Day – 13 LSP Assignment Task – 1

1. File Manipulation using System Calls in C++ on Linux

Objective:

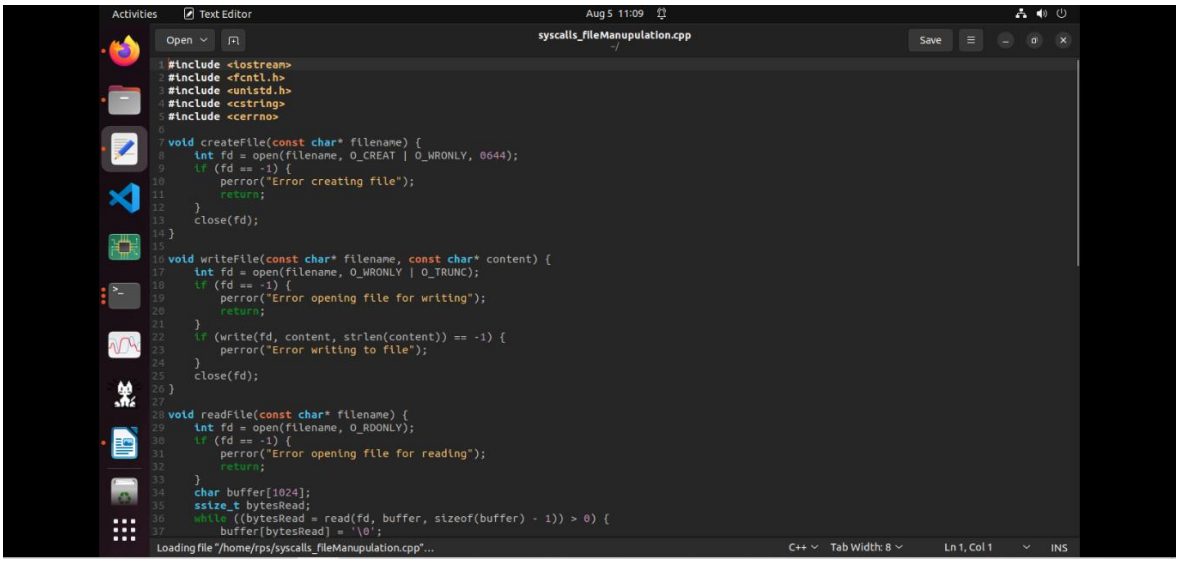
Create a C++ program that performs file manipulation using Linux system calls. The program should be able to:

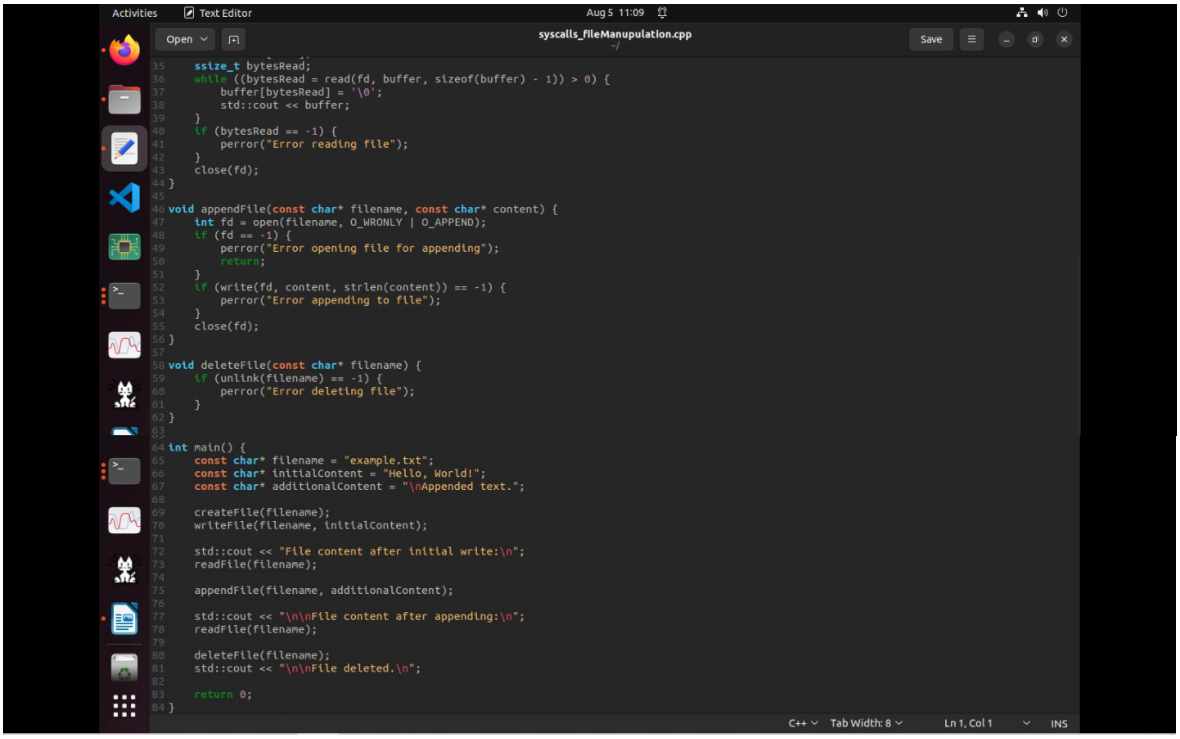
a. Create a new file.
b. Write a specified string to the file.
c. Read the contents of the file and display them on the console.
d. Append additional text to the file.
e. Delete the file.

Requirements:

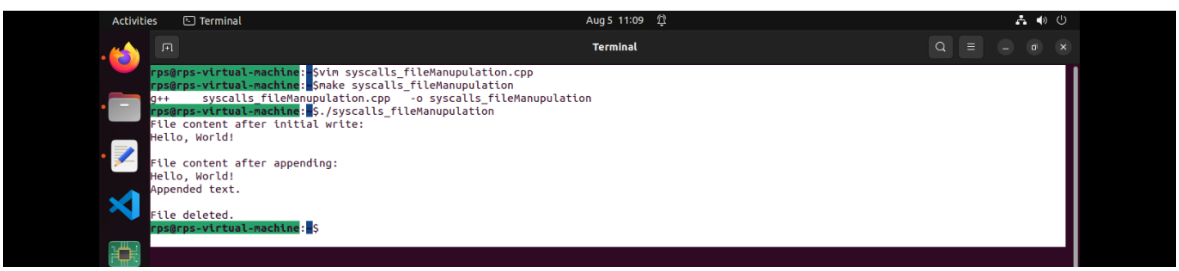
a. Use system calls like open, read, write, close, and unlink.
b. Handle errors appropriately by checking the return values of system calls and using perror to print error messages.

Ensure the program is modular with separate functions for each file operation (create, write, read, append, delete).



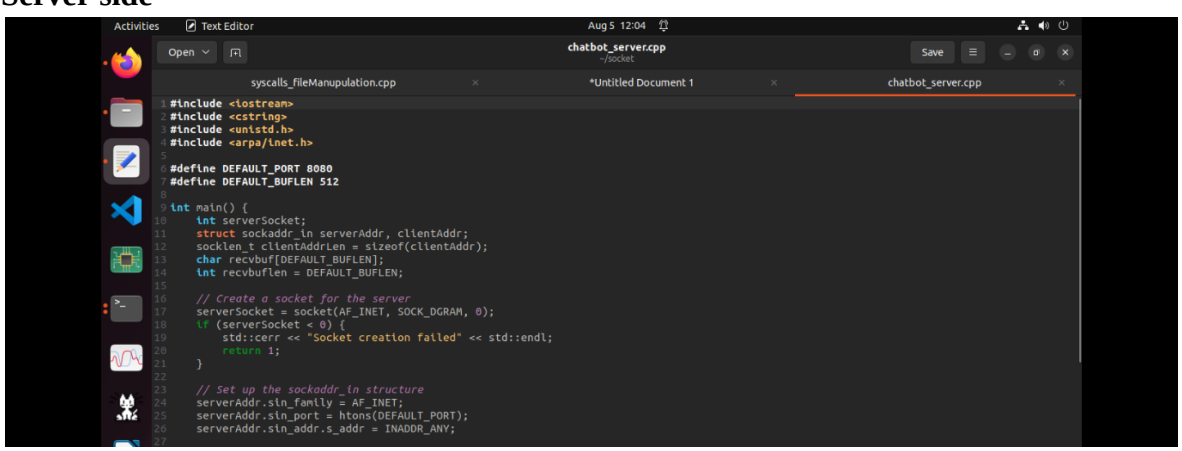


Execution:



2. 1. Create a chat-bot using UDP Protocol from one side.

a. Server side



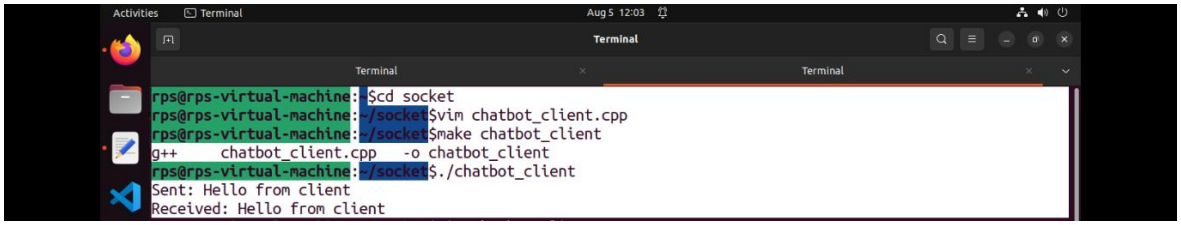
```cpp
    // Bind the socket
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
        std::cerr << "Bind failed" << std::endl;
        close(serverSocket);
        return 1;
    }

    // Receive data
    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'; // Null-terminate the received data
        std::cout << "Received: " << recvbuf << std::endl;

        // Echo the data back to the client
        int sendLen = sendto(serverSocket, recvbuf, recvLen, 0, (struct sockaddr*)&clientAddr, clientAddrLen);
        if (sendLen < 0) {
            std::cerr << "sendto failed" << std::endl;
            close(serverSocket);
            return 1;
        }
    }

    // Cleanup
    close(serverSocket);
    return 0;
}
```

```
client_test          pass_SERVERfile.txt  SERVER_PASSfile.cpp    socket_server1       TCP_client1.cpp
client_test1         :q                   server_test1           socket_server1.cpp   tcp_client.cpp
client_test1.cpp     q1.sh                server_test1.cpp       socket_serverA       TCP_client.cpp
rps@rps-virtual-machine:~/socket$vim chatbot_server.cpp
rps@rps-virtual-machine:~/socket$make chatbot_server
g++     chatbot_server.cpp   -o chatbot_server
rps@rps-virtual-machine:~/socket$./chatbot_server
Received: Hello from client
^Z
[1]+  Stopped                 ./chatbot_server
```

b. Client- side

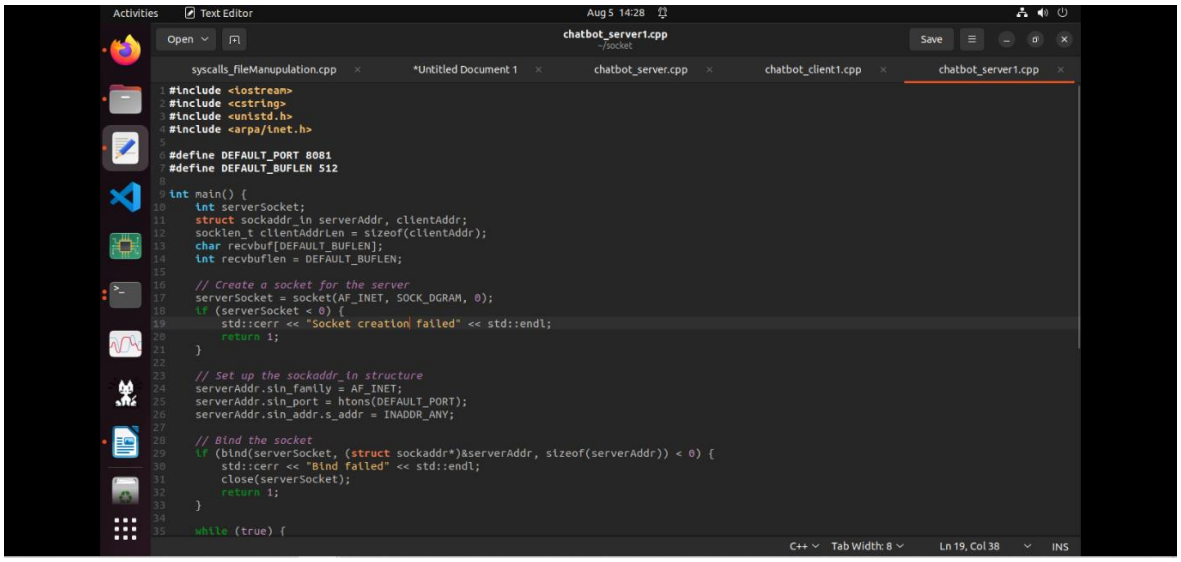
```cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8081
#define DEFAULT_BUFLEN 512

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char sendbuf[DEFAULT_BUFLEN];
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the client
    clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (clientSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);
```

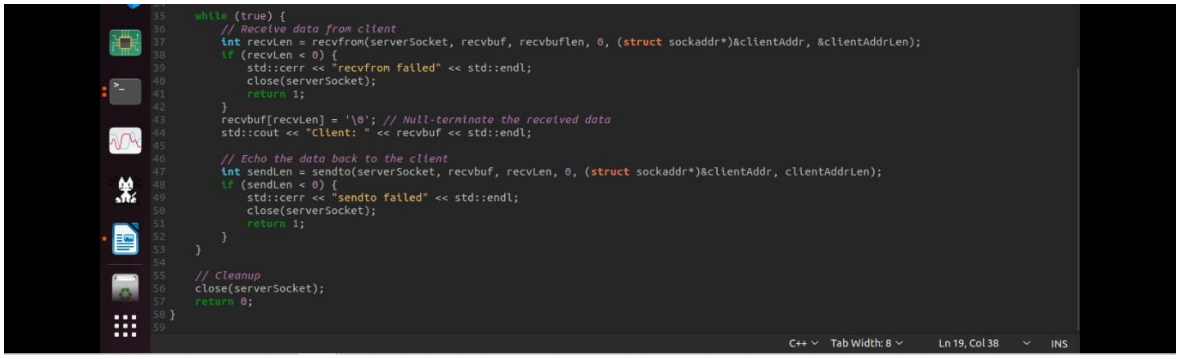
```cpp
    while (true) {
        // Send data to the server
        std::cout << "Enter message to send (or type 'exit' to quit): ";
        std::cin.getline(sendbuf, DEFAULT_BUFLEN);

        if (strcmp(sendbuf, "exit") == 0)
            break;

        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;
        }

        // Receive data from the server
        int recvLen = recvfrom(clientSocket, recvbuf, recvbuflen, 0, nullptr, nullptr);
        if (recvLen < 0) {
            std::cerr << "recvfrom failed" << std::endl;
            close(clientSocket);
            return 1;
        }

        recvbuf[recvLen] = '\0'; // Null-terminate the received data
        std::cout << "Server: " << recvbuf << std::endl;
    }

    // Cleanup
    close(clientSocket);
    return 0;
}
```

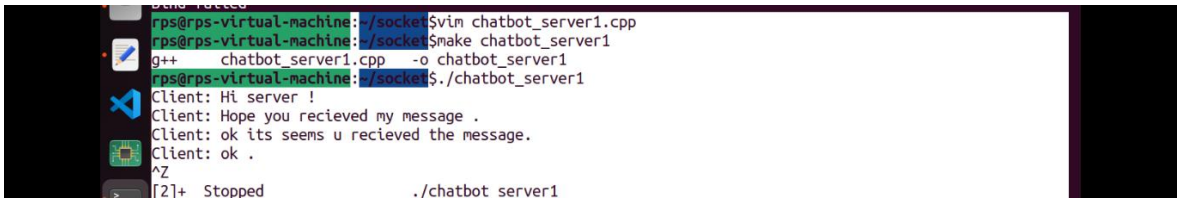


2. ii. Create a chat-bot using UDP Protocol from both – side

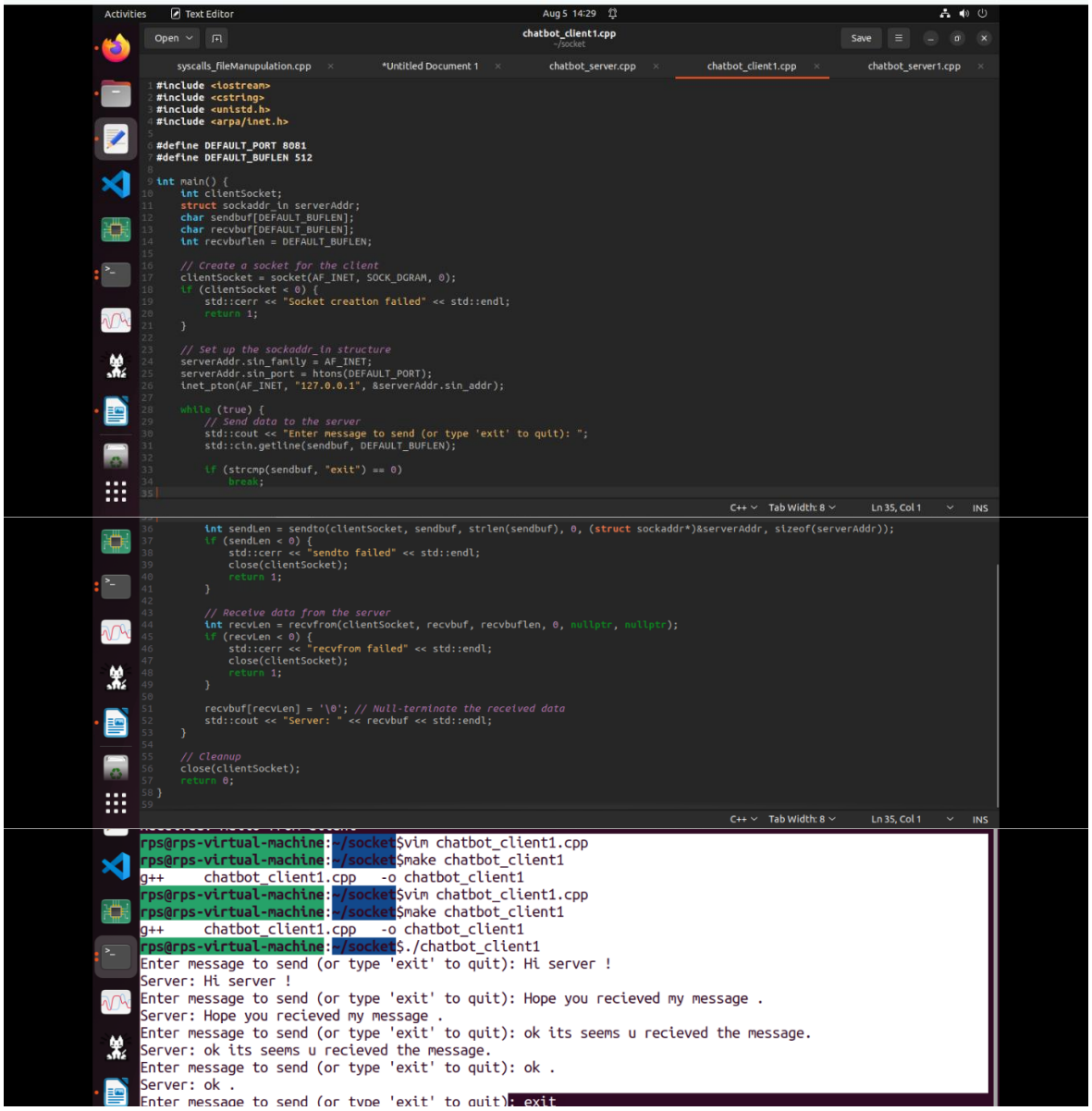
a. server – side







b. client - side



```
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8081
#define DEFAULT_BUFLEN 512

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char sendbuf[DEFAULT_BUFLEN];
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the client
    clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (clientSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);

    while (true) {
        // Send data to the server
        std::cout << "Enter message to send (or type 'exit' to quit): ";
        std::cin.getline(sendbuf, DEFAULT_BUFLEN);

        if (strcmp(sendbuf, "exit") == 0)
            break;

        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;
        }

        // Receive data from the server
        int recvLen = recvfrom(clientSocket, recvbuf, recvbuflen, 0, nullptr, nullptr);
        if (recvLen < 0) {
            std::cerr << "recvfrom failed" << std::endl;
            close(clientSocket);
            return 1;
        }

        recvbuf[recvLen] = '\0'; // Null-terminate the received data
        std::cout << "Server: " << recvbuf << std::endl;
    }

    // Cleanup
    close(clientSocket);
    return 0;
}
```

```
rps@rps-virtual-machine:~/socket$vim chatbot_client1.cpp
rps@rps-virtual-machine:~/socket$make chatbot_client1
g++     chatbot_client1.cpp   -o chatbot_client1
rps@rps-virtual-machine:~/socket$vim chatbot_client1.cpp
rps@rps-virtual-machine:~/socket$make chatbot_client1
g++     chatbot_client1.cpp   -o chatbot_client1
rps@rps-virtual-machine:~/socket$./chatbot_client1
Enter message to send (or type 'exit' to quit): Hi server !
Server: Hi server !
Enter message to send (or type 'exit' to quit): Hope you recieved my message .
Server: Hope you recieved my message .
Enter message to send (or type 'exit' to quit): ok its seems u recieved the message.
Server: ok its seems u recieved the message.
Enter message to send (or type 'exit' to quit): ok .
Server: ok .
Enter message to send (or type 'exit' to quit): exit
```

3. Using UDP Protocol send file from client to server.

a. Server side

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8082
#define DEFAULT_BUFLEN 1024

int main() {
    int serverSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t clientAddrLen = sizeof(clientAddr);
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the server
    serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (serverSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    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;
        }

        // Extract file name from received buffer
        std::string filename(recvbuf, recvLen);

        // Open file to save received content
        std::ofstream outfile(filename, std::ios::out | std::ios::binary);
        if (!outfile) {
            std::cerr << "Failed to open file: " << filename << std::endl;
            close(serverSocket);
            return 1;
        }

        // Receive file content until the end
        while (true) {
            recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
            if (recvLen < 0) {
                std::cerr << "recvfrom failed" << std::endl;
                outfile.close();
                close(serverSocket);
                return 1;
            }
            if (recvLen == 0)
                break;  // End of file

            // Write received data to file
            outfile.write(recvbuf, recvLen);
        }

        std::cout << "File received successfully: " << filename << std::endl;
        outfile.close();
    }

    // Cleanup
    close(serverSocket);
    return 0;
}
```

C++ Tab Width: 8 Ln 12, Col 50 INS

```
rps@rps-virtual-machine:~/socket$vim file_serverUDP.cpp
rps@rps-virtual-machine:~/socket$make file_serverUDP
g++     file_serverUDP.cpp   -o file_serverUDP
rps@rps-virtual-machine:~/socket$./file_serverUDP
Bind failed
rps@rps-virtual-machine:~/socket$vim file_serverUDP.cpp
rps@rps-virtual-machine:~/socket$make file_serverUDP
g++     file_serverUDP.cpp   -o file_serverUDP
rps@rps-virtual-machine:~/socket$./file_serverUDP
UDP Server is listening on port 8082
File received successfully: file_UDP.txt
^Z
[3]+  Stopped                 ./file_serverUDP
rps@rps-virtual-machine:~/socket$
```

b. Client side

Activities Text Editor Aug 5 14:40

Open file_clientUDP.cpp ~/socket Save

*Untitled Document 1 | chatbot_server.cpp | chatbot_client1.cpp | chatbot_server1.cpp | file_serverUDP.cpp | file_clientUDP.cpp

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8082
#define DEFAULT_BUFLEN 1024

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char sendbuf[DEFAULT_BUFLEN];
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the client
    clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (clientSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);

    // Prompt user for file name to send
    std::cout << "Enter file name to send: ";
    std::string filename;
    std::getline(std::cin, filename);

    // Send file name to server
    int sendLen = sendto(clientSocket, filename.c_str(), filename.length(), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    if (sendLen < 0) {
```

Bracket match found on line: 70 C++ Tab Width: 8 Ln 9, Col 13 INS

```cpp
    // Send file name to server
    int sendLen = sendto(clientSocket, filename.c_str(), filename.length(), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    if (sendLen < 0) {
        std::cerr << "sendto failed" << std::endl;
        close(clientSocket);
        return 1;
    }

    // Open file to send its content
    std::ifstream infile(filename, std::ios::in | std::ios::binary);
    if (!infile) {
        std::cerr << "Failed to open file: " << filename << std::endl;
        close(clientSocket);
        return 1;
    }

    // Send file content in chunks
    while (!infile.eof()) {
        infile.read(sendbuf, DEFAULT_BUFLEN);
        sendLen = sendto(clientSocket, sendbuf, infile.gcount(), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
        if (sendLen < 0) {
            std::cerr << "sendto failed" << std::endl;
            infile.close();
            close(clientSocket);
            return 1;
        }
    }

    // Signal end of file
    sendLen = sendto(clientSocket, sendbuf, 0, 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
```

C++ Tab Width: 8 Ln 9, Col 13 INS

```cpp
    // Signal end of file
    sendLen = sendto(clientSocket, sendbuf, 0, 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));

    std::cout << "File sent successfully: " << filename << std::endl;

    // Cleanup
    infile.close();
    close(clientSocket);
    return 0;
}
```

C++ Tab Width: 8 Ln 9, Col 13 INS

```
rps@rps-virtual-machine:~/socket$vim file_clientUDP.cpp
rps@rps-virtual-machine:~/socket$vim file_UDP.txt
rps@rps-virtual-machine:~/socket$make file_clientUDP
g++     file_clientUDP.cpp   -o file_clientUDP
rps@rps-virtual-machine:~/socket$vim file_clientUDP.cpp
```

4. Implement a UDP server in C++ for file transfer with file type information.

a. Server side

Activities Text Editor Aug 5 15:13

Open filetype_serverUDP.cpp ~/socket Save

chatbot_server.cpp chatbot_client1.cpp chatbot_server1.cpp file_serverUDP.cpp file_clientUDP.cpp filetype_serverUDP.cpp

```cpp
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <arpa/inet.h>
#include <cstring>

#define DEFAULT_PORT 8083
#define DEFAULT_BUFLEN 1024

struct FileTransferInfo {
    char filename[256];
    char filetype[10];
};

int main() {
    int serverSocket;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t clientAddrLen = sizeof(clientAddr);
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the server
    serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (serverSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
```

C++ Tab Width: 8 Ln 1, Col 1 INS

```cpp
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
        std::cerr << "Bind failed" << std::endl;
        close(serverSocket);
        return 1;
    }

    // Receive file metadata from client
    std::cout << "UDP Server is listening on port " << DEFAULT_PORT << std::endl;

    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;
        }

        // Deserialize file transfer info
        FileTransferInfo fileInfo;
        memcpy(&fileInfo, recvbuf, sizeof(FileTransferInfo));

        std::cout << "Receiving file: " << fileInfo.filename << " (" << fileInfo.filetype << ")" << std::endl;

        // Open file to save received content
        std::ofstream outfile(fileInfo.filename, std::ios::out | std::ios::binary);
        if (!outfile) {
            std::cerr << "Failed to open file: " << fileInfo.filename << std::endl;
            close(serverSocket);
            return 1;
        }
```

```cpp
        // Receive file content until the end
        while (true) {
            recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
            if (recvLen < 0) {
                std::cerr << "recvfrom failed" << std::endl;
                outfile.close();
                close(serverSocket);
                return 1;
            }
            if (recvLen == 0)
                break;  // End of file

            // Write received data to file
            outfile.write(recvbuf, recvLen);
        }

        std::cout << "File received successfully: " << fileInfo.filename << std::endl;
        outfile.close();
    }

    // Cleanup
    close(serverSocket);
    return 0;
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

```
rps@rps-virtual-machine:~/socket$rm filetype_serverUDP.cpp
rps@rps-virtual-machine:~/socket$vim filetype_serverUDP.cpp
rps@rps-virtual-machine:~/socket$make filetype_serverUDP
g++     filetype_serverUDP.cpp   -o filetype_serverUDP
rps@rps-virtual-machine:~/socket$./filetype_serverUDP
UDP Server is listening on port 8083
Receiving file: file_UDP.txt (txt)
File received successfully: file_UDP.txt
```

b. Client side

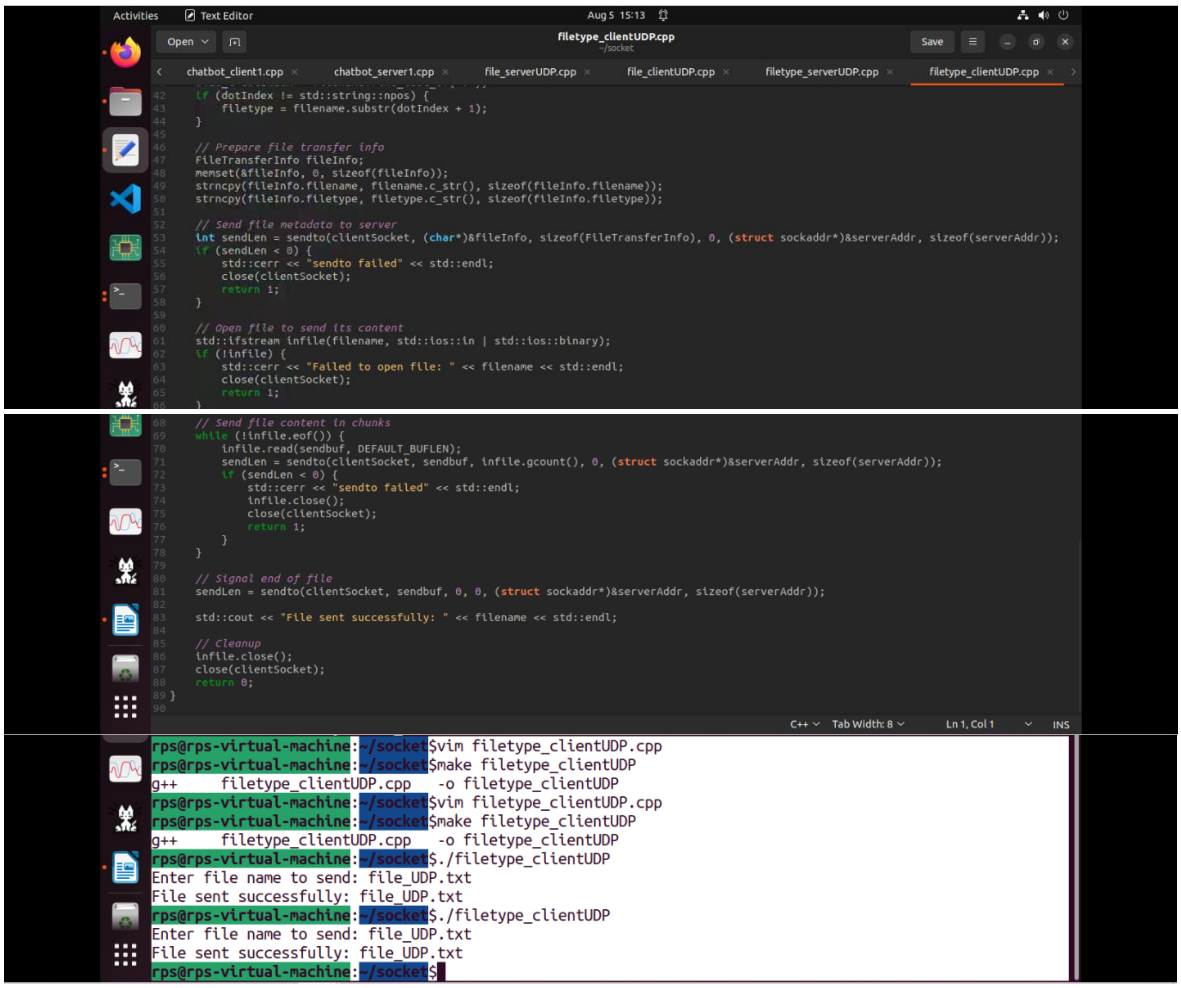
Activities Text Editor Aug 5 15:13

Open filetype_clientUDP.cpp ~/socket Save

chatbot_client1.cpp | chatbot_server1.cpp | file_serverUDP.cpp | file_clientUDP.cpp | filetype_serverUDP.cpp | filetype_clientUDP.cpp

```cpp
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <arpa/inet.h>
#include <cstring>

#define DEFAULT_PORT 8083
#define DEFAULT_BUFLEN 1024

struct FileTransferInfo {
    char filename[256];
    char filetype[10];
};

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char sendbuf[DEFAULT_BUFLEN];
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    // Create a socket for the client
    clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (clientSocket < 0) {
        std::cerr << "Socket creation failed" << std::endl;
        return 1;
    }

    // Set up the sockaddr_in structure
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(DEFAULT_PORT);
    inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);

    // Prompt user for file name to send
    std::cout << "Enter file name to send: ";
```

Loading file "/home/rps/socket/filetype_clientUDP.cpp"... C++ Tab Width: 8 Ln 1, Col 1 INS



5. Problem statement :

1. **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.

a. Server side

```
#include <iostream>
#include <string>
#include <cstring>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

#define SERVER_PORT 12345
#define BUFFER_SIZE 1024

void udp_server() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len;
    char buffer[BUFFER_SIZE];

    // Create a UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("Error creating socket");
        return;
    }

    // Bind the socket to the specified port
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
        perror("Error binding socket");
        close(sockfd);
        return;
    }

    std::cout << "UDP server is listening on port " << SERVER_PORT << "..." << std::endl;
```

```
    std::cout << "UDP server is listening on port " << SERVER_PORT << "..." << std::endl;

    while (true) {
        // Receive message and address from client
        client_len = sizeof(client_addr);
        ssize_t recv_len = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr *)&client_addr, &client_len);

        if (recv_len == -1) {
            perror("Error receiving message");
            continue;
        }

        buffer[recv_len] = '\0';  // Null-terminate the received data

        // Print received message and client's address and port
        std::cout << "Received message from " << inet_ntoa(client_addr.sin_addr) << ":" << ntohs(client_addr.sin_port) << ": " << buffer << std::endl;

        // Send acknowledgment message back to the client
        const char *ack_message = "Message received";
        ssize_t send_len = sendto(sockfd, ack_message, strlen(ack_message), 0, (struct sockaddr *)&client_addr, client_len);

        if (send_len == -1) {
            perror("Error sending acknowledgment");
            continue;
        }
    }
```

```
    close(sockfd);
}

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

```
rps@rps-virtual-machine:~/socket$vim serverUDP_chat.cpp
rps@rps-virtual-machine:~/socket$make serverUDP_chat
g++     serverUDP_chat.cpp    -o serverUDP_chat
rps@rps-virtual-machine:~/socket$./serverUDP_chat
UDP server is listening on port 12345...
Received message from 127.0.0.1:40660: Hello, Server!
```

b. Client - side

clientUDP_chat.cpp

```cpp
#include <iostream>
#include <cstring>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

#define SERVER_IP "127.0.0.1"  // Replace with the IP address of your server
#define SERVER_PORT 12345
#define BUFFER_SIZE 1024

void udp_client() {
    int sockfd;
    struct sockaddr_in server_addr;
    socklen_t server_len;
    char buffer[BUFFER_SIZE];

    // Create a UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("Error creating socket");
        return;
    }

    // Configure server address
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    if (inet_aton(SERVER_IP, &server_addr.sin_addr) == 0) {
        perror("Invalid IP address");
        close(sockfd);
        return;
    }

    // Send a predefined message to the server
    const char *message = "Hello, Server!";
    ssize_t send_len = sendto(sockfd, message, strlen(message), 0, (struct sockaddr *)&server_addr, sizeof(server_addr));
```

```cpp
    if (send_len == -1) {
        perror("Error sending message");
        close(sockfd);
        return;
    }

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

    // Receive acknowledgment from the server
    server_len = sizeof(server_addr);
    ssize_t recv_len = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr *)&server_addr, &server_len);

    if (recv_len == -1) {
        perror("Error receiving acknowledgment");
        close(sockfd);
        return;
    }

    buffer[recv_len] = '\0';  // Null-terminate the received data

    // Print acknowledgment message from the server
    std::cout << "Received acknowledgment from server: " << buffer << std::endl;

    close(sockfd);
}

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

```
rps@rps-virtual-machine:~/socket$vim clientUDP_chat.cpp
rps@rps-virtual-machine:~/socket$make clientUDP_chat
g++     clientUDP_chat.cpp    -o clientUDP_chat
rps@rps-virtual-machine:~/socket$./clientUDP_chat
Message sent to server: Hello, Server!
Received acknowledgment from server: Message received
rps@rps-virtual-machine:~/socket$
```