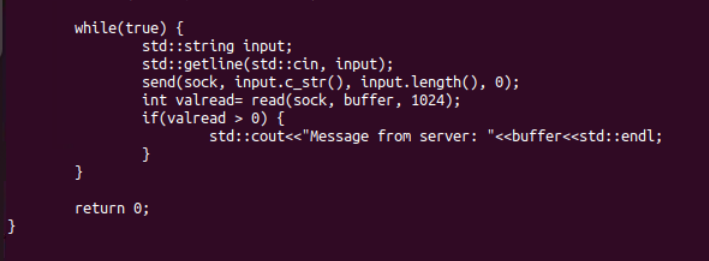
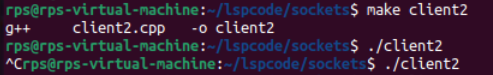


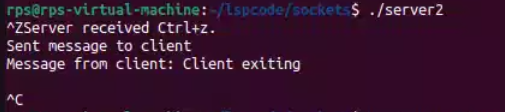
Client code:





Output:





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.

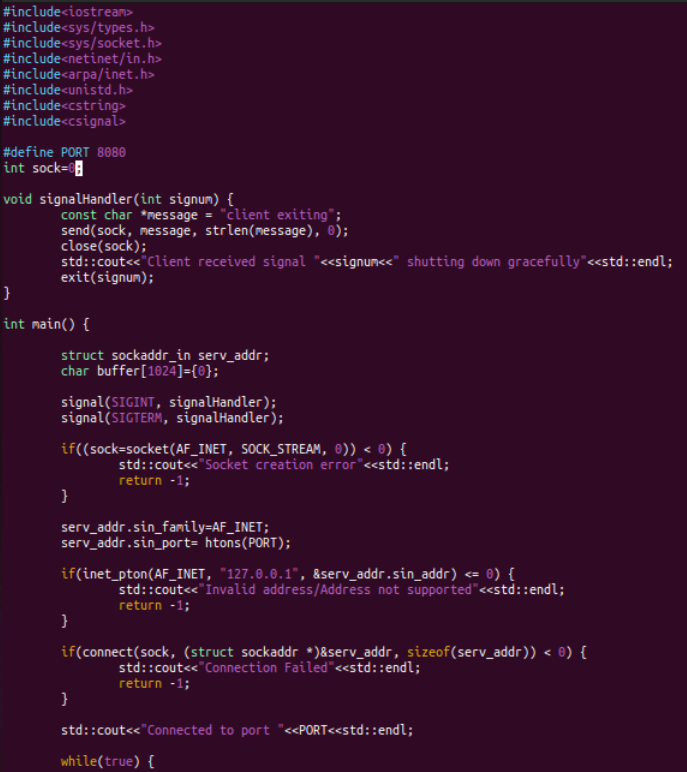
When the client receives a SIGINT or SIGTERM signal, it should inform the server before terminating.

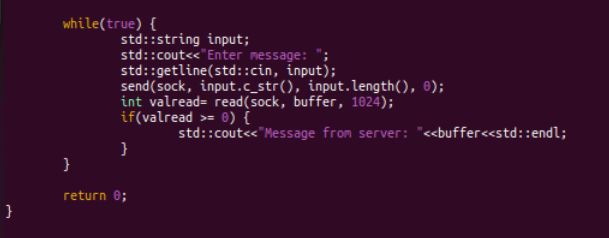
Server code:





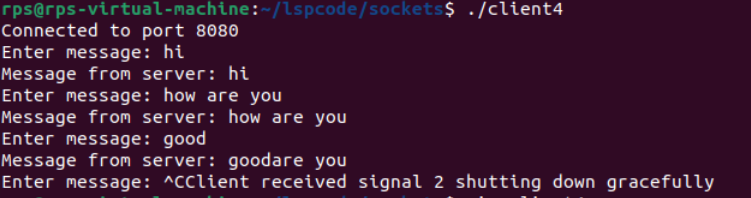
Client code:

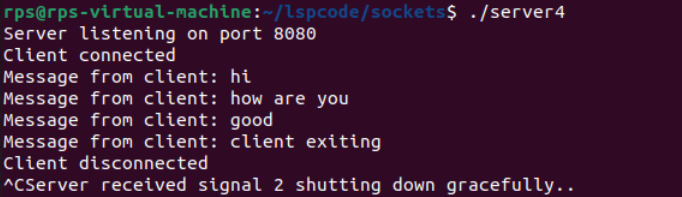






Output:





CP Echo Server:

Implement a TCP server that:

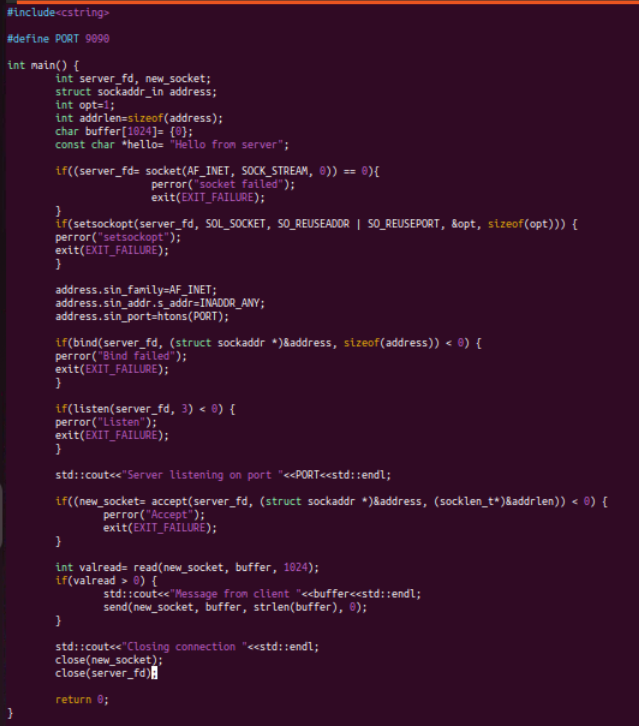
Binds to port 9090.

Listens for incoming connections.

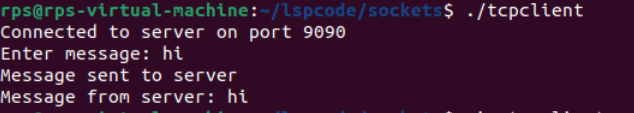
Accepts a connection from a client.

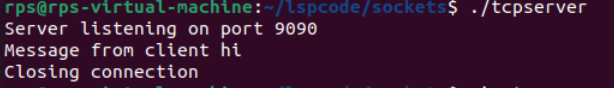
Receives a message from the client and echoes the same message back to the client.

Closes the connection and terminates.









Create a TCP client that:

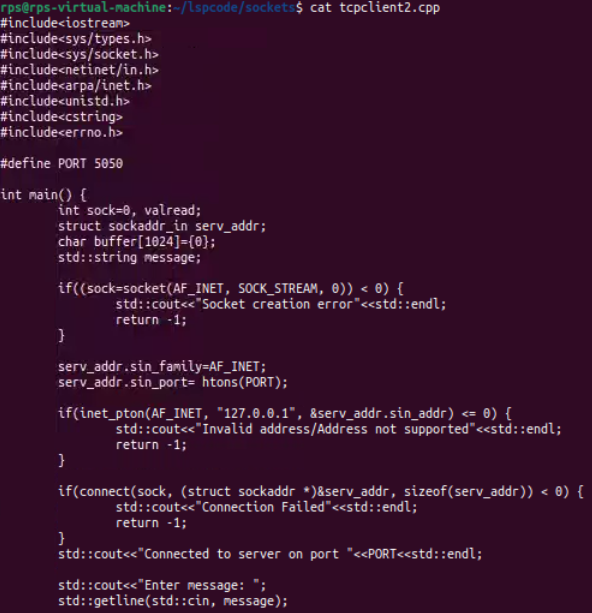
Connects to a server at port 5050.

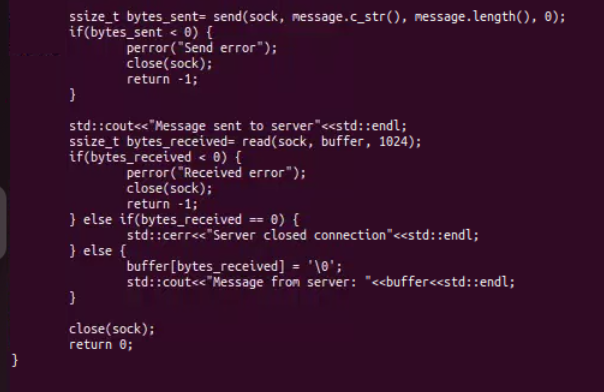
Sends a message to the server.

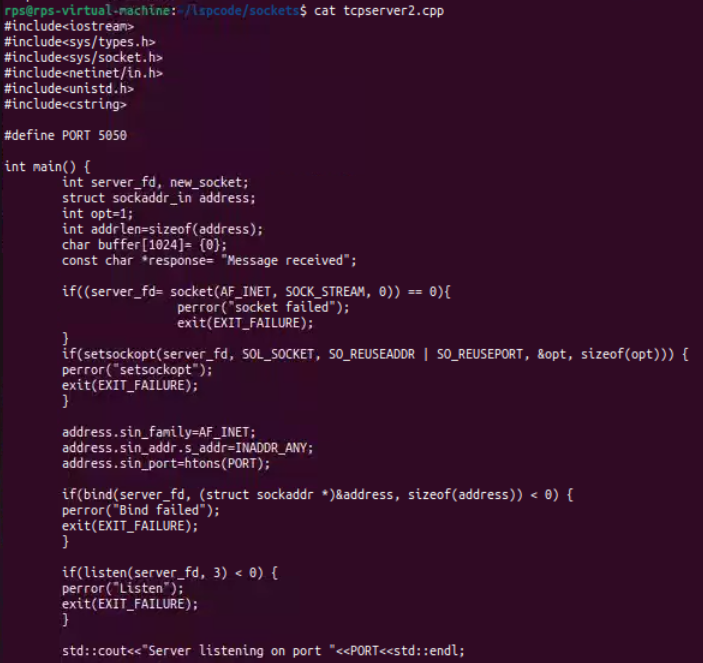
Handles and displays error messages for common issues such as connection failure or data transmission errors.

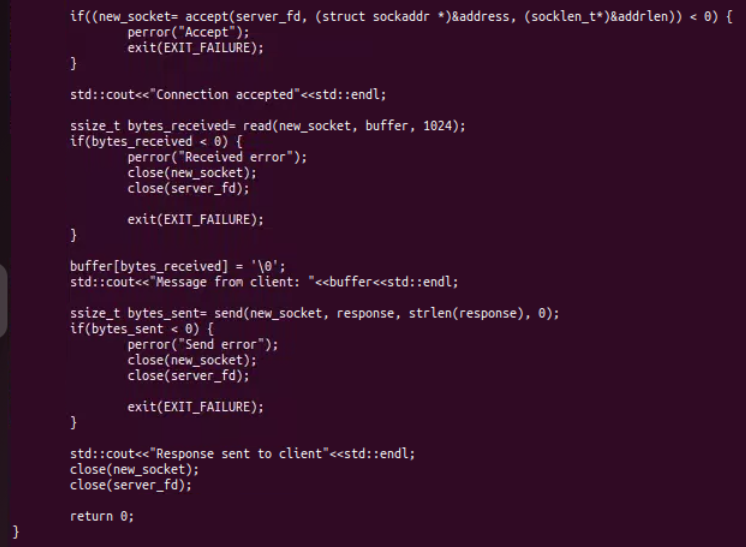
Receives and prints the response message from the server.

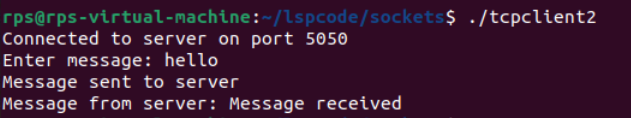
Closes the socket and terminates.

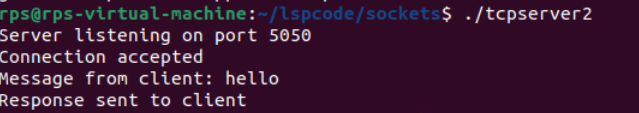












UDP Client-Server Communication:

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.

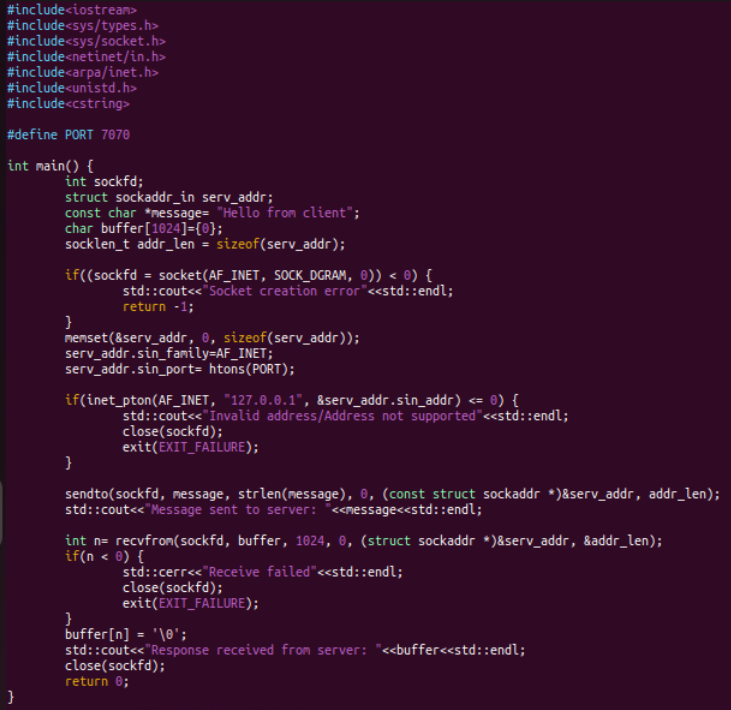
Receives and prints the response message from the server.

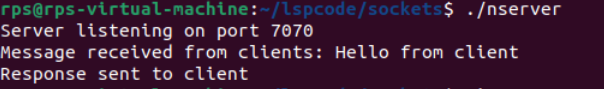
Closes the socket and terminates.

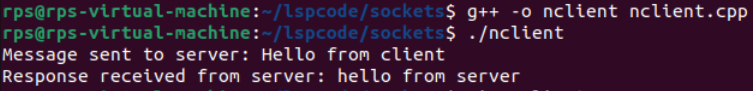
Server:



Client:







TCP Server with Custom Protocol:

Implement a TCP server that:

Binds to port 2020.

Listens for incoming connections.

Implements a simple custom protocol where:

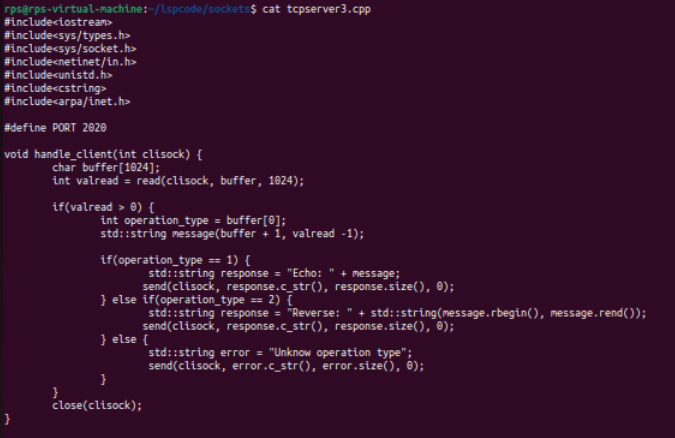
The first byte of the message indicates the type of operation (e.g., 1 for echo, 2 for reverse).

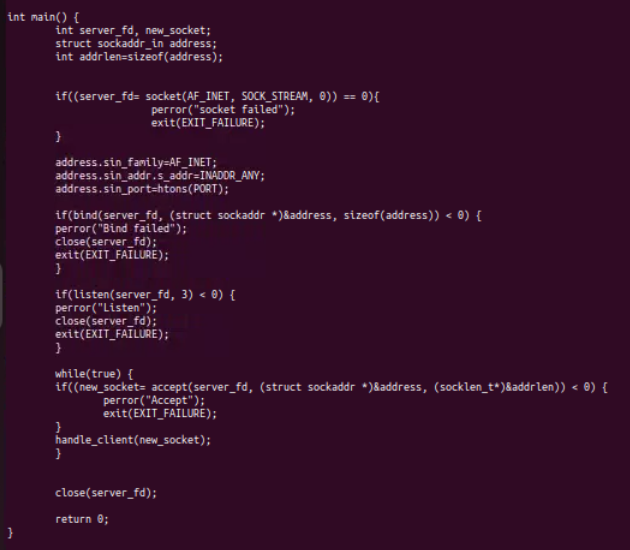
For operation type 1, the server echoes the message back.

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

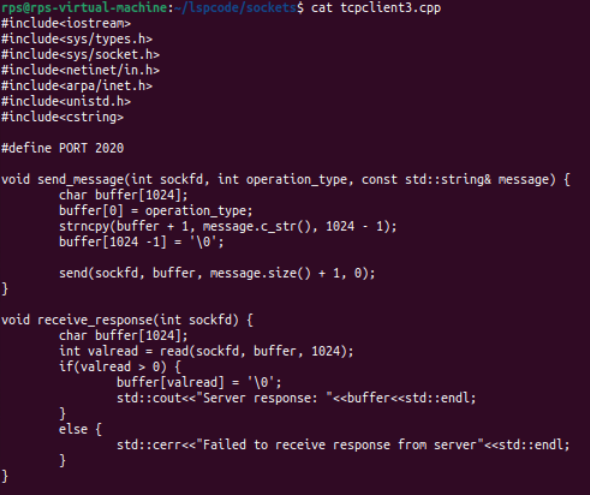
Closes the connection and terminates.

Server:





Client:





Output:

