket
    close(server_fd);
    std::cout << "Server terminated\n";

    return 0;
}
rps@rps-virtual-machine:~$
```

Client:

```
rps@rps-virtual-machine:~$ vim client5.cpp
rps@rps-virtual-machine:~$ make client5
g++     client5.cpp    -o client5
rps@rps-virtual-machine:~$ ./client5
Message sent to server: Hello, Server!
Server echoed: Hello, Server!
```

```
rps@rps-virtual-machine:~$ cat client5.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cerrno>
#include <arpa/inet.h>  // For inet_pton

int main() {
    int sock_fd;
    struct sockaddr_in server_addr;
    char buffer[1024] = {0};
    const char *message = "Hello, Server!";

    // Create socket
    sock_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (sock_fd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Set up server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(9090);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
        perror("Invalid address or Address not supported");
        close(sock_fd);
        return 1;
    }
```

```
    // Connect to server
    if (connect(sock_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        close(sock_fd);
        return 1;
    }

    // Send message to server
    if (send(sock_fd, message, strlen(message), 0) < 0) {
        perror("Send failed");
        close(sock_fd);
        return 1;
    }

    std::cout << "Message sent to server: " << message << std::endl;

    // Receive and print echoed message
    ssize_t valread = read(sock_fd, buffer, sizeof(buffer) - 1);
    if (valread < 0) {
        perror("Read failed");
        close(sock_fd);
        return 1;
    }

    // Null-terminate the buffer and print it
    buffer[valread] = '\0';
    std::cout << "Server echoed: " << buffer << std::endl;

    // Close socket
    close(sock_fd);

    return 0;
}
rps@rps-virtual-machine:~$
```

Create a UDP server that:

Binds to port 7070.

Receives a message from a client.

Sends a response message back to the client.

Closes the socket and terminates.

Create a UDP client that:

Sends a message to the server on port 7070.

Receives and prints the response message from the server.

Closes the socket and terminates.

Server:

```
rps@rps-virtual-machine:~$ vim server6.cpp
rps@rps-virtual-machine:~$ make server6
g++     server6.cpp   -o server6
rps@rps-virtual-machine:~$ ./server6
UDP Server is listening on port 7070...
Received message from client 127.0.0.1:60662: Hello UDP Server
Response sent back to client
```

```
rps@rps-virtual-machine:~$ cat server6.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>

using namespace std;

void udp_server() {
    int server_port = 7070;

    // Create UDP socket
    int server_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (server_socket == -1) {
        cerr << "Error creating socket\n";
        return;
    }

    // Bind to port
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(server_port);

    if (bind(server_socket, (struct sockaddr *) &server_address, sizeof(server_address)) == -1) {
        cerr << "Error binding to port " << server_port << endl;
        close(server_socket);
        return;
    }
```

```
    cout << "UDP Server is listening on port " << server_port << "...\n";

    // Receive message from client
    char buffer[1024];
    struct sockaddr_in client_address;
    socklen_t client_address_len = sizeof(client_address);
    int bytes_received = recvfrom(server_socket, buffer, sizeof(buffer), 0,
                                  (struct sockaddr *) &client_address, &client_address_len);
    if (bytes_received == -1) {
        cerr << "Error receiving data from client\n";
        close(server_socket);
        return;
    }

    buffer[bytes_received] = '\0'; // Null-terminate the received data
    cout << "Received message from client " << inet_ntoa(client_address.sin_addr)
         << ":" << ntohs(client_address.sin_port) << ": " << buffer << endl;

    // Send response back to client
    sendto(server_socket, buffer, bytes_received, 0,
           (struct sockaddr *) &client_address, client_address_len);
    cout << "Response sent back to client\n";

    // Close server socket
    close(server_socket);
}

int main() {
    udp_server();
    return 0;
}

rps@rps-virtual-machine:~$
```

Client:

```
rps@rps-virtual-machine:~$ vim client6.cpp
rps@rps-virtual-machine:~$ make client6
g++     client6.cpp   -o client6
rps@rps-virtual-machine:~$ ./client6
Message sent to UDP server
Received response from server: Hello UDP Server
```

```
rps@rps-virtual-machine:~$ cat client6.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

using namespace std;

void udp_client() {
    int client_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (client_socket == -1) {
        cerr << "Error creating socket\n";
        return;
    }

    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(7070);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {
        cerr << "Invalid address or Address not supported\n";
        close(client_socket);
        return;
    }

    const char *message = "Hello UDP Server";
    sendto(client_socket, message, strlen(message), 0,
           (const struct sockaddr *) &server_address, sizeof(server_address));
    cout << "Message sent to UDP server\n";
```

```
    char buffer[1024];
    socklen_t server_address_len = sizeof(server_address);
    int bytes_received = recvfrom(client_socket, buffer, sizeof(buffer), 0,
                                  (struct sockaddr *) &server_address, &server_address_len);
    if (bytes_received == -1) {
        cerr << "Error receiving response from server\n";
        close(client_socket);
        return;
    }

    buffer[bytes_received] = '\0'; // Null-terminate the received data
    cout << "Received response from server: " << buffer << endl;

    // Close client socket
    close(client_socket);
}

int main() {
    udp_client();
    return 0;
}

rps@rps-virtual-machine:~$
```

Create a TCP client that:

Connects to a server at port 5050.

Sends a message to the server.

Handles and displays error messages for common issues such as connection failure or data transmission errors.

Receives and prints the response message from the server.

Closes the socket and terminates.

Server:

```
rps@rps-virtual-machine:~$ vim server8.cpp
rps@rps-virtual-machine:~$ make server8
g++     server8.cpp   -o server8
rps@rps-virtual-machine:~$ ./server8
Server is listening on port 5050...
New connection accepted. Client connected from 127.0.0.1:55838
Message from client: Hello, Server!
```

```
rps@rps-virtual-machine:~$ cat server8.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <cerrno>

using namespace std;

void tcp_server() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len;
    char buffer[1024];

    // Create TCP socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Socket creation failed");
        return;
    }

    // Prepare server address
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(5050);

    // Bind socket to address
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Bind failed");
        close(server_socket);
        return;
    }
```

```
    // Listen for incoming connections
    if (listen(server_socket, 5) == -1) {
        perror("Listen failed");
        close(server_socket);
        return;
    }

    cout << "Server is listening on port 5050..." << endl;

    // Accept incoming connections
    client_len = sizeof(client_addr);
    client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_len);
    if (client_socket == -1) {
        perror("Accept failed");
        close(server_socket);
        return;
    }

    cout << "New connection accepted. Client connected from "
         << inet_ntoa(client_addr.sin_addr)
         << ":" << ntohs(client_addr.sin_port) << endl;

    // Receive message from client
    ssize_t valread = read(client_socket, buffer, sizeof(buffer) - 1);
    if (valread == -1) {
        perror("Read failed");
        close(client_socket);
        close(server_socket);
        return;
    }

    buffer[valread] = '\0';
    cout << "Message from client: " << buffer << endl;
```

```
    // Echo message back to client
    if (send(client_socket, buffer, strlen(buffer), 0) == -1) {
        perror("Send failed");
        close(client_socket);
        close(server_socket);
        return;
    }

    // Close client socket
    close(client_socket);

    // Close server socket
    close(server_socket);
}

int main() {
    tcp_server();
    return 0;
}

rps@rps-virtual-machine:~$
```

Client:

```
rps@rps-virtual-machine:~$ vim client8.cpp
rps@rps-virtual-machine:~$ make client8
g++      client8.cpp    -o client8
rps@rps-virtual-machine:~$ ./client8
Message sent to server: Hello, Server!
Server response: Hello, Server!
```

```
rps@rps-virtual-machine:~$ cat client8.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cerrno>

using namespace std;

void tcp_client() {
    // Create TCP socket
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Socket creation failed");
        return;
    }

    // Server address setup
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(5050);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {
        perror("Invalid address or Address not supported");
        close(client_socket);
        return;
    }
```

```
    // Connect to server
    if (connect(client_socket, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {
        perror("Connection failed");
        close(client_socket);
        return;
    }

    // Message to send
    const char *message = "Hello, Server!";
    if (send(client_socket, message, strlen(message), 0) < 0) {
        perror("Send failed");
        close(client_socket);
        return;
    }

    cout << "Message sent to server: " << message << endl;

    // Receive and print response from server
    char buffer[1024];
    ssize_t valread = read(client_socket, buffer, sizeof(buffer) - 1);
    if (valread < 0) {
        perror("Read failed");
        close(client_socket);
        return;
    }

    buffer[valread] = '\0'; // Null-terminate the received data
    cout << "Server response: " << buffer << endl;

    // Close socket
    close(client_socket);
}
```

```
int main() {
    tcp_client();
    return 0;
}

rps@rps-virtual-machine:~$
```

Implement a TCP server that:

Binds to port 4040.

Listens for incoming connections.

Uses select() to handle multiple client connections.

Receives a message from each client and sends a response back.

Closes the connections and terminates.

Server:

```
rps@rps-virtual-machine:~$ vim server9.cpp
rps@rps-virtual-machine:~$ make server9
g++      server9.cpp    -o server9
rps@rps-virtual-machine:~$ ./server9
Server listening on port 4040...
New connection, socket fd is 4, ip is: 127.0.0.1, port: 45066
Adding to list of sockets as 0
Host disconnected, ip: 127.0.0.1, port: 45066
^Z
[1]+  Stopped                 ./server9
```

```
rps@rps-virtual-machine:~$ cat server9.cpp
#include <iostream>
#include <cstring>     // For memset
#include <unistd.h>    // For close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/select.h>
#define PORT 4040
#define BUFFER_SIZE 1024
#define MAX_CLIENTS 10
int main() {
    int server_fd, new_socket, client_socket[MAX_CLIENTS], max_clients = MAX_CLIENTS, activity, i, valread, sd;
    int max_sd;
    struct sockaddr_in address;
    char buffer[BUFFER_SIZE];
int  addrlen = sizeof(address);
    for (i = 0; i < max_clients; i++) {
        client_socket[i] = 0;
    }    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) {
        std::cerr << "Socket creation failed\n";
        return 1;
    }    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
        std::cerr << "Bind failed\n";
        close(server_fd);
        return 1;   }
    if (listen(server_fd, 3) < 0) {
        std::cerr << "Listen failed\n";
        close(server_fd);
        return 1;    }
    std::cout << "Server listening on port " << PORT << "...\n";
    fd_set readfds;
```

```
    while (true) {
        FD_ZERO(&readfds);
        FD_SET(server_fd, &readfds);
        max_sd = server_fd;
        for (i = 0; i < max_clients; i++) {
            sd = client_socket[i];
            if (sd > 0) {
                FD_SET(sd, &readfds);
            }
            if (sd > max_sd) {
                max_sd = sd;    }
        }
        activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);
        if ((activity < 0) && (errno != EINTR)) {
            std::cerr << "Select error\n";     }
        if (FD_ISSET(server_fd, &readfds)) {
            int addrlen = sizeof(address);
            if ((new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
                std::cerr << "Accept failed\n";
                close(server_fd);
                return 1;
            }  std::cout << "New connection, socket fd is " << new_socket << ", ip is: "
                      << inet_ntoa(address.sin_addr) << ", port: " << ntohs(address.sin_port) << "\n";
            for (i = 0; i < max_clients; i++) {
                if (client_socket[i] == 0) {
                    client_socket[i] = new_socket;
                    std::cout << "Adding to list of sockets as " << i << "\n";
                    break;           }   }
        }
        for (i = 0; i < max_clients; i++) {
            sd = client_socket[i];
```

```
            sd = client_socket[i];
          if (FD_ISSET(sd, &readfds)) {
                if ((valread = read(sd, buffer, BUFFER_SIZE)) == 0) {
                   getpeername(sd, (struct sockaddr*)&address, (socklen_t*)&addrlen);
                   std::cout << "Host disconnected, ip: " << inet_ntoa(address.sin_addr) << ", port: " << ntohs(address.sin_port) << "\n";
                   close(sd);
                   client_socket[i] = 0;
               } else {
                   buffer[valread] = '\0';
                   send(sd, buffer, strlen(buffer), 0);         }
           }    }
    }   close(server_fd);
  return 0;
}
rps@rps-virtual-machine:~$
```

Client:

```
rps@rps-virtual-machine:~$ vim client9.cpp
rps@rps-virtual-machine:~$ make client9
g++      client9.cpp    -o client9
rps@rps-virtual-machine:~$ ./client9
Connected to server
Received message from server: Hello, Server!
```

```
rps@rps-virtual-machine:~$ cat client9.cpp
#include <iostream>
#include <cstring>   // For memset
#include <unistd.h>  // For close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <fcntl.h>   // For fcntl
#define SERVER_PORT 4040
#define SERVER_IP "127.0.0.1" // Localhost for testing
#define BUFFER_SIZE 1024
#define TIMEOUT_SEC 5
int main() {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        std::cerr << "Socket creation failed\n";
        return 1;    }
    int flags = fcntl(sock, F_GETFL, 0);
    fcntl(sock, F_SETFL, flags | O_NONBLOCK);
    sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(SERVER_PORT);
    inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr);
    if (connect(sock, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {
        if (errno != EINPROGRESS) {
            std::cerr << "Connection failed\n";
            close(sock);
            return 1;        }
    }
```

```
    fd_set write_fds;
    struct timeval timeout;
    int select_result;
    FD_ZERO(&write_fds);
    FD_SET(sock, &write_fds);
    timeout.tv_sec = TIMEOUT_SEC;
    timeout.tv_usec = 0;
    select_result = select(sock + 1, nullptr, &write_fds, nullptr, &timeout);
    if (select_result == -1) {
        std::cerr << "Select error\n";
        close(sock);
        return 1;
    } else if (select_result == 0) {
        std::cerr << "Connection timed out\n";
        close(sock);
        return 1;
    } else if (FD_ISSET(sock, &write_fds)) {
        std::cout << "Connected to server\n";   }
   const char* message = "Hello, Server!";
    ssize_t bytes_sent = send(sock, message, strlen(message), 0);
    if (bytes_sent < 0) {
        std::cerr << "Send failed\n";
        close(sock);
        return 1;    }
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    ssize_t bytes_received = recv(sock, buffer, BUFFER_SIZE, 0);
    if (bytes_received < 0) {
        std::cerr << "Receive failed\n";
        close(sock);
        return 1;
    } else if (bytes_received == 0) {
        std::cout << "Server closed the connection\n";
    } else {
        std::cout << "Received message from server: " << buffer << "\n";   }
    close(sock);
    return 0;
}
rps@rps-virtual-machine:~$
```

Create a TCP client that:

Connects to a server at port 3030.

Sends a message to the server.

Implements a timeout mechanism to handle cases where the server does not respond within a specified time.

Receives and prints the response message if available.

Closes the socket and terminates.

Server:

```
rps@rps-virtual-machine:~$ vim server11.cpp
rps@rps-virtual-machine:~$ make server11
g++      server11.cpp    -o server11
rps@rps-virtual-machine:~$ ./server11
Server listening on port 3030...
Client connected
Received message: Hello, Server!
Response sent to client
Server terminated
```

```
rps@rps-virtual-machine:~$ cat server11.cpp
#include <iostream>
#include <cstring>   // For memset
#include <unistd.h>  // For close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 3030
#define BUFFER_SIZE 1024

int main() {
    // Step 1: Create a socket
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) {
        std::cerr << "Socket creation failed\n";
        return 1;
    }

    // Step 2: Bind the socket to port 3030
    sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {
        std::cerr << "Bind failed\n";
        close(server_fd);
        return 1;
    }
```

```
    // Step 3: Listen for incoming connections
    if (listen(server_fd, 5) < 0) {
        std::cerr << "Listen failed\n";
        close(server_fd);
        return 1;
    }

    std::cout << "Server listening on port " << PORT << "...\n";

    // Step 4: Accept a connection
    sockaddr_in client_address;
    socklen_t client_address_len = sizeof(client_address);
    int client_fd = accept(server_fd, (struct sockaddr*)&client_address, &client_address_len);
    if (client_fd < 0) {
        std::cerr << "Accept failed\n";
        close(server_fd);
        return 1;
    }

    std::cout << "Client connected\n";

    // Step 5: Receive a message from the client
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);
    ssize_t bytes_received = recv(client_fd, buffer, BUFFER_SIZE, 0);
    if (bytes_received < 0) {
        std::cerr << "Receive failed\n";
        close(client_fd);
        close(server_fd);
        return 1;
    } else if (bytes_received == 0) {
        std::cout << "Client disconnected\n";
    } else {
        std::cout << "Received message: " << buffer << "\n";
```

```
        // Step 6: Send a response back to the client
        const char* response = "Hello, Client!";
        ssize_t bytes_sent = send(client_fd, response, strlen(response), 0);
        if (bytes_sent < 0) {
            std::cerr << "Send failed\n";
            close(client_fd);
            close(server_fd);
            return 1;
        }

        std::cout << "Response sent to client\n";
    }

    // Step 7: Close the connection and terminate
    close(client_fd);
    close(server_fd);

    std::cout << "Server terminated\n";

    return 0;
}
rps@rps-virtual-machine:~$
```

Client:

```
rps@rps-virtual-machine:~$ vim client11.cpp
rps@rps-virtual-machine:~$ make client11
g++      client11.cpp    -o client11
rps@rps-virtual-machine:~$ ./client11
Connected to server
Received message from server: Hello, Client!
```

```
rps@rps-virtual-machine:~$ cat client11.cpp
#include <iostream>
#include <cstring>   // For memset
#include <unistd.h>  // For close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <fcntl.h>   // For fcntl

#define SERVER_PORT 3030
#define SERVER_IP "127.0.0.1" // Localhost for testing
#define BUFFER_SIZE 1024
#define TIMEOUT_SEC 5

int main() {
    // Step 1: Create a socket
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        std::cerr << "Socket creation failed\n";
        return 1;
    }

    // Set socket to non-blocking
    int flags = fcntl(sock, F_GETFL, 0);
    fcntl(sock, F_SETFL, flags | O_NONBLOCK);

    // Step 2: Connect to the server
    sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(SERVER_PORT);
    inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr);
```

```
    if (connect(sock, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {
        if (errno != EINPROGRESS) {
            std::cerr << "Connection failed\n";
            close(sock);
            return 1;
        }
    }

    // Step 3: Implement a timeout mechanism
    fd_set write_fds;
    struct timeval timeout;
    int select_result;

    FD_ZERO(&write_fds);
    FD_SET(sock, &write_fds);

    timeout.tv_sec = TIMEOUT_SEC;
    timeout.tv_usec = 0;

    select_result = select(sock + 1, nullptr, &write_fds, nullptr, &timeout);
    if (select_result == -1) {
        std::cerr << "Select error\n";
        close(sock);
        return 1;
    } else if (select_result == 0) {
        std::cerr << "Connection timed out\n";
        close(sock);
        return 1;
    } else if (FD_ISSET(sock, &write_fds)) {
        std::cout << "Connected to server\n";
    }
```

```
    // Step 4: Send a message to the server
    const char* message = "Hello, Server!";
    ssize_t bytes_sent = send(sock, message, strlen(message), 0);
    if (bytes_sent < 0) {
        std::cerr << "Send failed\n";
        close(sock);
        return 1;
    }

    // Step 5: Receive and print the response
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, BUFFER_SIZE);

    ssize_t bytes_received = recv(sock, buffer, BUFFER_SIZE, 0);
    if (bytes_received < 0) {
        std::cerr << "Receive failed\n";
        close(sock);
        return 1;
    } else if (bytes_received == 0) {
        std::cout << "Server closed the connection\n";
    } else {
        std::cout << "Received message from server: " << buffer << "\n";
    }

    // Step 6: Close the socket and terminate
    close(sock);

    return 0;
}
rps@rps-virtual-machine:~$
```

Implement a TCP server that:

Binds to port 6060.

Listens for incoming connections.

Accepts multiple client connections concurrently.

Receives a message from each client and sends a unique response back.

Closes the connections and terminates after handling all clients.

Server:

```
rps@rps-virtual-machine:~$ vim server12.cpp
rps@rps-virtual-machine:~$ make server12
g++      server12.cpp    -o server12
rps@rps-virtual-machine:~$ ./server12
Server is running on port 6060
Client connected
^C
```

```
rps@rps-virtual-machine:~$ cat server12.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <vector>
#include <thread>
#include <atomic>

#define PORT 6060
#define MAX_CLIENTS 3

// Define unique responses for clients
const std::vector<std::string> client_responses = {
    "Hi from client 1",
    "Hello from client 2",
    "Greetings from client 3"
};

std::atomic<int> client_counter(0);  // Atomic counter to ensure uniqueness

void handle_client(int client_fd) {
    char buffer[1024] = {0};
    std::string response;

    // Receive message from client
    ssize_t valread = read(client_fd, buffer, sizeof(buffer) - 1);
    if (valread > 0) {
        buffer[valread] = '\0';  // Null-terminate the received message
```

```
        // Generate unique response based on client id
        int id = client_counter.fetch_add(1);  // Get and increment the counter
        if (id < client_responses.size()) {
            response = client_responses[id];
        } else {
            response = "Hello from server!";
        }

        // Send unique response to the client
        if (send(client_fd, response.c_str(), response.length(), 0) < 0) {
            perror("Send failed");
        }
    } else {
        perror("Read failed");
    }

    // Close client connection
    close(client_fd);
}
```

```
int main() {
    int server_fd, client_fd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    std::vector<int> client_sockets;

    // Create socket
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Set up server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind socket
    if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("Bind failed");
        close(server_fd);
        return 1;
    }

    // Listen for connections
    if (listen(server_fd, MAX_CLIENTS) < 0) {
        perror("Listen failed");
        close(server_fd);
        return 1;
    }

    std::cout << "Server is running on port " << PORT << "\n";
```

```
    while (true) {
        // Accept new connections if the maximum number of clients hasn't been reached
        if (client_sockets.size() < MAX_CLIENTS) {
            client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &addr_len);
            if (client_fd < 0) {
                perror("Accept failed");
                continue;
            }

            std::cout << "Client connected\n";
            client_sockets.push_back(client_fd);

            // Handle client in a separate thread
            std::thread(handle_client, client_fd).detach();
        }

        // Exit if no clients are connected (optional, remove if you want to keep server running)
        if (client_sockets.empty()) {
            break;
        }
    }

    // Close server socket
    close(server_fd);

    return 0;
}
rps@rps-virtual-machine:~$
```

Client:

```
rps@rps-virtual-machine:~$ vim client12.cpp
rps@rps-virtual-machine:~$ vim client12.cpp
rps@rps-virtual-machine:~$ make client12
g++      client12.cpp    -o client12
rps@rps-virtual-machine:~$ ./client12
Server response: Hi from client 1
```

```
rps@rps-virtual-machine:~$ cat client12.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 6060
#define SERVER_ADDRESS "127.0.0.1"

int main() {
    int sock_fd;
    struct sockaddr_in server_addr;
    const char *message = "Hello, Server!";
    char buffer[1024] = {0};

    // Create socket
    sock_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (sock_fd < 0) {
        std::cerr << "Socket creation failed: " << strerror(errno) << std::endl;
        return 1;
    }

    // Set up server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr(SERVER_ADDRESS);

    // Connect to server
    if (connect(sock_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        std::cerr << "Connection failed: " << strerror(errno) << std::endl;
        close(sock_fd);
        return 1;
    }
```

```
    // Send message to server
    if (send(sock_fd, message, strlen(message), 0) < 0) {
        std::cerr << "Failed to send message: " << strerror(errno) << std::endl;
        close(sock_fd);
        return 1;
    }

    // Receive and print server response
    ssize_t valread = read(sock_fd, buffer, sizeof(buffer) - 1);
    if (valread < 0) {
        std::cerr << "Failed to receive message: " << strerror(errno) << std::endl;
        close(sock_fd);
        return 1;
    }
    buffer[valread] = '\0';
    std::cout << "Server response: " << buffer << std::endl;

    // Close socket
    close(sock_fd);

    return 0;
}
rps@rps-virtual-machine:~$
```

TCP Server with Custom Protocol:

Implement a TCP server that:

Binds to port 2020.

Listens for incoming connections.

Implements a simple custom protocol where:

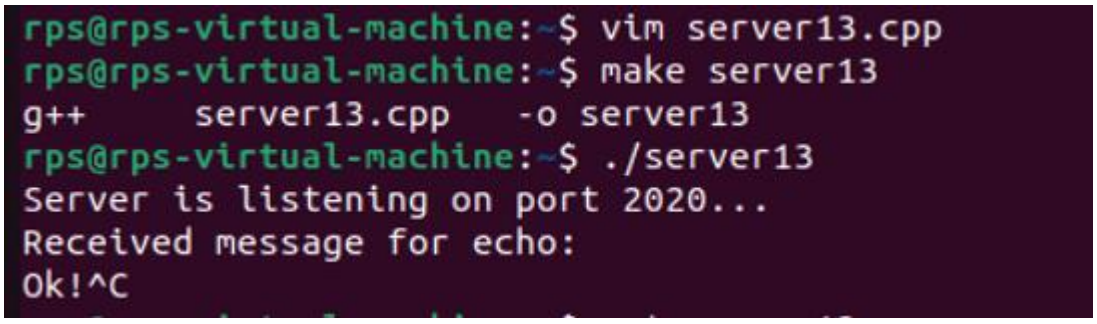
The first byte of the message indicates the type of operation (e.g., 1 for echo, 2 for reverse).

For operation type 1, the server echoes the message back.

For operation type 2, the server sends back the reversed message.

Closes the connection and terminates.

Server:



```
rps@rps-virtual-machine:~$ cat server13.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cerrno>
#include <arpa/inet.h>

using namespace std;

const int PORT = 2020;

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

    // Read the first byte to determine the operation
    valread = read(client_socket, buffer, 1);
    if (valread <= 0) {
        perror("Read failed");
        close(client_socket);
        return;
    }

    int operation = buffer[0];

    if (operation == 1) { // Echo operation
        valread = read(client_socket, buffer, sizeof(buffer) - 1);
        if (valread <= 0) {
            perror("Read failed");
            close(client_socket);
            return;
        }
        buffer[valread] = '\0'; // Null-terminate the received data
        cout << "Received message for echo: " << buffer << endl;
```

```
        // Echo the message back to the client
        if (send(client_socket, buffer, strlen(buffer), 0) == -1) {
            perror("Send failed");
            close(client_socket);
            return;
        }
    } else if (operation == 2) { // Reverse operation
        valread = read(client_socket, buffer, sizeof(buffer) - 1);
        if (valread <= 0) {
            perror("Read failed");
            close(client_socket);
            return;
        }
        buffer[valread] = '\0'; // Null-terminate the received data
        cout << "Received message for reverse: " << buffer << endl;

        // Reverse the message
        int len = strlen(buffer);
        for (int i = 0; i < len / 2; ++i) {
            char temp = buffer[i];
            buffer[i] = buffer[len - i - 1];
            buffer[len - i - 1] = temp;
        }

        // Send the reversed message back to the client
        if (send(client_socket, buffer, strlen(buffer), 0) == -1) {
            perror("Send failed");
            close(client_socket);
            return;
        }
    } else {
        cout << "Unknown operation type: " << operation << endl;
        close(client_socket);
        return;
    }
```

```
    // Close the client socket
    close(client_socket);
}

void tcp_server() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len;

    // Create TCP socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Socket creation failed");
        return;
    }

    // Prepare server address
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind socket to address
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Bind failed");
        close(server_socket);
        return;
    }

    // Listen for incoming connections
    if (listen(server_socket, 5) == -1) {
        perror("Listen failed");
        close(server_socket);
        return;
    }
```

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

    while (true) {
        // Accept incoming connection
        client_len = sizeof(client_addr);
        client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_len);
        if (client_socket == -1) {
            perror("Accept failed");
            continue;
        }

        // Handle client request
        handle_client(client_socket);
    }

    // Close server socket
    close(server_socket);
}

int main() {
    tcp_server();
    return 0;
}

rps@rps-virtual-machine:~$
```

Client:

```
rps@rps-virtual-machine:~$ vim client13.cpp
rps@rps-virtual-machine:~$ make client13
g++      client13.cpp    -o client13
rps@rps-virtual-machine:~$ ./client13
Message sent to server: Hello, Server!
Server response:
Send operation type failed: Bad file descriptor
```

```
rps@rps-virtual-machine:~$ cat client13.cpp
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cerrno>

using namespace std;

const int PORT = 2020;
const char* SERVER_IP = "127.0.0.1";

void send_operation(int client_socket, int operation, const char* message) {
    // Send operation type
    if (send(client_socket, &operation, sizeof(operation), 0) == -1) {
        perror("Send operation type failed");
        close(client_socket);
        return;
    }

    // Send message
    if (send(client_socket, message, strlen(message), 0) == -1) {
        perror("Send message failed");
        close(client_socket);
        return;
    }

    cout << "Message sent to server: " << message << endl;
```

```
    // Receive response
    char buffer[1024];
    ssize_t valread = read(client_socket, buffer, sizeof(buffer) - 1);
    if (valread < 0) {
        perror("Read failed");
        close(client_socket);
        return;
    }

    buffer[valread] = '\0'; // Null-terminate the received data
    cout << "Server response: " << buffer << endl;

    // Close socket
    close(client_socket);
}

void tcp_client() {
    // Create TCP socket
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Socket creation failed");
        return;
    }

    // Server address setup
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);
```

```
    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr) <= 0) {
        perror("Invalid address or Address not supported");
        close(client_socket);
        return;
    }

    // Connect to server
    if (connect(client_socket, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {
        perror("Connection failed");
        close(client_socket);
        return;
    }

    // Test operations
    const char* message1 = "Hello, Server!";
    send_operation(client_socket, 1, message1);

    const char* message2 = "Reverse this message";
    send_operation(client_socket, 2, message2);
}

int main() {
    tcp_client();
    return 0;
}

rps@rps-virtual-machine:~$
```