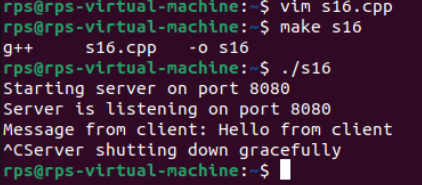
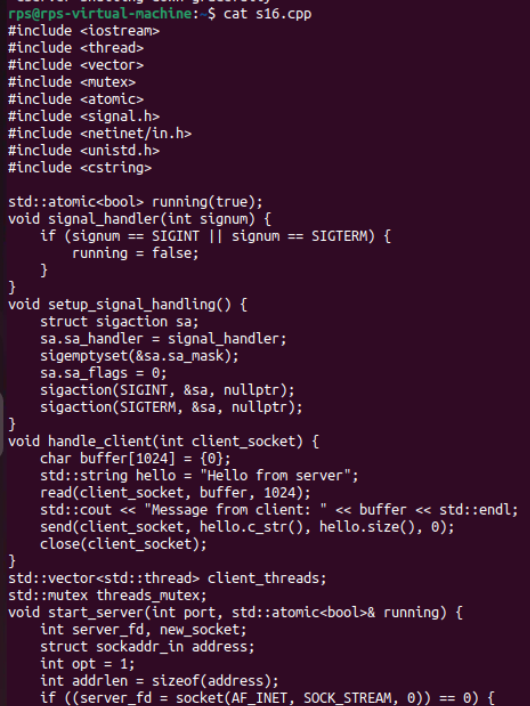
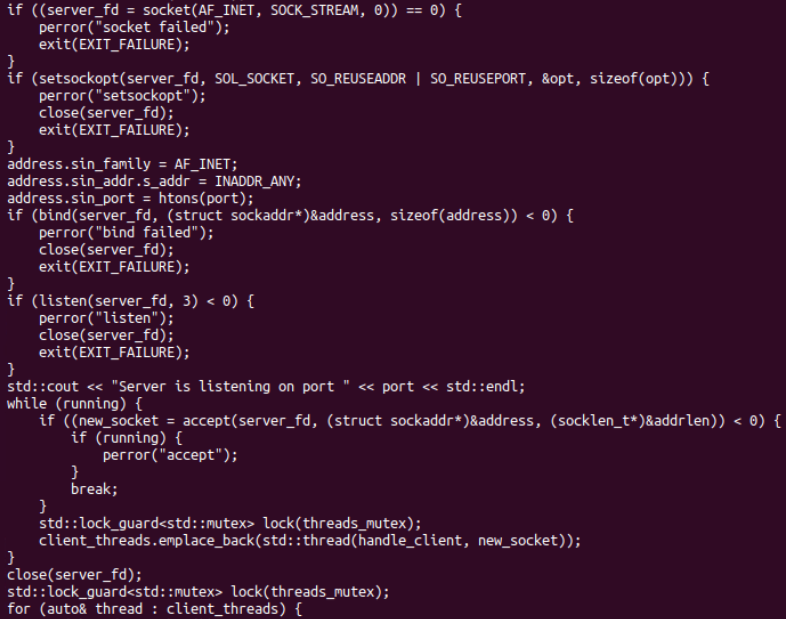
**Design and implement a network service that reliably handles concurrent client connections while ensuring graceful termination in response to external signals (e.g., SIGTERM, SIGINT). The service must maintain data consistency and avoid resource leaks throughout its lifecycle.**

**Server:**

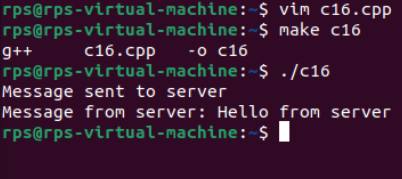


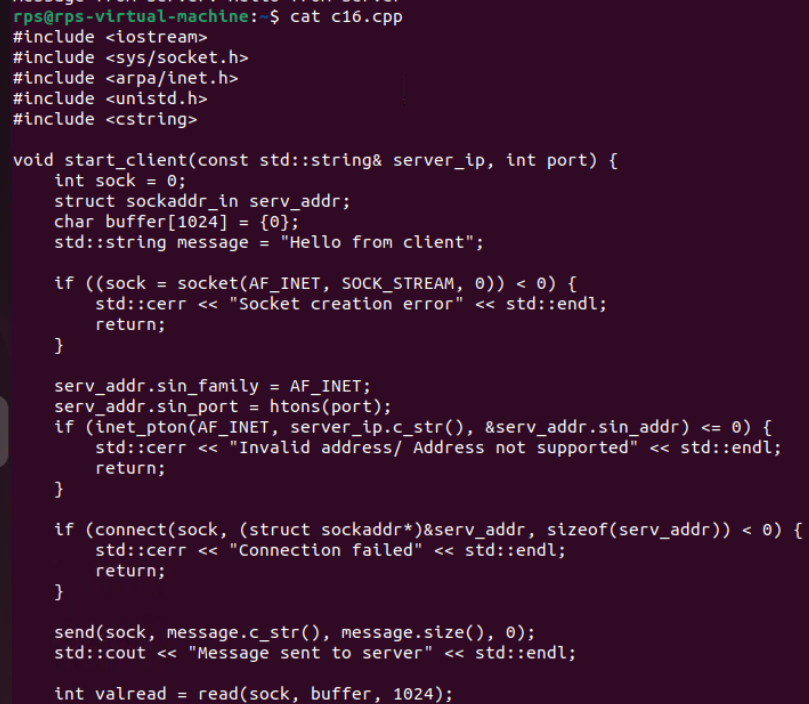






**Client:**

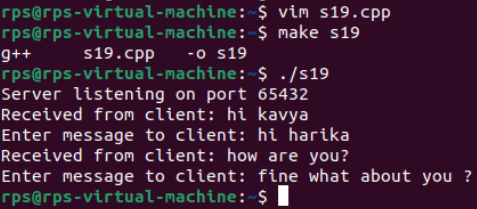


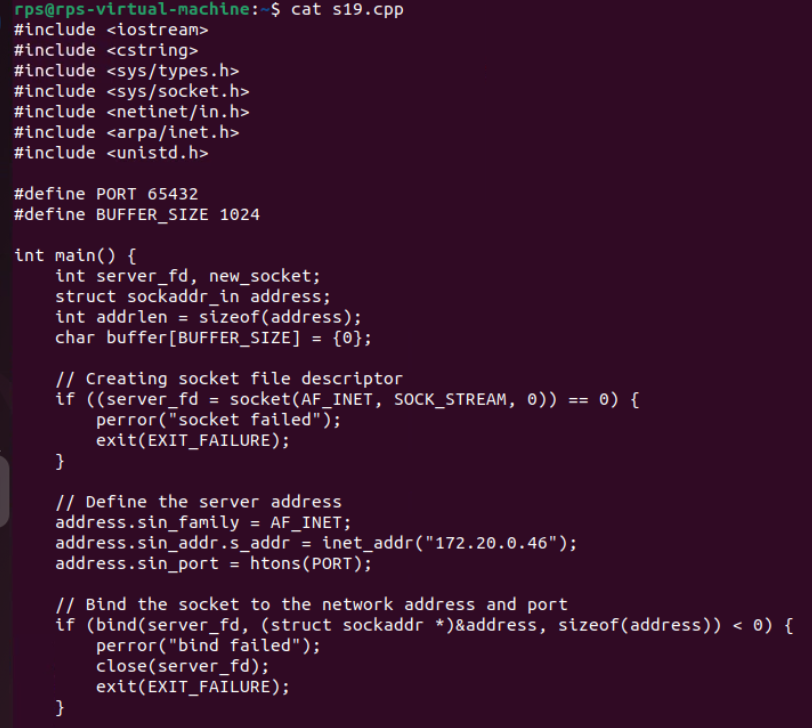


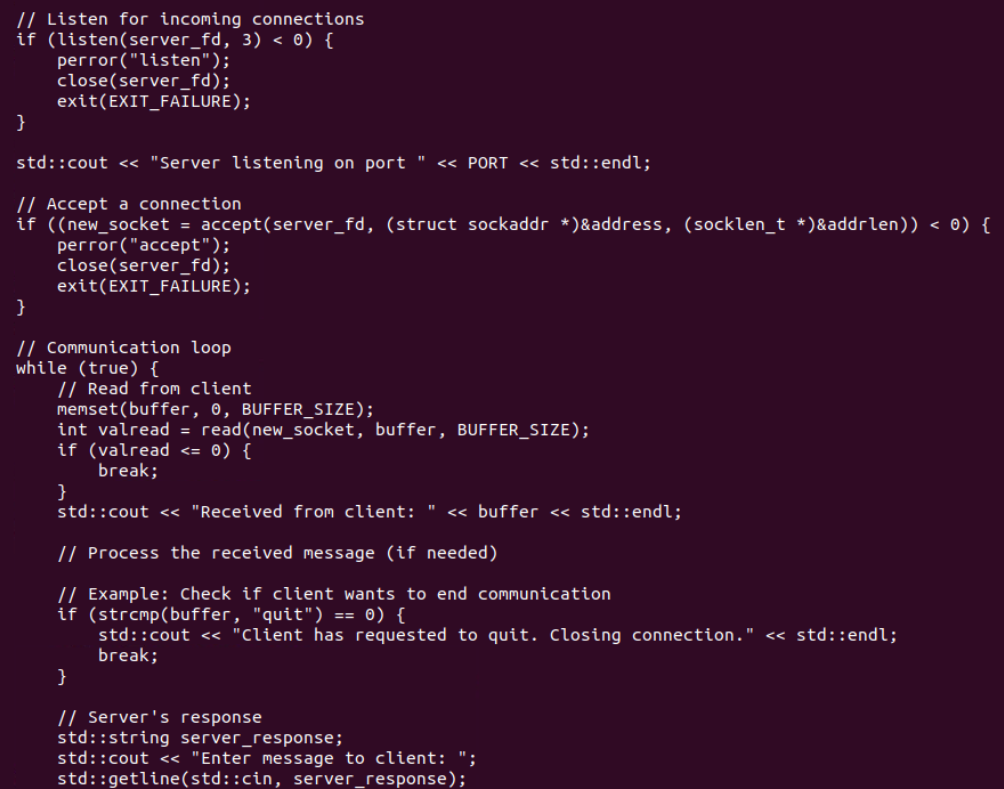


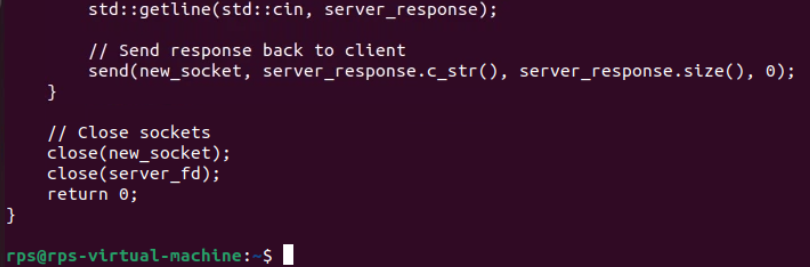
**Connecting two system as a client and server:**

**Server:**

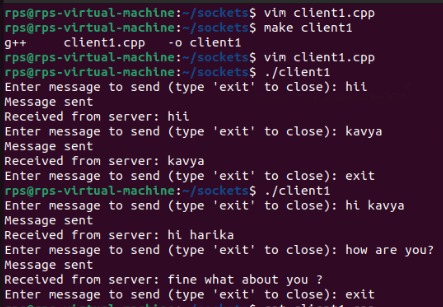
****

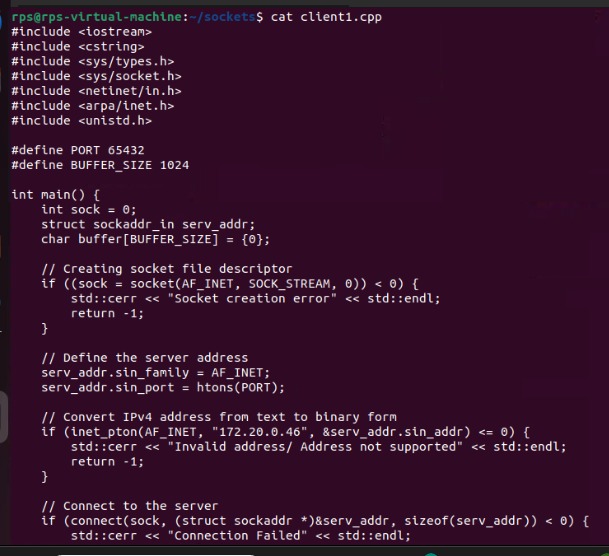
****

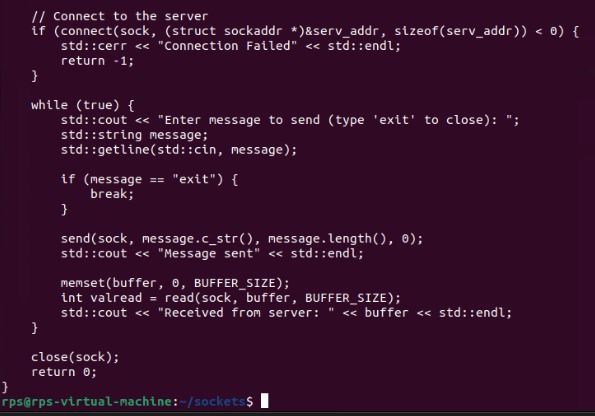
****

****

**Client:**

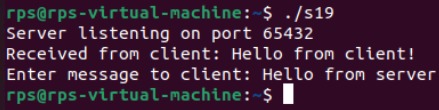
****

****

****

**Connecting two systems client and server and chatting**

**Server:**



#include <iostream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 65432

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Define the server address

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY; // Bind to any available address

address.sin\_port = htons(PORT);

// Bind the socket to the network address and port

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("listen");

close(server\_fd);

exit(EXIT\_FAILURE);

}

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

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen)) < 0) {

perror("accept");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Communication loop

while (true) {

// Read from client

memset(buffer, 0, BUFFER\_SIZE);

int valread = read(new\_socket, buffer, BUFFER\_SIZE);

if (valread <= 0) {

break;

}

std::cout << "Received from client: " << buffer << std::endl;

// Process the received message (if needed)

// Example: Check if client wants to end communication

if (strcmp(buffer, "quit") == 0) {

std::cout << "Client has requested to quit. Closing connection." << std::endl;

break;

}

// Server's response

std::string server\_response;

std::cout << "Enter message to client: ";

std::getline(std::cin, server\_response);

// Send response back to client

send(new\_socket, server\_response.c\_str(), server\_response.size(), 0);

}

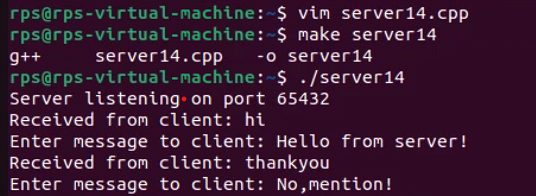
// Close sockets

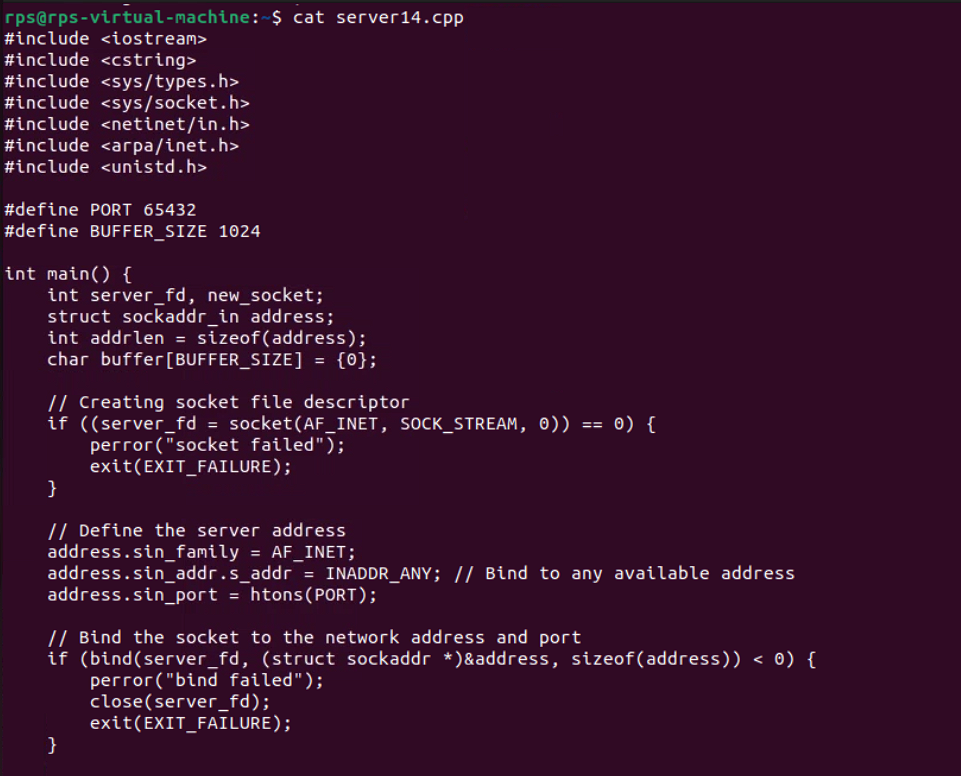
close(new\_socket);

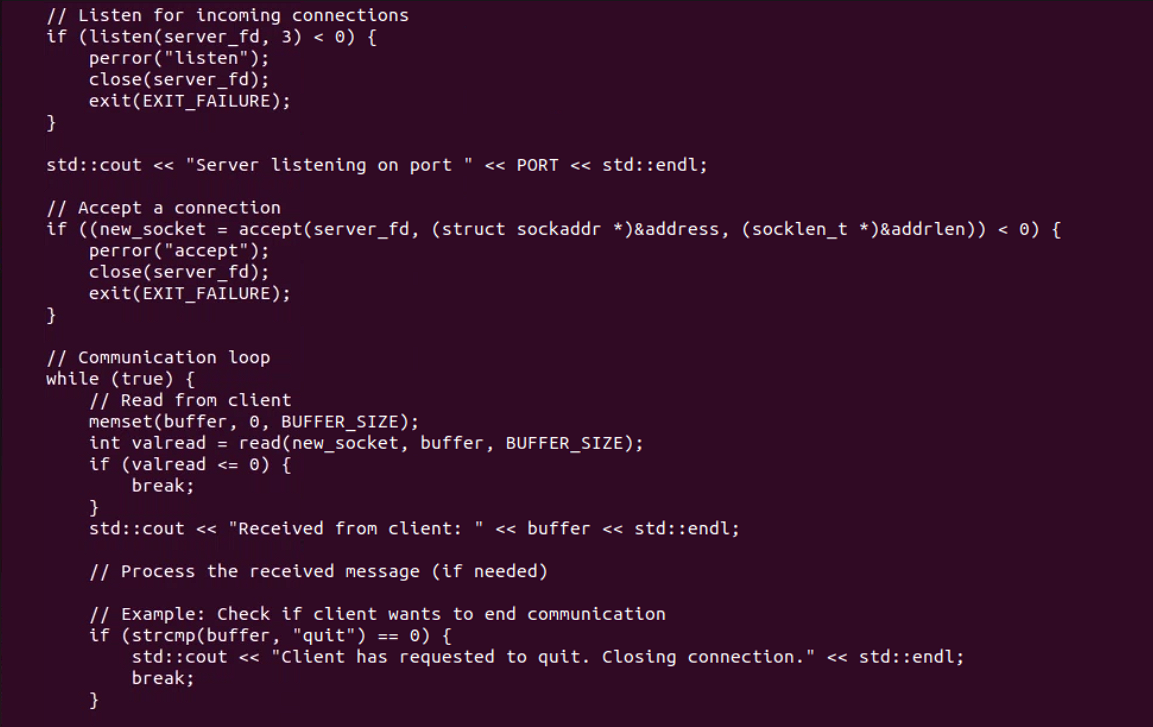
close(server\_fd);

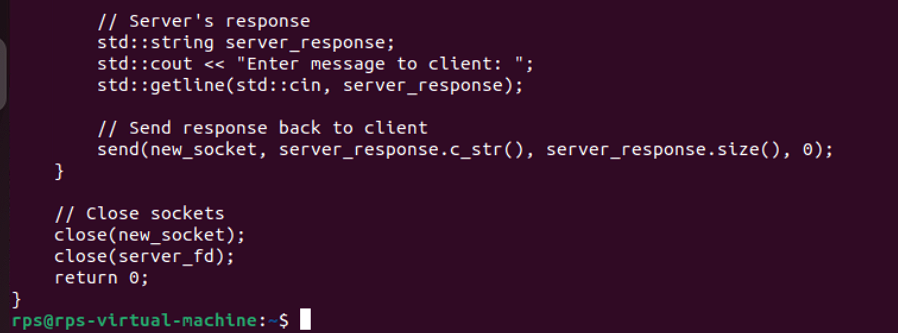
return 0;

}

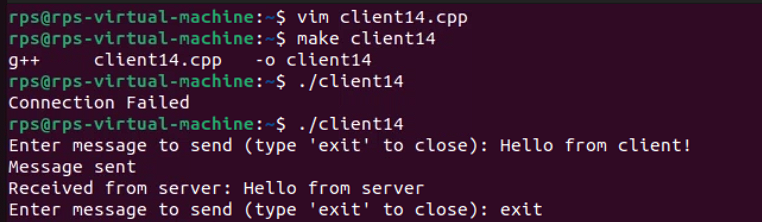


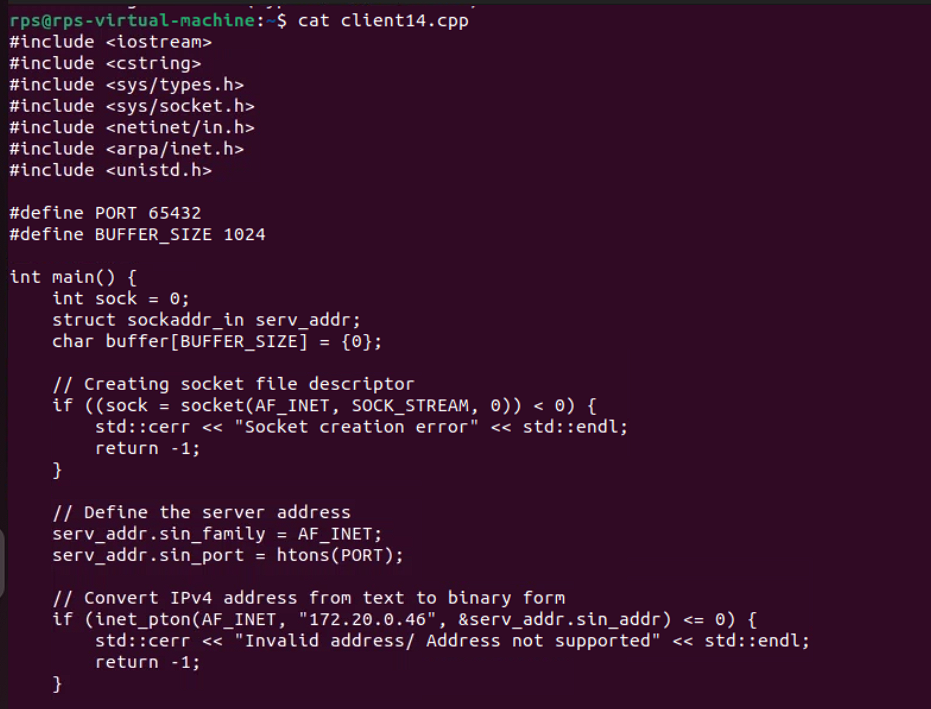


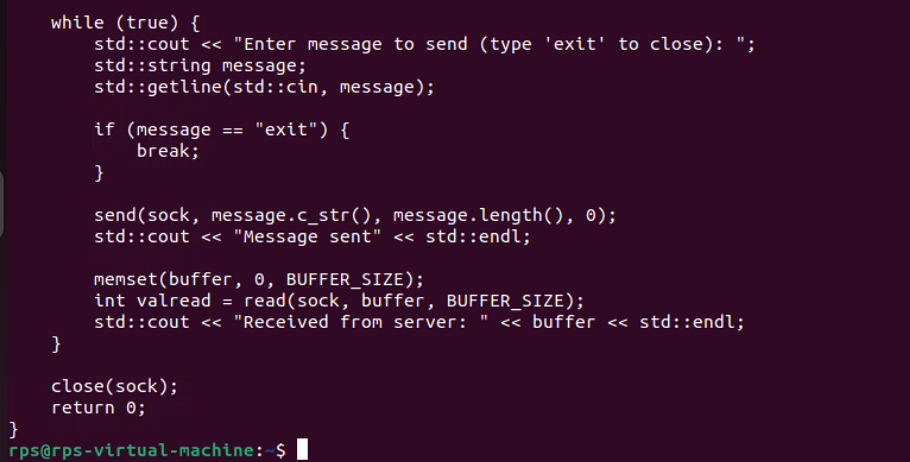




**Client:**

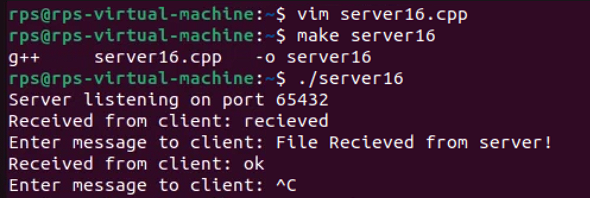
****

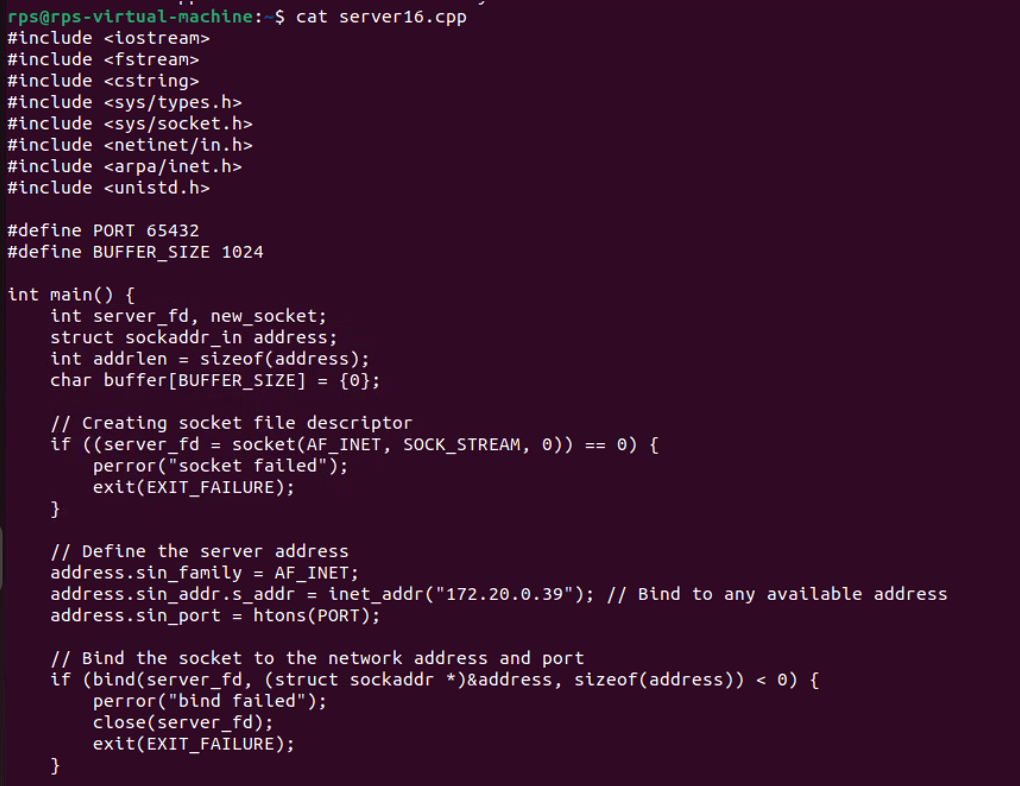
****

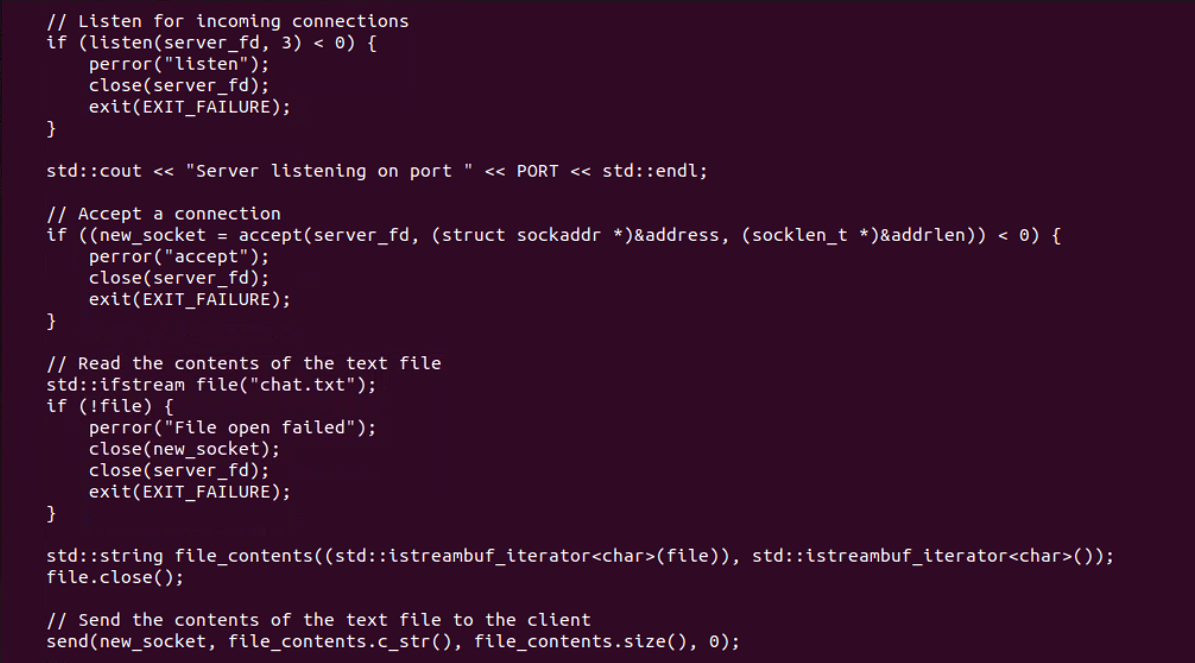
****

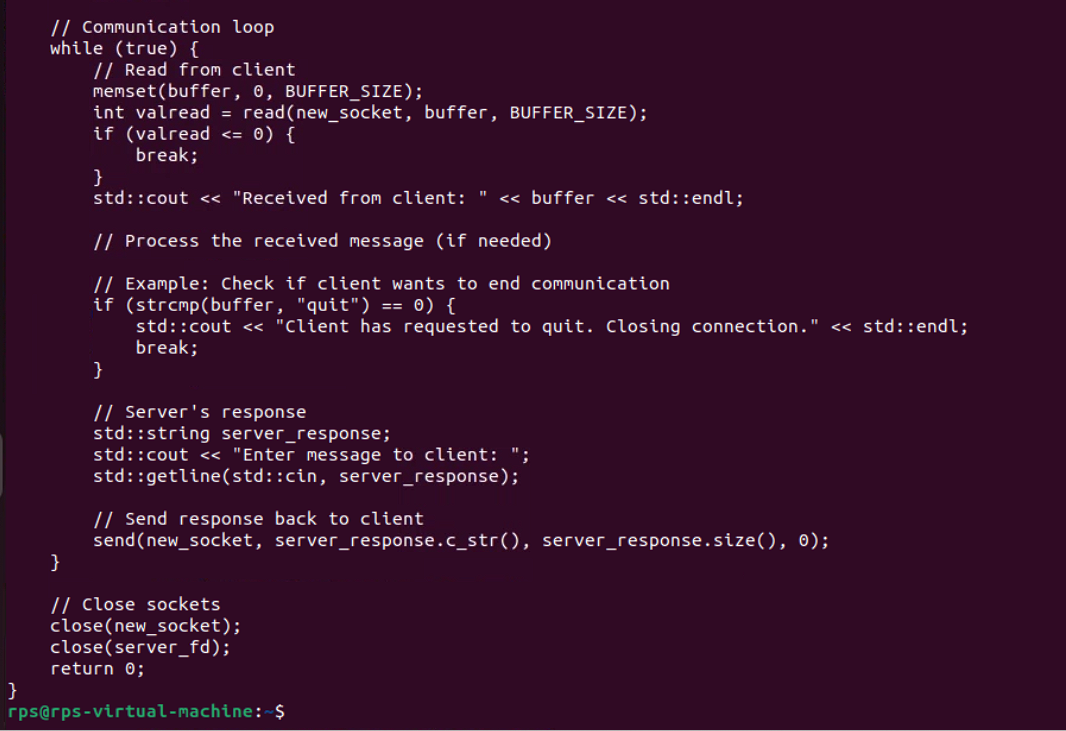
**Task: Create a file and then send from server to client and vice versa.**

**Server:**

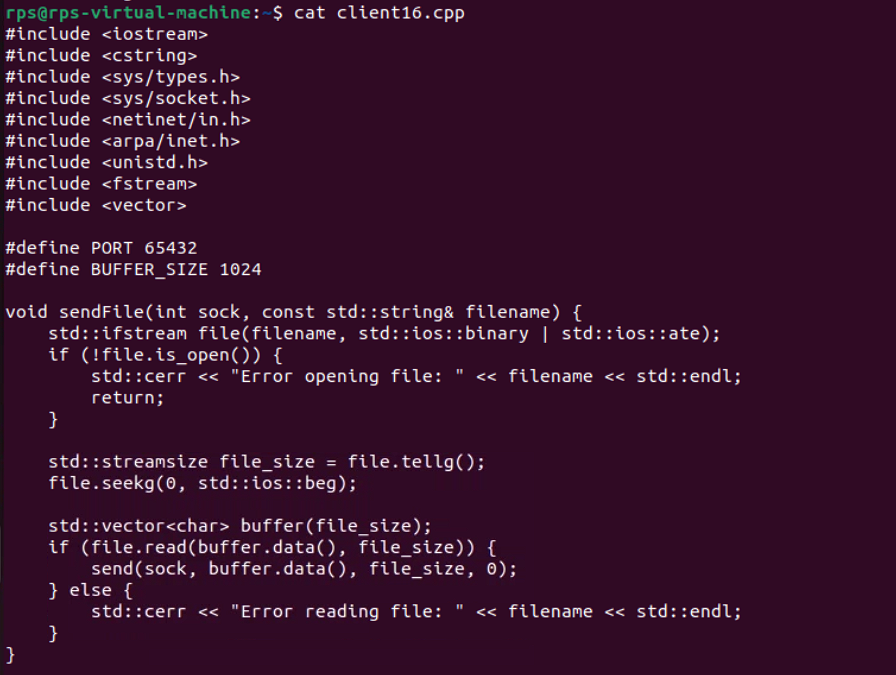
****

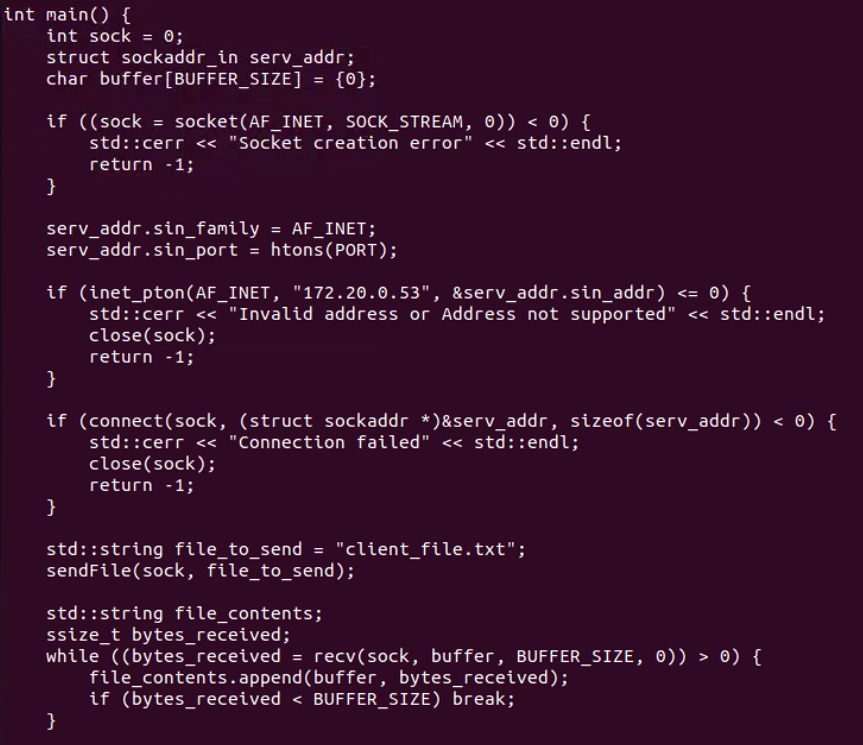
****

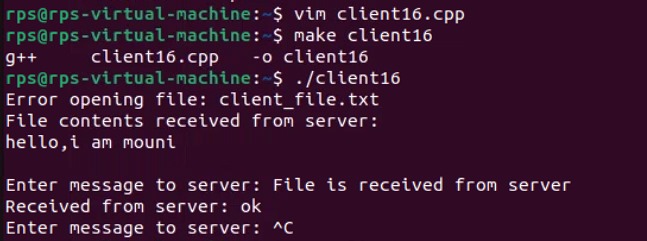
****

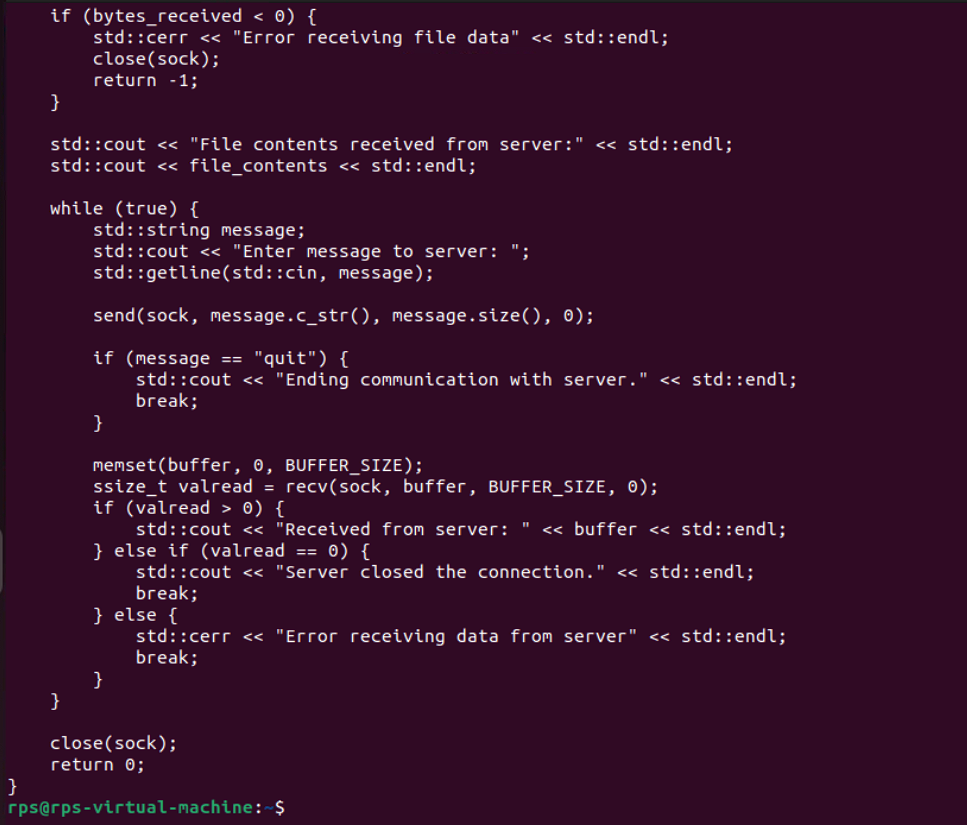
****

**Client:**

****

****



****