**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.**

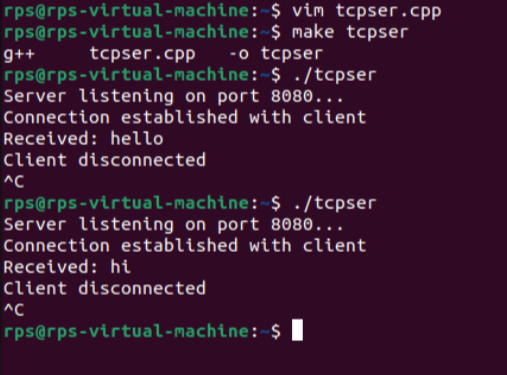
**Requirements:**

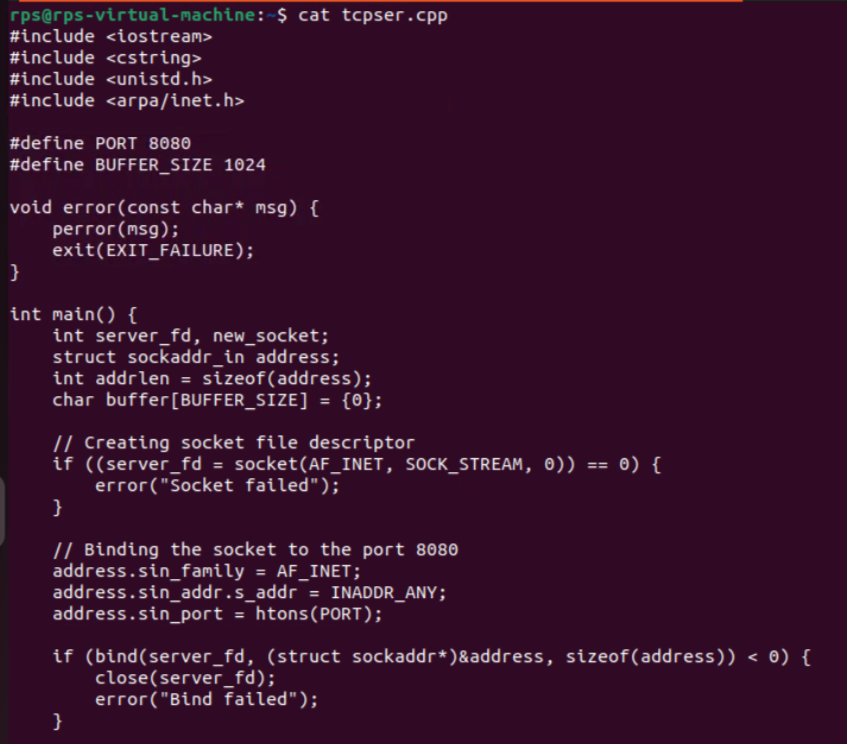
**The server should run indefinitely, waiting for client connections.**

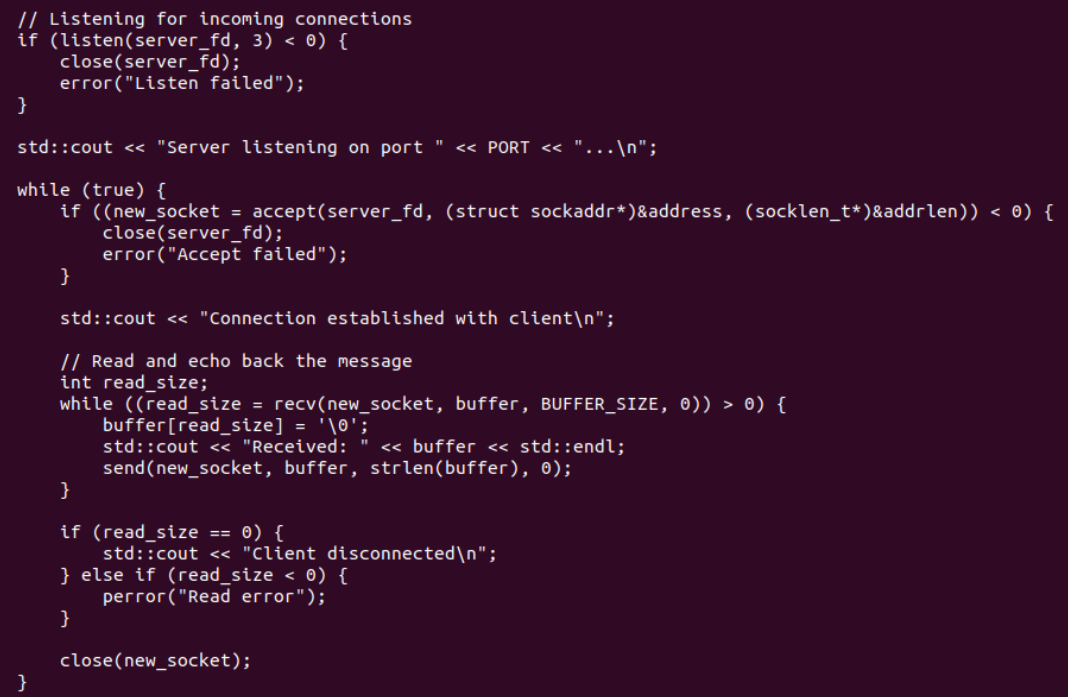
**The client should take a message as input from the user, send it to the server, and display the response.**

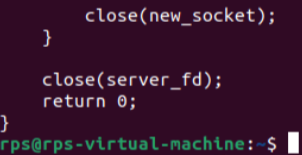
**Implement proper error handling and cleanup (e.g., closing sockets).**

**Server:**

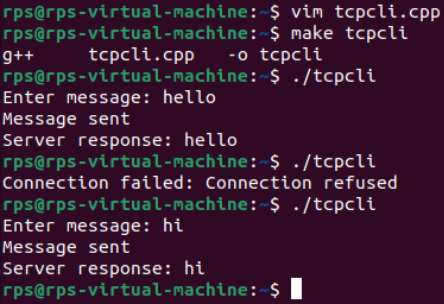
****

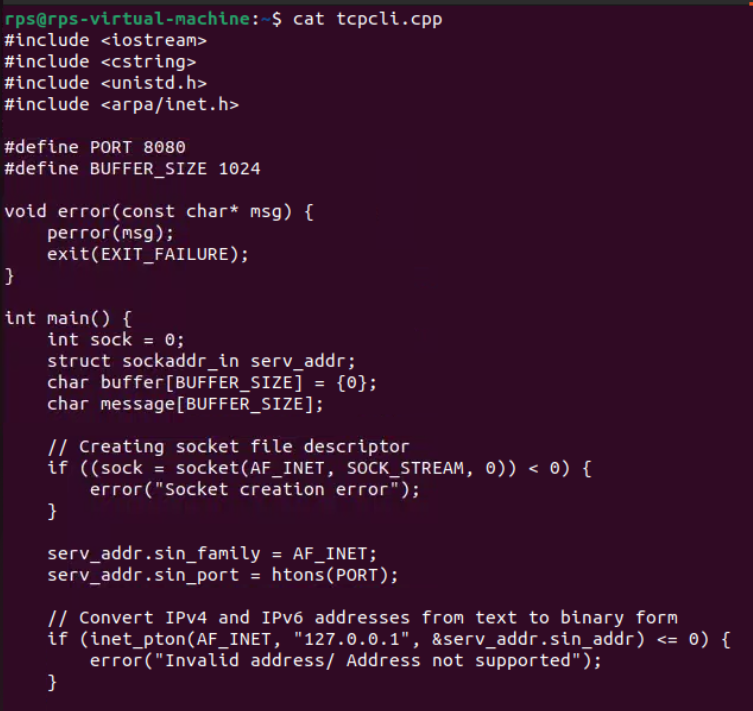
****

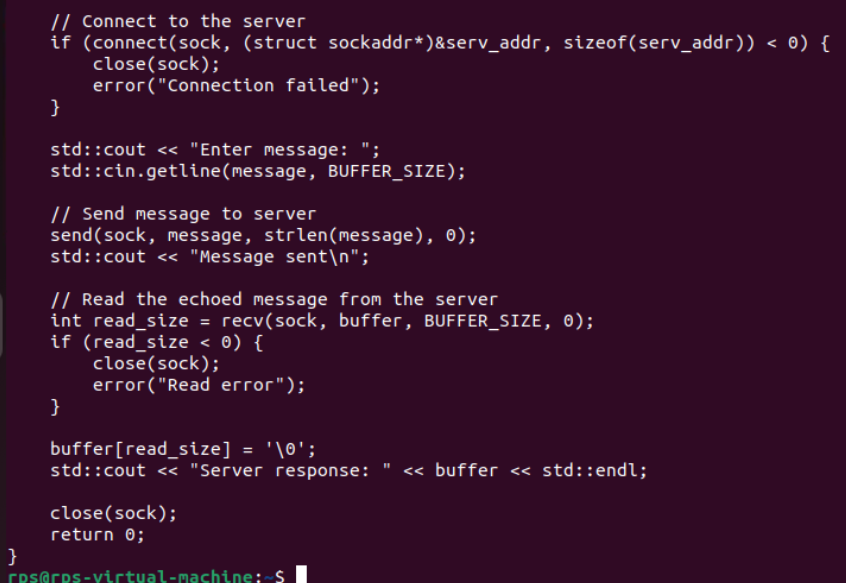
****

****

**Client:**

****

****

****

**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.**

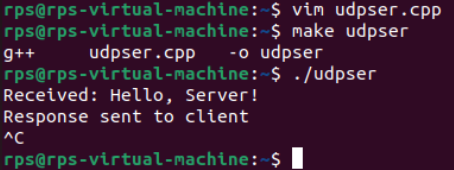
**Requirements:**

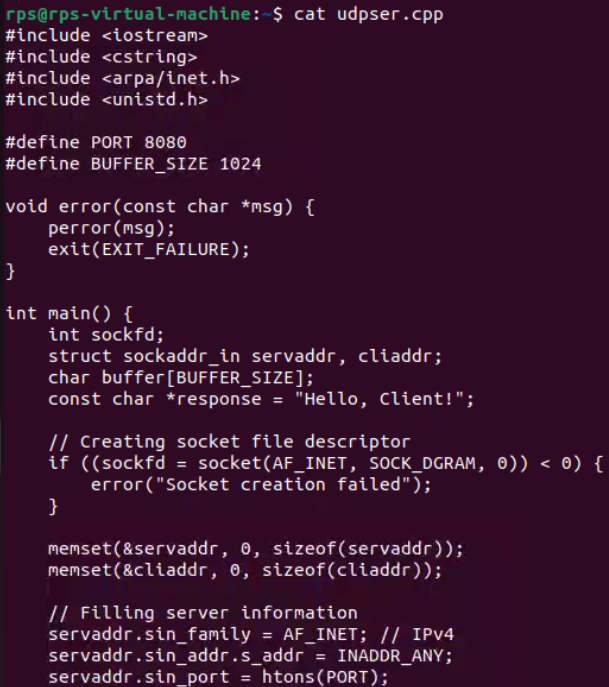
**The server should run indefinitely, waiting for incoming messages.**

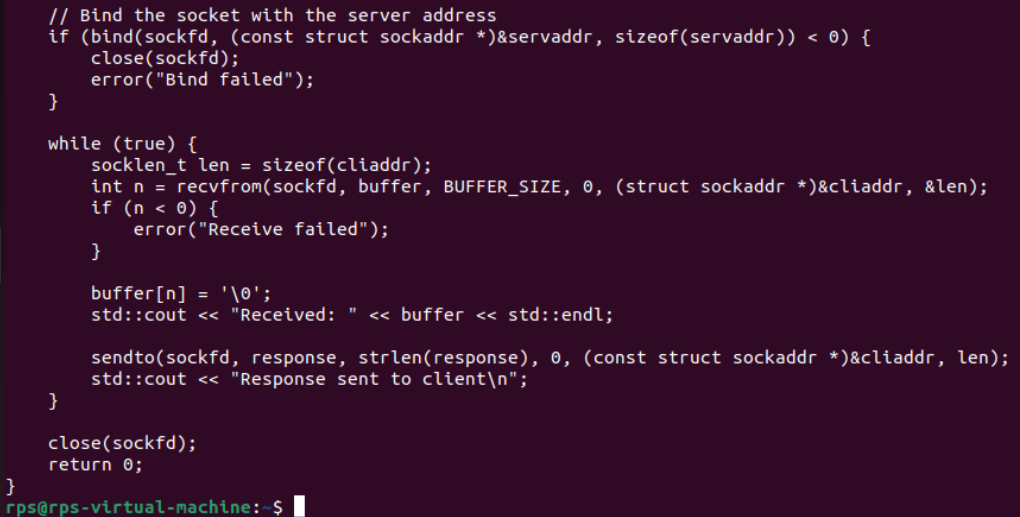
**The client should send a predefined message (e.g., "Hello, Server!") and display the server's response.**

**Implement proper error handling.**

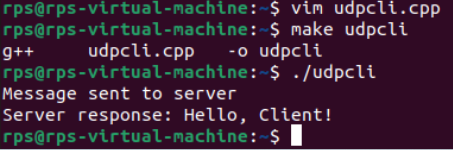
**Server:**

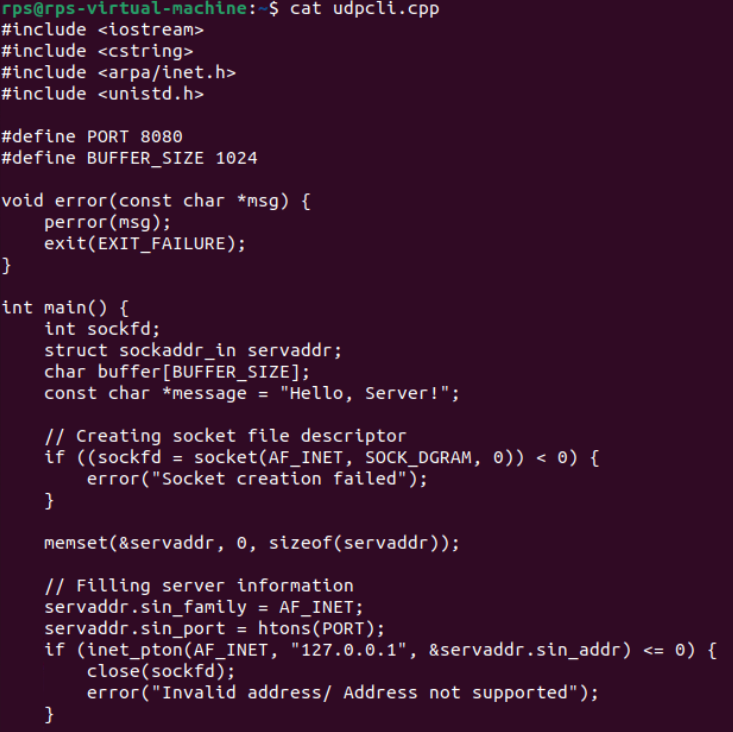
****

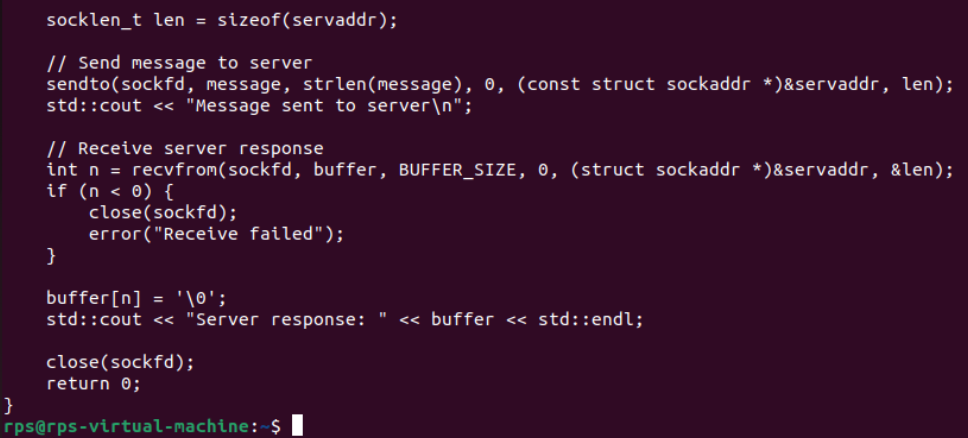
****

****

**Client:**

****

****

****

**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.**

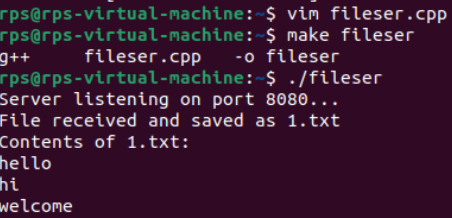
**Requirements:**

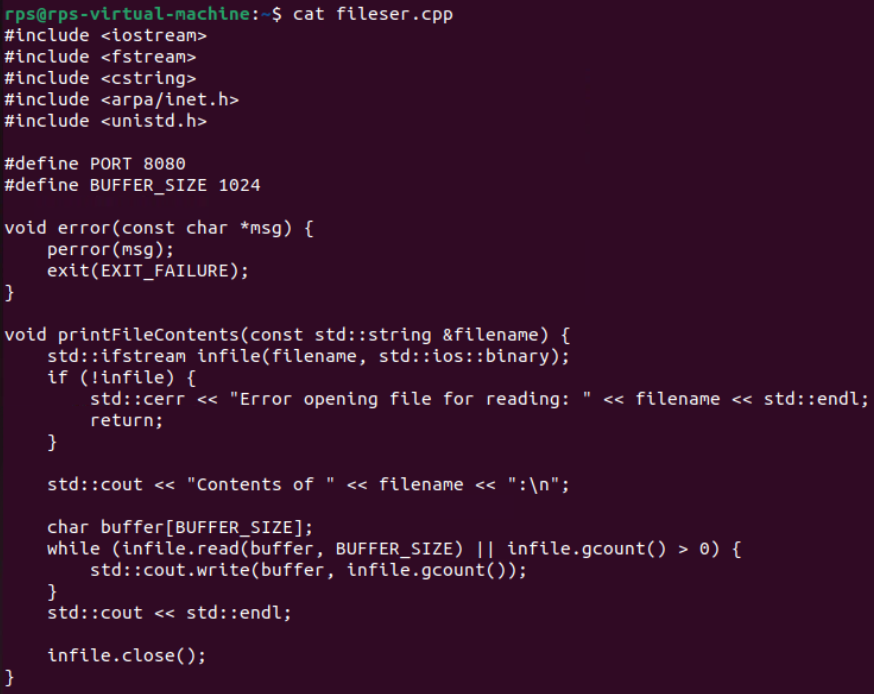
**The server should run indefinitely, waiting for file transfer requests.**

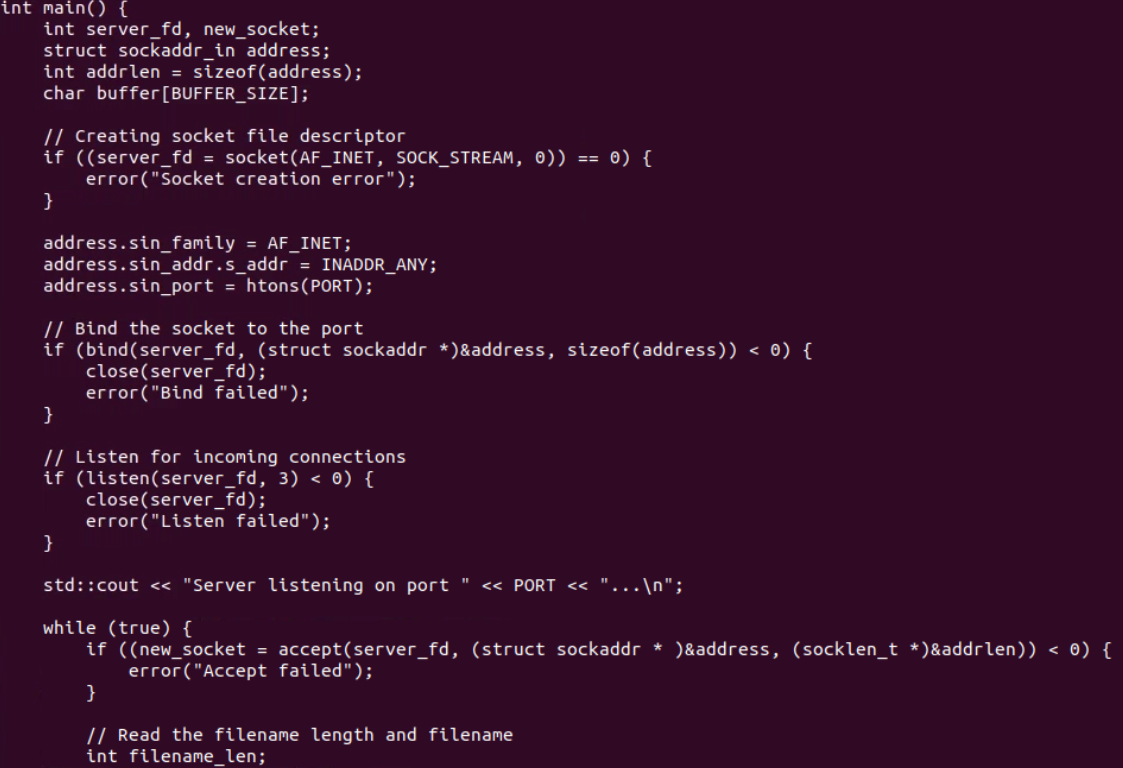
**The client should prompt the user for a file path, read the file, and send its contents to the server.**

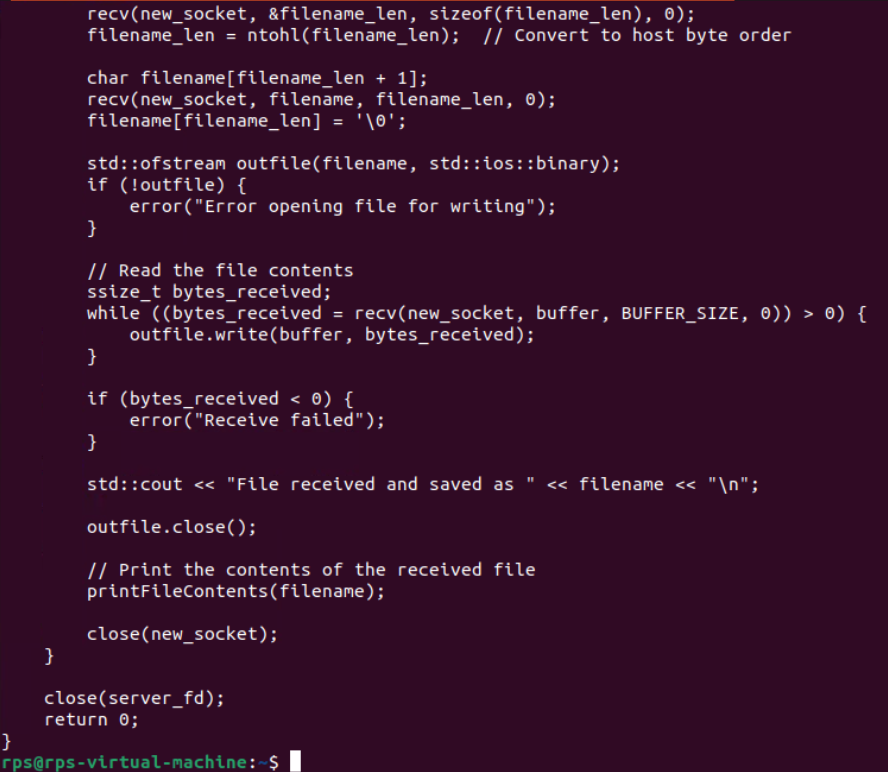
**Implement proper error handling and file operations.**

**Server:**

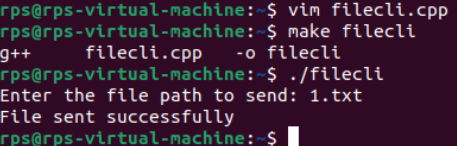
****

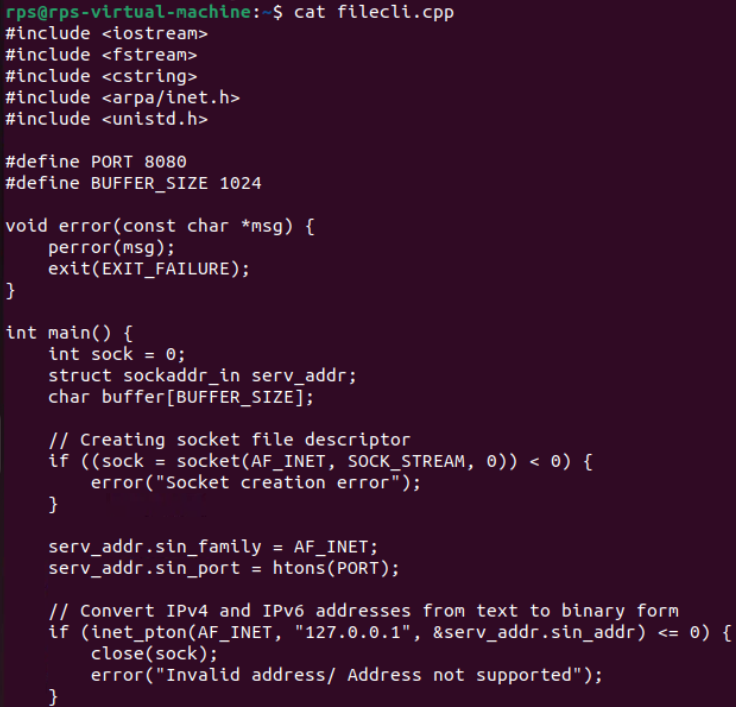
****

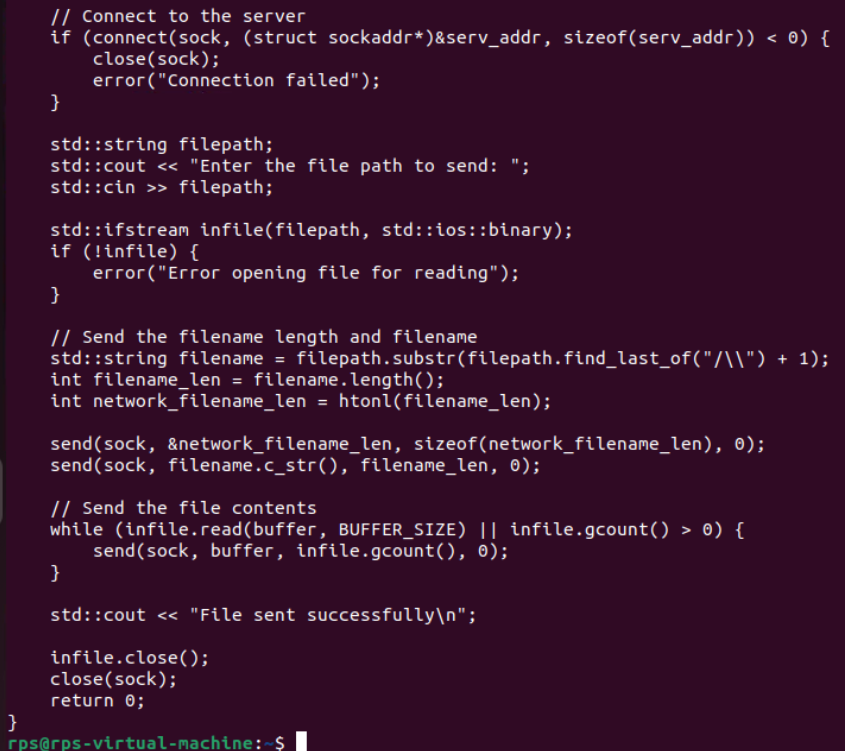
****

****

**Client:**

****

****

****

**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.**

**SERVER:**

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#define BROADCAST\_IP "172.20.0.53"

#define BROADCAST\_PORT 12345

#define MESSAGE "Hello, this is a broadcast message!"

int main() {

int sockfd;

struct sockaddr\_in broadcast\_addr;

int broadcast = 1;

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

if (setsockopt(sockfd, SOL\_SOCKET, SO\_BROADCAST, &broadcast, sizeof(broadcast)) < 0) {

perror("setsockopt (SO\_BROADCAST) failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

memset(&broadcast\_addr, 0, sizeof(broadcast\_addr));

broadcast\_addr.sin\_family = AF\_INET;

broadcast\_addr.sin\_port = htons(BROADCAST\_PORT);

broadcast\_addr.sin\_addr.s\_addr = inet\_addr(BROADCAST\_IP);

if (sendto(sockfd, MESSAGE, strlen(MESSAGE), 0, (struct sockaddr\*)&broadcast\_addr, sizeof(broadcast\_addr)) < 0) {

perror("sendto failed");

close(sockfd);

exit(EXIT\_FAILURE);

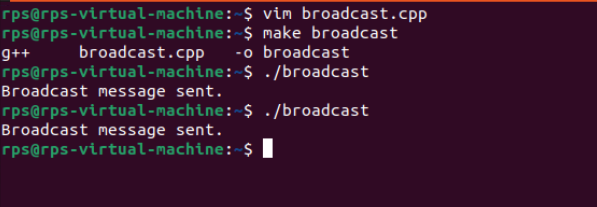
}

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

close(sockfd);

return 0;

}



**CLIENT:**

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <arpa/inet.h>

#define BROADCAST\_PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in recv\_addr;

char buffer[BUFFER\_SIZE];

int recv\_len;

socklen\_t addr\_len = sizeof(recv\_addr);

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

memset(&recv\_addr, 0, sizeof(recv\_addr));

recv\_addr.sin\_family = AF\_INET;

recv\_addr.sin\_port = htons(BROADCAST\_PORT);

recv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(sockfd, (struct sockaddr\*)&recv\_addr, sizeof(recv\_addr)) < 0) {

perror("bind failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

std::cout << "Listening for broadcast messages on port " << BROADCAST\_PORT << "..." << std::endl;

while (true) {

recv\_len = recvfrom(sockfd, buffer, BUFFER\_SIZE - 1, 0, (struct sockaddr\*)&recv\_addr, &addr\_len);

if (recv\_len < 0) {

perror("recvfrom failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

buffer[recv\_len] = '\0';

std::cout << "Received broadcast message: " << buffer << std::endl;

}

close(sockfd);

return 0;

}