**File Manipulation using System Calls in C++ on Linux**

**Objective:**

**Create a C++ program that performs file manipulation using Linux system calls. The program should be able to:**

**Create a new file.**

**Write a specified string to the file.**

**Read the contents of the file and display them on the console.**

**Append additional text to the file.**

**Delete the file.**

**Requirements: Use system calls like open, read, write, close, and unlink. Handle errors appropriately by checking the return values of system calls and using perror to print error messages. Ensure the program is modular with separate functions for each file operation (create, write, read, append, delete).**

**#include <iostream>**

**#include <fcntl.h>**

**#include <unistd.h>**

**#include <cstring>**

**#include <cerrno>**

**void createFile(const char\* filename) {**

**int fd = open(filename, O\_CREAT | O\_WRONLY, 0644);**

**if (fd == -1) {**

**perror("Error creating file");**

**return; }**

**close(fd);**

**}**

**void writeFile(const char\* filename, const char\* content) {**

**int fd = open(filename, O\_WRONLY | O\_TRUNC);**

**if (fd == -1) {**

**perror("Error opening file for writing");**

**return;**

**}**

**if (write(fd, content, strlen(content)) == -1) {**

**perror("Error writing to file");**

**}**

**close(fd);**

**}**

**void readFile(const char\* filename) {**

**int fd = open(filename, O\_RDONLY);**

**if (fd == -1) {**

**perror("Error opening file for reading");**

**return;**

**}**

**const size\_t bufferSize = 1024;**

**char buffer[bufferSize];**

**ssize\_t bytesRead;**

**while ((bytesRead = read(fd, buffer, bufferSize - 1)) > 0) {**

**buffer[bytesRead] = '\0';**

**std::cout << buffer;**

**}**

**if (bytesRead == -1) {**

**perror("Error reading file"); }**

**close(fd); }**

**void appendFile(const char\* filename, const char\* content) {**

**int fd = open(filename, O\_WRONLY | O\_APPEND);**

**if (fd == -1) {**

**perror("Error opening file for appending");**

**return;**

**}**

**if (write(fd, content, strlen(content)) == -1) {**

**perror("Error appending to file"); }**

**close(fd); }**

**void deleteFile(const char\* filename) {**

**if (unlink(filename) == -1) {**

**perror("Error deleting file"); }**

**}**

**int main() {**

**const char\* filename = "testfile.txt";**

**const char\* content = "This is a test string.\n";**

**const char\* appendContent = "This is appended text.\n";**

**std::cout << "Creating file..." << std::endl;**

**createFile(filename);**

**std::cout << "Writing to file..." << std::endl;**

**writeFile(filename, content);**

**std::cout << "Reading file contents..." << std::endl;**

**readFile(filename);**

**std::cout << "\nAppending to file..." << std::endl;**

**appendFile(filename, appendContent);**

**std::cout << "Reading file contents again..." << std::endl;**

**readFile(filename);**

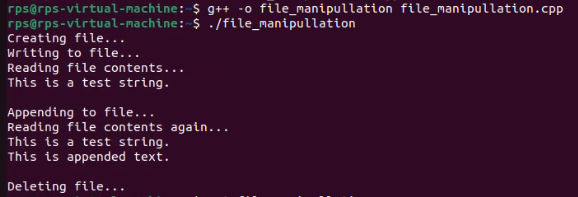
**std::cout << "\nDeleting file..." << std::endl;**

**deleteFile(filename);**

**return 0;**

**}**

**OUTPUT:**

****

**Conversation between client and server**

**Server:**

**#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;**

**serverSocket = socket(AF\_INET, SOCK\_DGRAM, 0);**

**if (serverSocket < 0) {**

**std::cerr << "Socket creation failed" << std::endl;**

**return 1;**

**}**

**serverAddr.sin\_family = AF\_INET;**

**serverAddr.sin\_port = htons(DEFAULT\_PORT);**

**serverAddr.sin\_addr.s\_addr = INADDR\_ANY;**

**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; }**

**recvbuf[recvLen] = '\0';**

**std::cout << "Received: " << recvbuf << std::endl;**

**int sendLen = sendto(serverSocket, recvbuf, recvLen, 0, (struct sockaddr\*)&clientAddr, clientAddrLen);**

**if (sendLen < 0) {**

**std::cerr << "sendto failed" << std::endl;**

**close(serverSocket);**

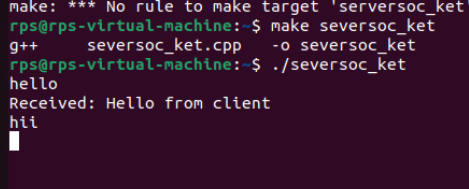
**return 1; }**

**}**

**close(serverSocket);**

**return 0;**

**}**

****

**CLIENT:**

**#include <iostream>**

**#include <cstring>**

**#include <unistd.h>**

**#include <arpa/inet.h>**

**#define DEFAULT\_PORT 8080**

**#define DEFAULT\_BUFLEN 512**

**int main() {**

**int clientSocket;**

**struct sockaddr\_in serverAddr;**

**char sendbuf[DEFAULT\_BUFLEN] = "Hello from client";**

**char recvbuf[DEFAULT\_BUFLEN];**

**int recvbuflen = DEFAULT\_BUFLEN;**

**socklen\_t serverAddrLen = sizeof(serverAddr);**

**clientSocket = socket(AF\_INET, SOCK\_DGRAM, 0);**

**if (clientSocket < 0) {**

**std::cerr << "Socket creation failed" << std::endl;**

**return 1; }**

**serverAddr.sin\_family = AF\_INET;**

**serverAddr.sin\_port = htons(DEFAULT\_PORT);**

**inet\_pton(AF\_INET, "127.0.0.1", &serverAddr.sin\_addr);**

**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; }**

**std::cout << "Sent: " << sendbuf << std::endl;**

**int recvLen = recvfrom(clientSocket, recvbuf, recvbuflen, 0, (struct sockaddr\*)&serverAddr, &serverAddrLen);**

**if (recvLen < 0) {**

**std::cerr << "recvfrom failed" << std::endl;**

**close(clientSocket);**

**return 1; }**

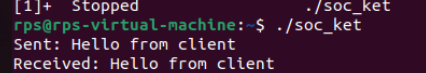
**recvbuf[recvLen] = '\0'; // Null-terminate the received data**

**std::cout << "Received: " << recvbuf << std::endl;**

**close(clientSocket);**

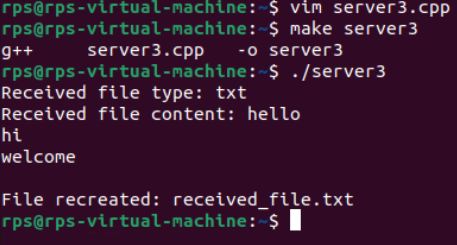
**return 0;**

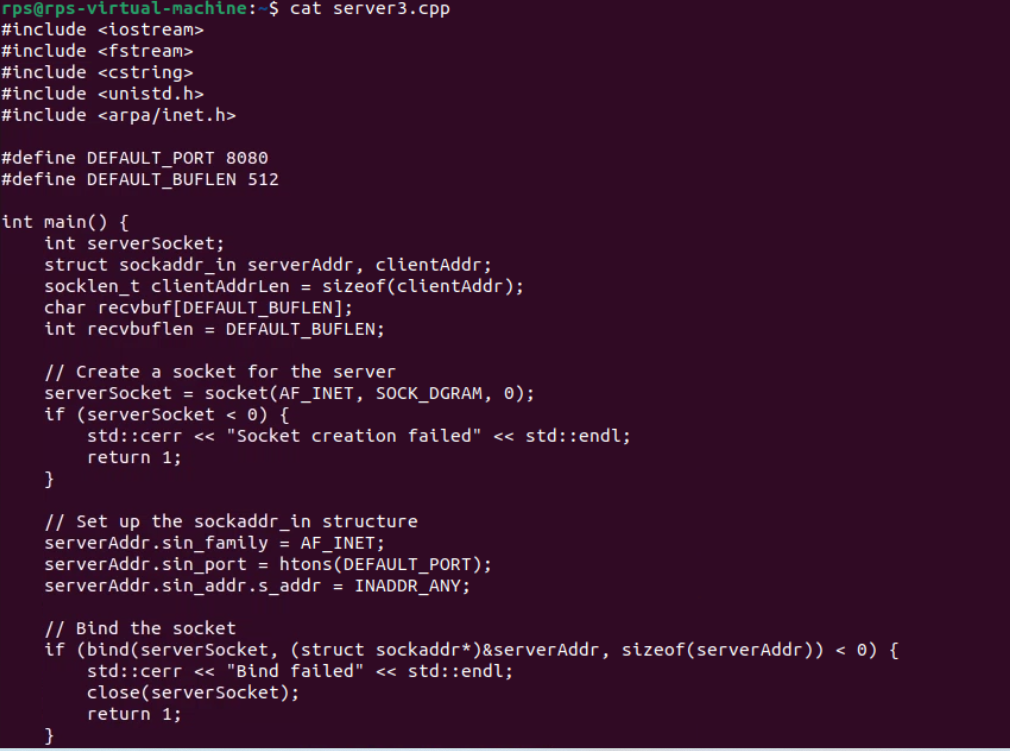
**}**

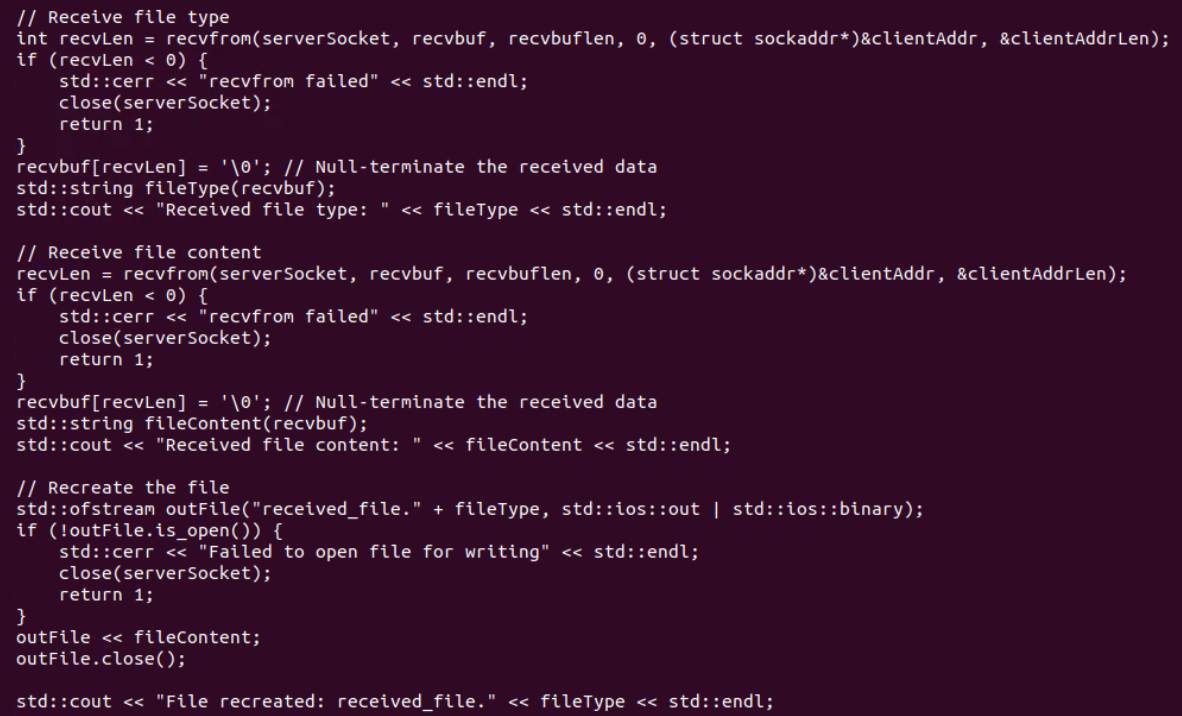
****

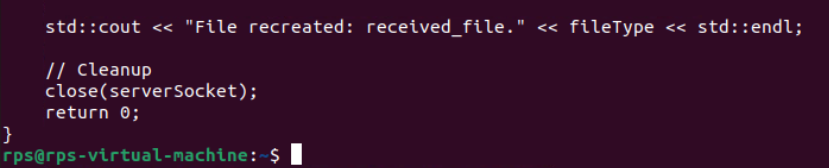
**File send Client to Server:**

**Server:**

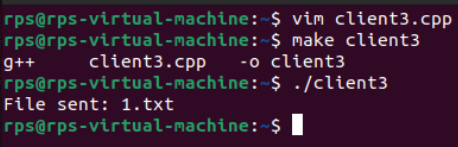
****

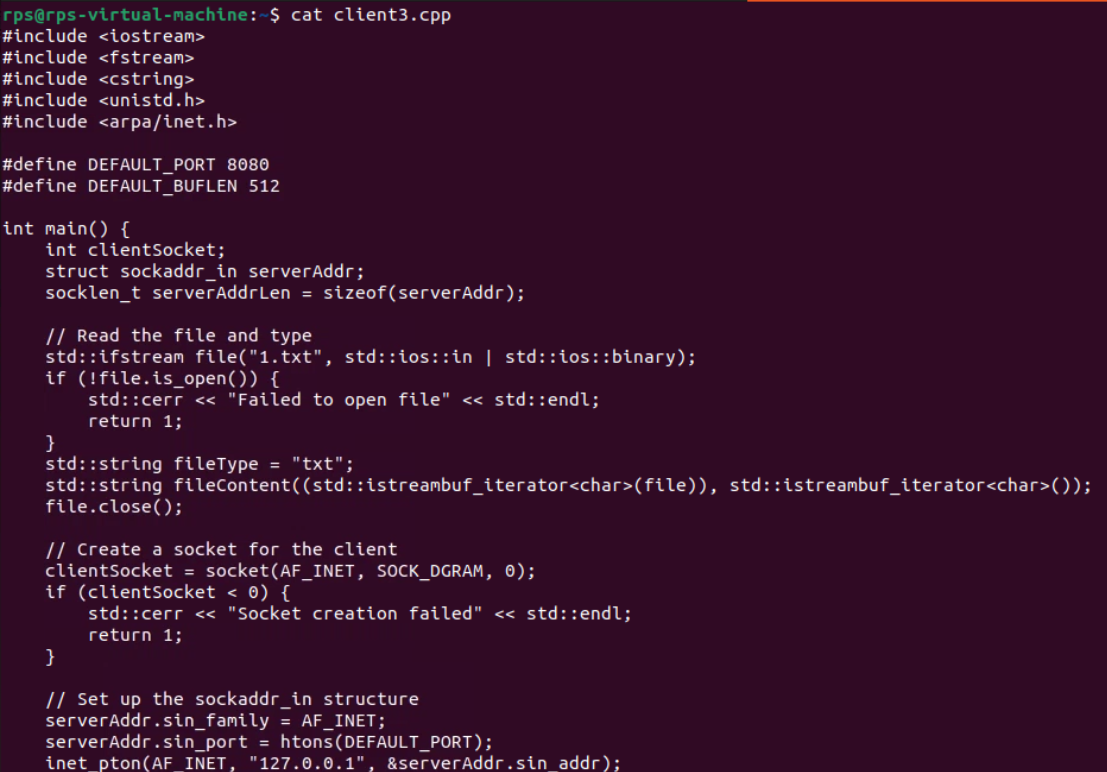
****

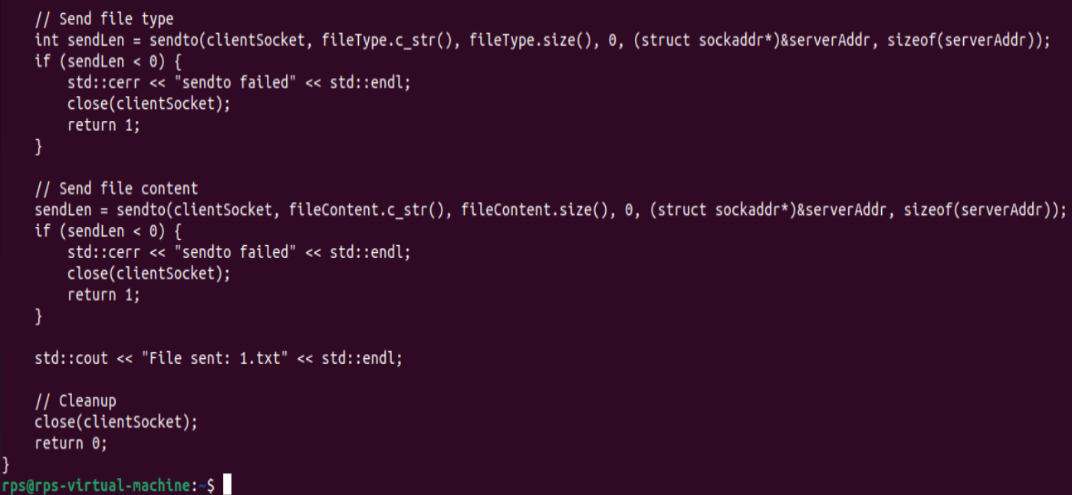
****

****

**Client:**

****

****

****

**UDP Server Implementation:**

**Create a UDP socket.**

**Bind the socket to a specified port.**

**Implement a loop to continuously listen for incoming messages.**

**Upon receiving a message:**

**Print the received message along with the client’s address and port.**

**Send an acknowledgment message ("Message received") back to the client.**

**Ensure proper error handling and resource cleanup.**

**2. UDP Client Implementation:**

**Create a UDP socket.**

**Allow the user to input the server’s IP address and port number.**

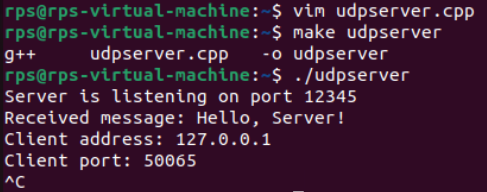
**Send a predefined message (e.g., "Hello, Server!") to the server.**

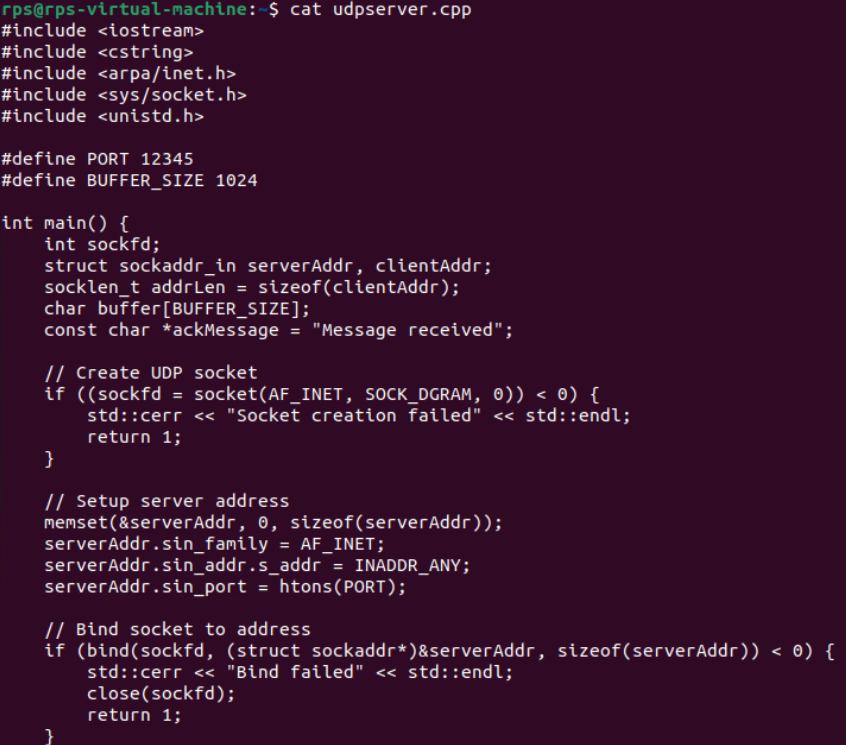
**Wait for an acknowledgment from the server.**

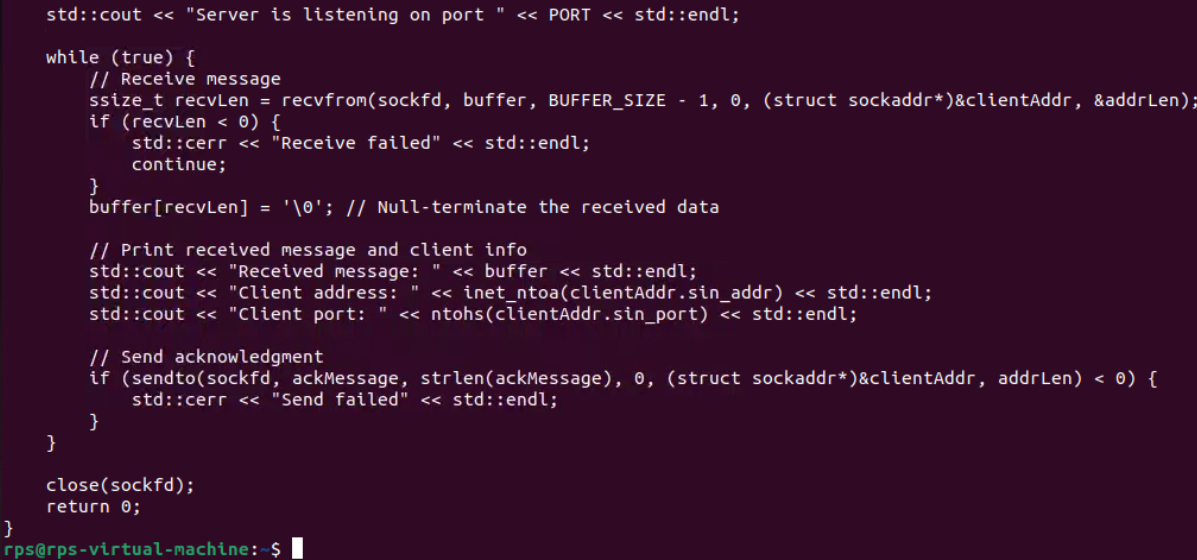
**Print the acknowledgment message to the console.**

**Ensure proper error handling and resource cleanup.**

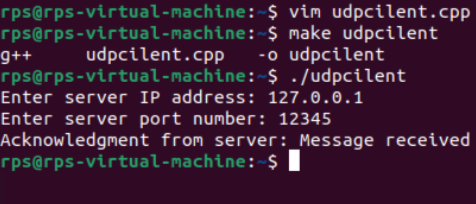
**Server:**

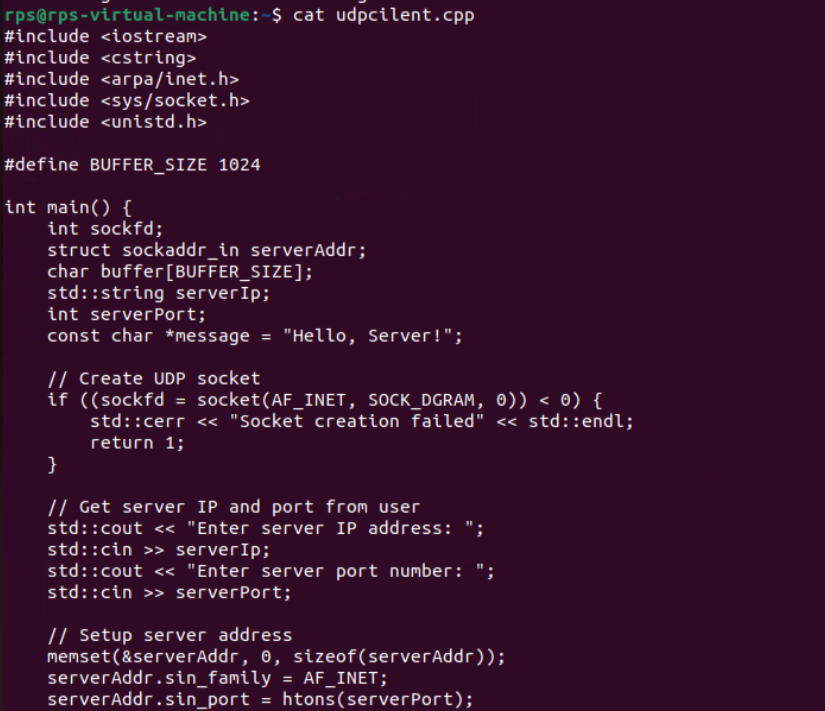
****

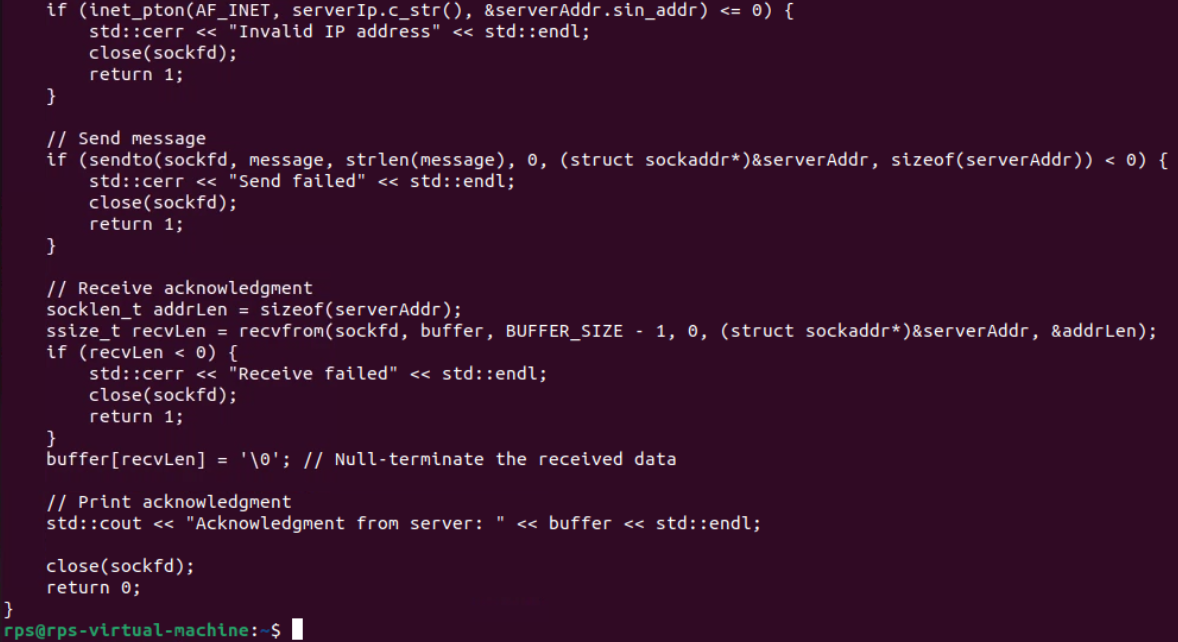
****

****

**Client:**

****

****

****