

Day – 13 LSP Assignment Task – 1

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

Objective:

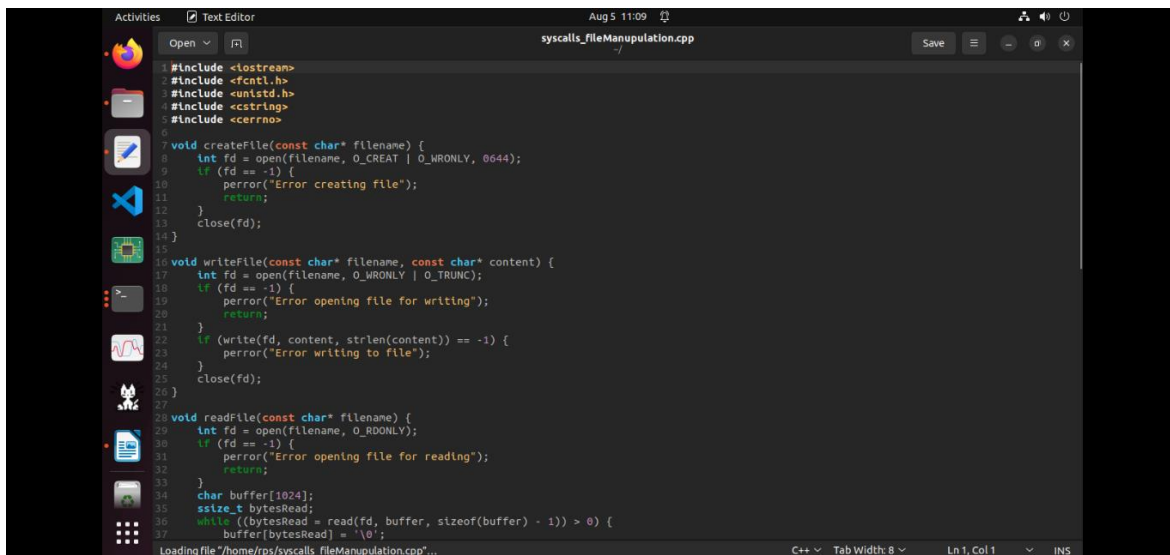
Create a C++ program that performs file manipulation using Linux system calls. The program should be able to:

- 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).



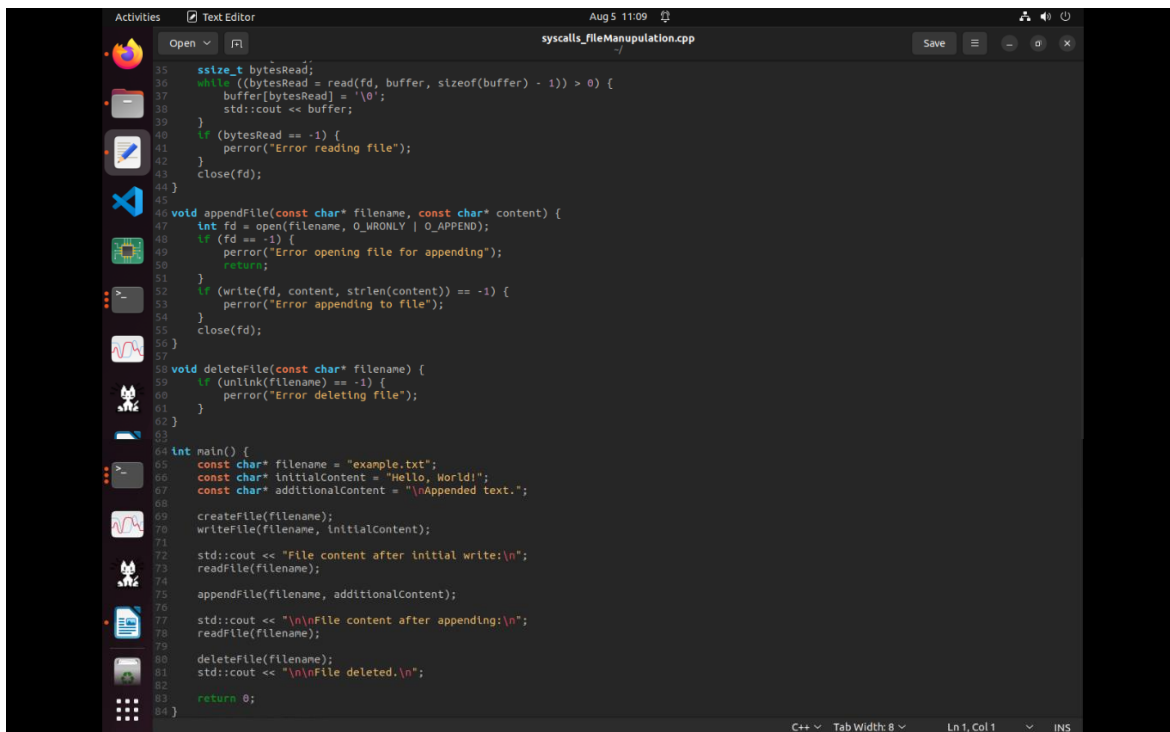
The screenshot shows a C++ program named `syscalls_fileManipulation.cpp` in a text editor. The program is modular, with functions for creating, writing, reading, and deleting files. It includes necessary headers and uses `perror` for error handling.

```
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <cstring>
#include <cerrno>

7 void createFile(const char* filename) {
8     int fd = open(filename, O_CREAT | O_WRONLY, 0644);
9     if (fd == -1) {
10         perror("Error creating file");
11         return;
12     }
13     close(fd);
14 }

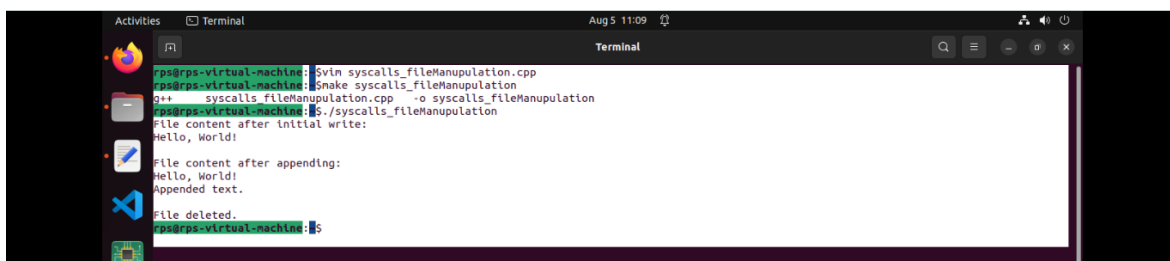
15
16 void writeFile(const char* filename, const char* content) {
17     int fd = open(filename, O_WRONLY | O_TRUNC);
18     if (fd == -1) {
19         perror("Error opening file for writing");
20         return;
21     }
22     if (write(fd, content, strlen(content)) == -1) {
23         perror("Error writing to file");
24     }
25     close(fd);
26 }

27
28 void readFile(const char* filename) {
29     int fd = open(filename, O_RDONLY);
30     if (fd == -1) {
31         perror("Error opening file for reading");
32         return;
33     }
34     char buffer[1024];
35     ssize_t bytesRead;
36     while ((bytesRead = read(fd, buffer, sizeof(buffer) - 1)) > 0) {
37         buffer[bytesRead] = '\0';
38     }
39 }
```



```
1  syscalls_fileManipulation.cpp
2
3  #include <iostream>
4  #include <string>
5  #include <unistd.h>
6  #include <arpa/inet.h>
7
8  #define DEFAULT_PORT 8080
9  #define DEFAULT_BUFLN 512
10
11 int main() {
12     int serverSocket;
13     struct sockaddr_in serverAddr, clientAddr;
14     socklen_t clientAddrLen = sizeof(clientAddr);
15     char recvbuf[DEFAULT_BUFLN];
16     int recvbuflen = DEFAULT_BUFLN;
17
18     // Create a socket for the server
19     serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
20     if (serverSocket < 0) {
21         std::cerr << "Socket creation failed" << std::endl;
22         return 1;
23     }
24
25     // Set up the sockaddr_in structure
26     serverAddr.sin_family = AF_INET;
27     serverAddr.sin_port = htons(DEFAULT_PORT);
28     serverAddr.sin_addr.s_addr = INADDR_ANY;
29
30     while (1) {
31         ssize_t bytesRead;
32         while ((bytesRead = read(serverSocket, recvbuf, sizeof(recvbuf) - 1)) > 0) {
33             buffer[bytesRead] = '\0';
34             std::cout << buffer;
35         }
36         if (bytesRead == -1) {
37             perror("Error reading file");
38         }
39         close(serverSocket);
40     }
41
42     void appendFile(const char* filename, const char* content) {
43         int fd = open(filename, O_WRONLY | O_APPEND);
44         if (fd == -1) {
45             perror("Error opening file for appending");
46             return;
47         }
48         if (write(fd, content, strlen(content)) == -1) {
49             perror("Error appending to file");
50         }
51         close(fd);
52     }
53
54     void deleteFile(const char* filename) {
55         if (unlink(filename) == -1) {
56             perror("Error deleting file");
57         }
58     }
59
60     int main() {
61         const char* filename = "example.txt";
62         const char* initialContent = "Hello, World!";
63         const char* additionalContent = "\nAppended text.";
64
65         createFile(filename);
66         writeFile(filename, initialContent);
67
68         std::cout << "File content after initial write:\n";
69         readFile(filename);
70
71         appendFile(filename, additionalContent);
72
73         std::cout << "\n\nFile content after appending:\n";
74         readFile(filename);
75
76         deleteFile(filename);
77         std::cout << "\n\nFile deleted.\n";
78
79         return 0;
80     }
81 }
```

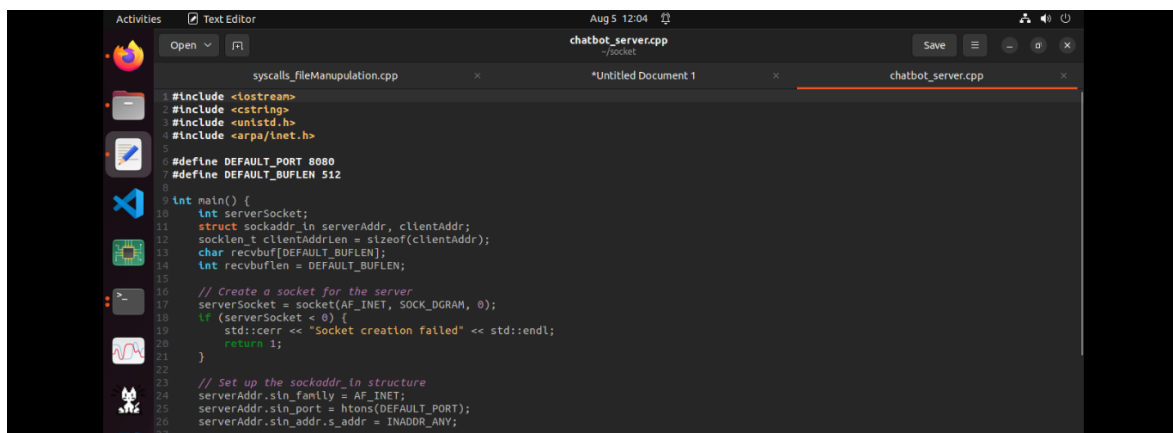
Execution:



```
1  Terminal
2
3  rps@rps-virtual-machine:~$ g++ syscalls_fileManipulation.cpp
4  rps@rps-virtual-machine:~$ ./syscalls_fileManipulation
5  File content after initial write:
6  Hello, World!
7
8  File content after appending:
9  Hello, World!
10 Appended text.
11
12 File deleted.
13 rps@rps-virtual-machine:~$
```

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

a. Server side



```
1  chatbot_server.cpp
2
3  #include <iostream>
4  #include <string>
5  #include <unistd.h>
6  #include <arpa/inet.h>
7
8  #define DEFAULT_PORT 8080
9  #define DEFAULT_BUFLN 512
10
11 int main() {
12     int serverSocket;
13     struct sockaddr_in serverAddr, clientAddr;
14     socklen_t clientAddrLen = sizeof(clientAddr);
15     char recvbuf[DEFAULT_BUFLN];
16     int recvbuflen = DEFAULT_BUFLN;
17
18     // Create a socket for the server
19     serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
20     if (serverSocket < 0) {
21         std::cerr << "Socket creation failed" << std::endl;
22         return 1;
23     }
24
25     // Set up the sockaddr_in structure
26     serverAddr.sin_family = AF_INET;
27     serverAddr.sin_port = htons(DEFAULT_PORT);
28     serverAddr.sin_addr.s_addr = INADDR_ANY;
29
30     while (1) {
31         ssize_t bytesRead;
32         while ((bytesRead = read(serverSocket, recvbuf, sizeof(recvbuf) - 1)) > 0) {
33             buffer[bytesRead] = '\0';
34             std::cout << buffer;
35         }
36         if (bytesRead == -1) {
37             perror("Error reading file");
38         }
39         close(serverSocket);
40     }
41
42     void appendFile(const char* filename, const char* content) {
43         int fd = open(filename, O_WRONLY | O_APPEND);
44         if (fd == -1) {
45             perror("Error opening file for appending");
46             return;
47         }
48         if (write(fd, content, strlen(content)) == -1) {
49             perror("Error appending to file");
50         }
51         close(fd);
52     }
53
54     void deleteFile(const char* filename) {
55         if (unlink(filename) == -1) {
56             perror("Error deleting file");
57         }
58     }
59
60     int main() {
61         const char* filename = "example.txt";
62         const char* initialContent = "Hello, World!";
63         const char* additionalContent = "\nAppended text.";
64
65         createFile(filename);
66         writeFile(filename, initialContent);
67
68         std::cout << "File content after initial write:\n";
69         readFile(filename);
70
71         appendFile(filename, additionalContent);
72
73         std::cout << "\n\nFile content after appending:\n";
74         readFile(filename);
75
76         deleteFile(filename);
77         std::cout << "\n\nFile deleted.\n";
78
79         return 0;
80     }
81 }
```

```

28 // Bind the socket
29 if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
30     std::cerr << "Bind failed" << std::endl;
31     close(serverSocket);
32     return 1;
33 }
34
35 // Receive data
36 while (true) {
37     int recrlen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
38     if (recrlen < 0) {
39         std::cerr << "recvfrom failed" << std::endl;
40         close(serverSocket);
41         return 1;
42     }
43
44     recvbuf[recrlen] = '\0'; // Null-terminate the received data
45     std::cout << "Received: " << recvbuf << std::endl;
46
47     // Echo the data back to the client
48     int sendlen = sendto(serverSocket, recvbuf, recrlen, 0, (struct sockaddr*)&clientAddr, clientAddrLen);
49     if (sendlen < 0) {
50         std::cerr << "sendto failed" << std::endl;
51         close(serverSocket);
52         return 1;
53     }
54 }
55
56 // Cleanup
57 close(serverSocket);
58 return 0;
59 }

```

Activities Terminal Aug 5 12:03

Terminal	Terminal	Terminal	Terminal	Terminal
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

Activities Text Editor Aug 5 12:04

Open chatbot_client1.cpp

```

1 #include <iostream>
2 #include <cstring>
3 #include <unistd.h>
4 #include <arpa/inet.h>
5
6 #define DEFAULT_PORT 8081
7 #define DEFAULT_BUFLN 512
8
9 int main() {
10     int clientSocket;
11     struct sockaddr_in serverAddr;
12     char sendbuf[DEFAULT_BUFLN];
13     char recvbuf[DEFAULT_BUFLN];
14     int recvbuflen = DEFAULT_BUFLN;
15
16     // Create a socket for the client
17     clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
18     if (clientSocket < 0) {
19         std::cerr << "Socket creation failed" << std::endl;
20         return 1;
21     }
22
23     // Set up the sockaddr_in structure
24     serverAddr.sin_family = AF_INET;
25     serverAddr.sin_port = htons(DEFAULT_PORT);
26     inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);
27
28     while (true) {
29         // Send data to the server
30         std::cout << "Enter message to send (or type 'exit' to quit): ";
31         std::cin.getline(sendbuf, DEFAULT_BUFLN);
32
33         if (strcmp(sendbuf, "exit") == 0)
34             break;
35
36         int sendlen = sendto(clientSocket, sendbuf, strlen(sendbuf), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
37         if (sendlen < 0) {
38             std::cerr << "sendto failed" << std::endl;
39             close(clientSocket);
40             return 1;
41         }
42
43         // Receive data from the server
44         int recrlen = recvfrom(clientSocket, recvbuf, recvbuflen, 0, nullptr, nullptr);
45         if (recrlen < 0) {
46             std::cerr << "recvfrom failed" << std::endl;
47             close(clientSocket);
48             return 1;
49         }
50
51         recvbuf[recrlen] = '\0'; // Null terminate the received data
52         std::cout << "Server: " << recvbuf << std::endl;
53     }
54
55     // Cleanup
56     close(clientSocket);
57     return 0;
58 }
59

```

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

```
Activities Terminal Aug 5 12:03
Terminal
rps@rps-virtual-machine:~$ cd socket
rps@rps-virtual-machine:~/socket$ vim chatbot_client.cpp
rps@rps-virtual-machine:~/socket$ make chatbot_client
g++ chatbot_client.cpp -o chatbot_client
rps@rps-virtual-machine:~/socket$ ./chatbot_client
Sent: Hello from client
Received: Hello from client
```

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

```
Activities Text Editor Aug 5 14:28
chatbot_server1.cpp
syscalls_fileManipulation.cpp x *Untitled Document 1 x chatbot_server.cpp x chatbot_client1.cpp x chatbot_server1.cpp x
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define DEFAULT_PORT 8081
#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;

    // 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) {
```

```
        // Receive data from client
        int recrlen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
        if (recrlen < 0) {
            std::cerr << "recvfrom failed" << std::endl;
            close(serverSocket);
            return 1;
        }
        recvbuf[recrlen] = '\0'; // Null-terminate the received data
        std::cout << "Client: " << recvbuf << std::endl;

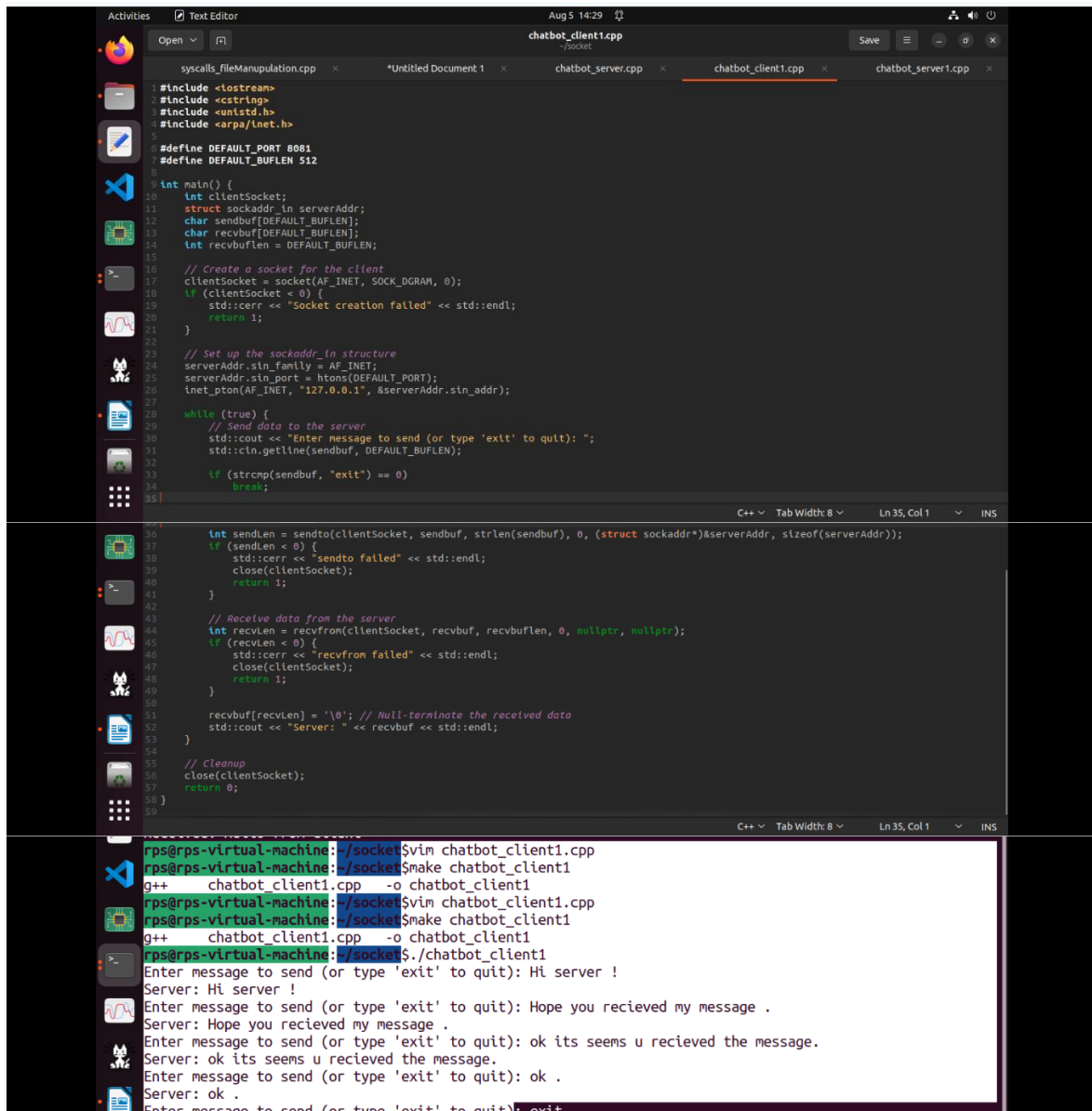
        // Echo the data back to the client
        int sendlen = sendto(serverSocket, recvbuf, recrlen, 0, (struct sockaddr*)&clientAddr, clientAddrLen);
        if (sendlen < 0) {
            std::cerr << "sendto failed" << std::endl;
            close(serverSocket);
            return 1;
        }
    }

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

```

rps@rps-virtual-machine:~/socket$ vim chatbot_server1.cpp
rps@rps-virtual-machine:~/socket$ make chatbot_server1
g++ chatbot_server1.cpp -o chatbot_server1
rps@rps-virtual-machine:~/socket$ ./chatbot_server1
Client: Hi server !
Client: Hope you recieved my message .
Client: ok its seems u recieved the message.
Client: ok .
^Z
[2]+  Stopped                  ./chatbot server1
```

b. client - side



The screenshot shows a C++ IDE with the file `chatbot_client1.cpp` open. The code defines a client socket, sets up the `sockaddr_in` structure, and enters a loop to send and receive data from the server. The execution output at the bottom shows the client sending messages to the server and receiving responses.

```
#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::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;
}
```

Execution output:

```
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



The screenshot shows a C++ IDE with the file `file_serverUDP.cpp` open. The code defines a server socket, sets up the `sockaddr_in` structure, and enters a loop to receive data from the client.

```
#include <iostream>
#include <cstring>
#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;
    }
}
```

```
39 while (true) {
40     int recvlen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
41     if (recvlen < 0) {
42         std::cerr << "recvfrom failed" << std::endl;
43         close(serverSocket);
44         return 1;
45     }
46     // Extract file name from received buffer
47     std::string filename(recvbuf, recvlen);
48     // Open file to save received content
49     std::ofstream outfile(filename, std::ios::out | std::ios::binary);
50     if (!outfile) {
51         std::cerr << "Failed to open file: " << filename << std::endl;
52         close(serverSocket);
53         return 1;
54     }
55     // Receive file content until the end
56     while (true) {
57         recvlen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
58         if (recvlen < 0) {
59             std::cerr << "recvfrom failed" << std::endl;
60             outfile.close();
61             close(serverSocket);
62             return 1;
63         }
64         if (recvlen == 0)
65             break; // End of file
66     }
67     // Write received data to file
68     outfile.write(recvbuf, recvlen);
69     std::cout << "File received successfully: " << filename << std::endl;
70     outfile.close();
71 }
72 // Cleanup
73 close(serverSocket);
74 return 0;
75 }
```

```
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

```
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) {
        std::cerr << "Send failed" << std::endl;
        return 1;
    }
}
```



```
33 // Send file name to server
34 int sendlen = sendto(clientSocket, filename.c_str(), filename.length(), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
35 if (sendlen < 0) {
36     std::cerr << "sendto failed" << std::endl;
37     close(clientSocket);
38     return 1;
39 }
40 // Open file to send its content
41 std::ifstream infile(filename, std::ios::in | std::ios::binary);
42 if (!infile) {
43     std::cerr << "Failed to open file: " << filename << std::endl;
44     close(clientSocket);
45     return 1;
46 }
47 // Send file content in chunks
48 while (!infile.eof()) {
49     infile.read(sendbuf, DEFAULT_BUFLEN);
50     sendlen = sendto(clientSocket, sendbuf, infile.gcount(), 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
51     if (sendlen < 0) {
52         std::cerr << "sendto failed" << std::endl;
53         infile.close();
54         close(clientSocket);
55         return 1;
56     }
57 }
58 // Signal end of file
59 sendlen = sendto(clientSocket, sendbuf, 0, 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
60 if (sendlen < 0) {
61     std::cerr << "Signal end of file failed" << std::endl;
62     close(clientSocket);
63     return 1;
64 }
65 // Cleanup
66 infile.close();
67 close(clientSocket);
68 return 0;
69 }
70 }
71 }
```

```
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

```
1 #include <iostream>
2 #include <fstream>
3 #include <unistd.h>
4 #include <arpa/inet.h>
5 #include <cstring>
6
7 #define DEFAULT_PORT 8083
8 #define DEFAULT_BUFLEN 1024
9
10 struct FileTransferInfo {
11     char filename[256];
12     char filetype[10];
13 };
14
15 int main() {
16     int serverSocket;
17     struct sockaddr_in serverAddr, clientAddr;
18     socklen_t clientAddrLen = sizeof(clientAddr);
19     char recvbuf[DEFAULT_BUFLEN];
20     int recvbuflen = DEFAULT_BUFLEN;
21
22     // Create a socket for the server
23     serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
24     if (serverSocket < 0) {
25         std::cerr << "Socket creation failed" << std::endl;
26         return 1;
27     }
28
29     // Set up the sockaddr_in structure
30     serverAddr.sin_family = AF_INET;
31     serverAddr.sin_port = htons(DEFAULT_PORT);
32     serverAddr.sin_addr.s_addr = INADDR_ANY;
33
34     // Bind the socket
35     if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
```

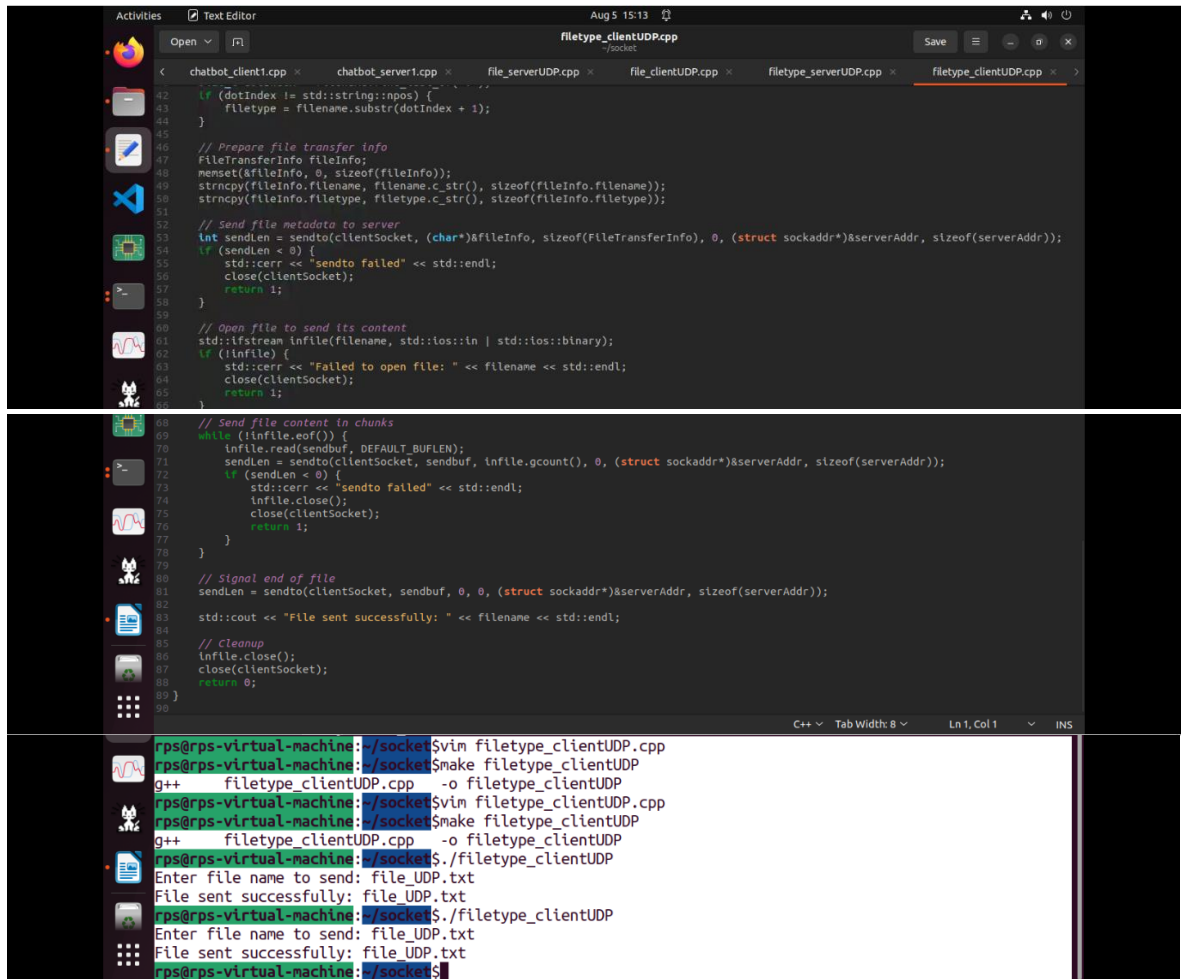
```
35 if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
36     std::cerr << "Bind failed" << std::endl;
37     close(serverSocket);
38     return 1;
39 }
40
41 // Receive file metadata from client
42 std::cout << "UDP Server is listening on port " << DEFAULT_PORT << std::endl;
43
44 while (true) {
45     int recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
46     if (recvLen < 0) {
47         std::cerr << "recvfrom failed" << std::endl;
48         close(serverSocket);
49         return 1;
50     }
51
52     // Deserialize file transfer info
53     FileTransferInfo fileInfo;
54     memcpy(&fileInfo, recvbuf, sizeof(FileTransferInfo));
55
56     std::cout << "Receiving file: " << fileInfo.filename << " (" << fileInfo.filetype << ")" << std::endl;
57
58     // Open file to save received content
59     std::ofstream outfile(fileInfo.filename, std::ios::out | std::ios::binary);
60     if (!outfile) {
61         std::cerr << "Failed to open file: " << fileInfo.filename << std::endl;
62         close(serverSocket);
63         return 1;
64     }
65
66     // Receive file content until the end
67     while (true) {
68         recvLen = recvfrom(serverSocket, recvbuf, recvbuflen, 0, (struct sockaddr*)&clientAddr, &clientAddrLen);
69         if (recvLen < 0) {
70             std::cerr << "recvfrom failed" << std::endl;
71             outfile.close();
72             close(serverSocket);
73             return 1;
74         }
75         if (recvLen == 0)
76             break; // End of file
77
78         // Write received data to file
79         outfile.write(recvbuf, recvLen);
80     }
81
82     std::cout << "File received successfully: " << fileInfo.filename << std::endl;
83     outfile.close();
84 }
85
86 // Cleanup
87 close(serverSocket);
88 return 0;
89 }
90
```

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

```
1 #include <iostream>
2 #include <fstream>
3 #include <unistd.h>
4 #include <arpa/inet.h>
5 #include <cstring>
6
7 #define DEFAULT_PORT 8083
8 #define DEFAULT_BUFLen 1024
9
10 struct FileTransferInfo {
11     char filename[250];
12     char filetype[10];
13 };
14
15 int main() {
16     int clientSocket;
17     struct sockaddr_in serverAddr;
18     char sendbuf[DEFAULT_BUFLen];
19     char recvbuf[DEFAULT_BUFLen];
20     int recvbuflen = DEFAULT_BUFLen;
21
22     // Create a socket for the client
23     clientSocket = socket(AF_INET, SOCK_DGRAM, 0);
24     if (clientSocket < 0) {
25         std::cerr << "Socket creation failed" << std::endl;
26         return 1;
27     }
28
29     // Set up the sockaddr_in structure
30     serverAddr.sin_family = AF_INET;
31     serverAddr.sin_port = htons(DEFAULT_PORT);
32     inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);
33
34     // Prompt user for file name to send
35     std::cout << "Enter file name to send: ";
36     Loading file "/home/rps/socket/filetype_clientUDP.cpp" ...

```

```
filetypes_clientUDP.cpp
// Prepare file transfer info
FileTransferInfo fileInfo;
memset(&fileInfo, 0, sizeof(fileInfo));
strcpy(fileInfo.filename, filename.c_str(), sizeof(fileInfo.filename));
strcpy(fileInfo.filetype, filetype.c_str(), sizeof(fileInfo.filetype));

// Send file metadata to server
int sendlen = sendto(clientSocket, (char*)&fileInfo, sizeof(FileTransferInfo), 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_BUFSIZE);
    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));
std::cout << "File sent successfully: " << filename << std::endl;

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

```
rps@rps-virtual-machine:~/socket$ vim filetype_clientUDP.cpp
rps@rps-virtual-machine:~/socket$ make filetype_clientUDP
g++ filetype_clientUDP.cpp -o filetype_clientUDP
rps@rps-virtual-machine:~/socket$ vim filetype_clientUDP.cpp
rps@rps-virtual-machine:~/socket$ make filetype_clientUDP
g++ filetype_clientUDP.cpp -o filetype_clientUDP
rps@rps-virtual-machine:~/socket$ ./filetype_clientUDP
Enter file name to send: file_UDP.txt
File sent successfully: file_UDP.txt
rps@rps-virtual-machine:~/socket$ ./filetype_clientUDP
Enter file name to send: file_UDP.txt
File sent successfully: file_UDP.txt
rps@rps-virtual-machine:~/socket$
```

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.

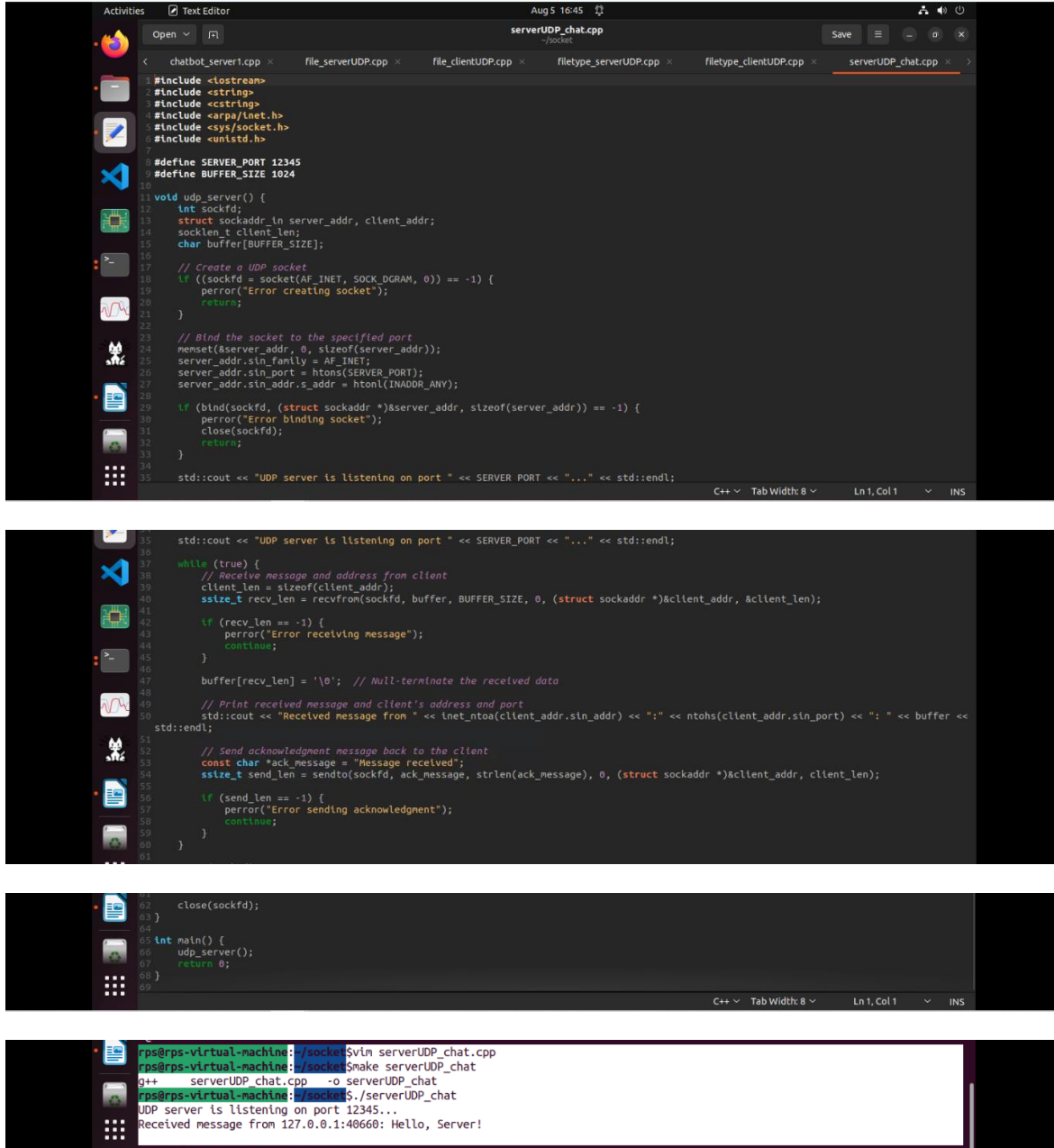
Send a predefined message (e.g., "Hello, Server!") to the server.

Wait for an acknowledgment from the server.

Print the acknowledgment message to the console.

Ensure proper error handling and resource cleanup.

a. Server side



The image shows a C++ server program in a text editor and its execution in a terminal. The program is named `serverUDP_chat.cpp` and is located in the `~/socket` directory. It defines a server port of 12345 and a buffer size of 1024. The `udp_server()` function creates a UDP socket, binds it to the specified port, and enters a loop to receive and send messages. The `main()` function calls `udp_server()` and returns 0.

```
#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;

    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;
}
```

The terminal output shows the server starting and receiving a message:

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

b. Client - side

```
clientUDP_chat.cpp
~/socket

file_serverUDP.cpp x file_clientUDP.cpp x filetype_serverUDP.cpp x filetype_clientUDP.cpp x serverUDP_chat.cpp x clientUDP_chat.cpp x

#include <iostream>
#include <string>
#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));

    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$ g++ 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$
```