Assignment 3 documentation

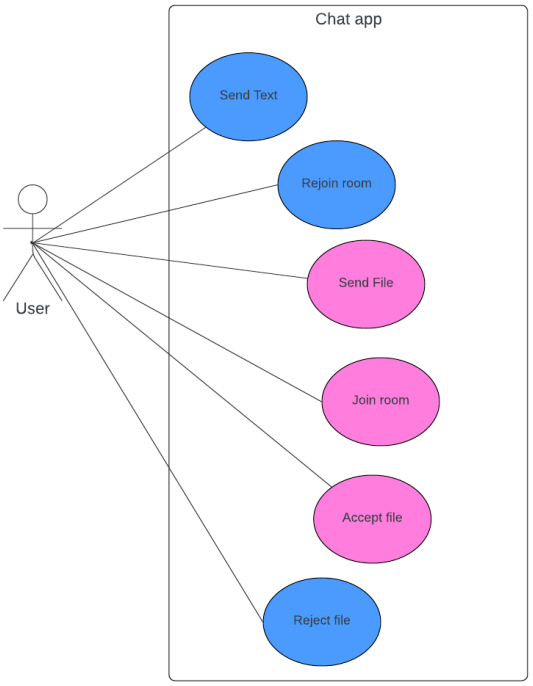
General program description(Server):

When the server is started it creates a map with vector<SOCKET>, these vectors represent rooms , after that server starts listening port 8080, server also starts constantly waiting for new connections with the user(for each user new thread is created). Each thread calls handleClient(clientSocket) function. Before entering infinite loop this function accepts the number of room that current client wants to join, and pushes client socket to the corresponding vector, then it enters infinite loop for constant message receiving, instead of normal message client may choose to input special command, like: FILE file.txt(for broadcasting file) or REJOIN roomName(for joining other room). Each received message is composed into a Message struct which contains: message itself, SOCKET of the sender, name of the room to which this message belongs,and type of that message(file/text). Each message is placed into a message queue which is then broadcasted to all users of the concrete room except the sender of that message. If the message is a file then the server downloads this file from the client and only after that broadcasts the file to other users of the concrete room. After broadcasting, the file still remains on the server for the purposes of selling data of users.

General program description(Client):

When the client is started it tries to connect to port 8080.
Client creates two threads: one receives messages and another sends messages. User can send both normal messages, and files , if he wishes to send a file he must type: FILE filename.txt.

Use case diagram:



Communication protocol:

Let's start from the implementation of send/receive message logic for text.

Send message mechanic is created by:

1. sending 4 bytes(size of int, checked in the app) as the size of the message itself. So on the other side the appropriate buffer was created.

2.message itself is sent.

3. data and received and stored in the created buffer and can be accessed.

For files it is mostly the same, however because of streamsize type is used we are sending not 4 bytes , but 8 (size of streamsize)

commands

Broadcast text:

client sends text as described above

server adds message to message queue

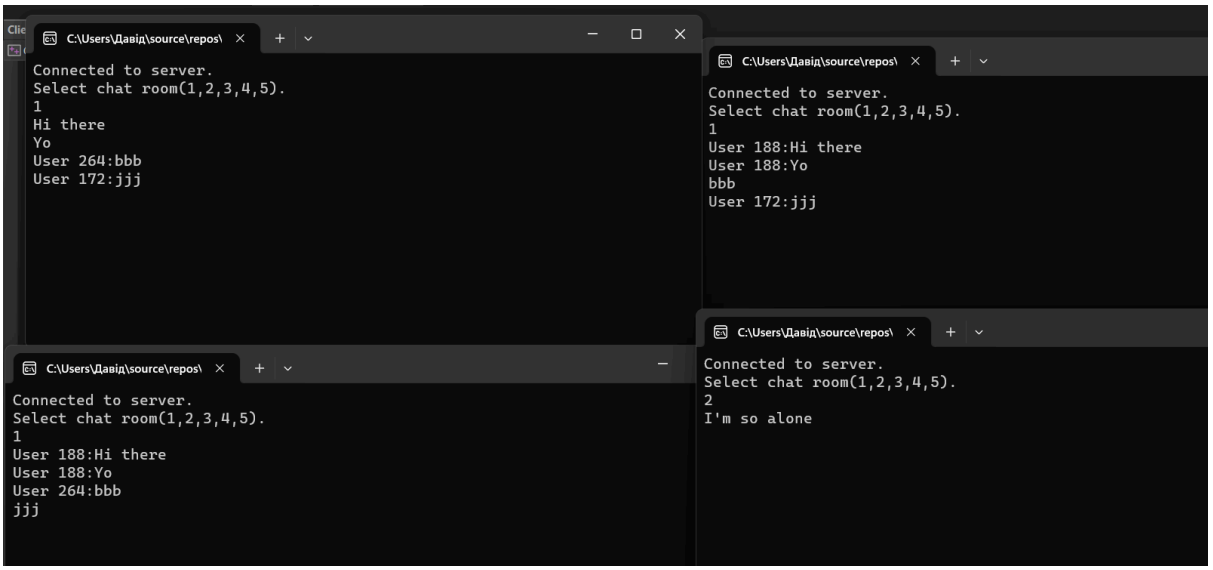
server sends message to all client that are in one room with sender as described above

Broadcast file:
The same as the text, but client needs to write message in the next format: FILE name.txt

Change room:
User must send message in the next format: REJOIN roomNumber

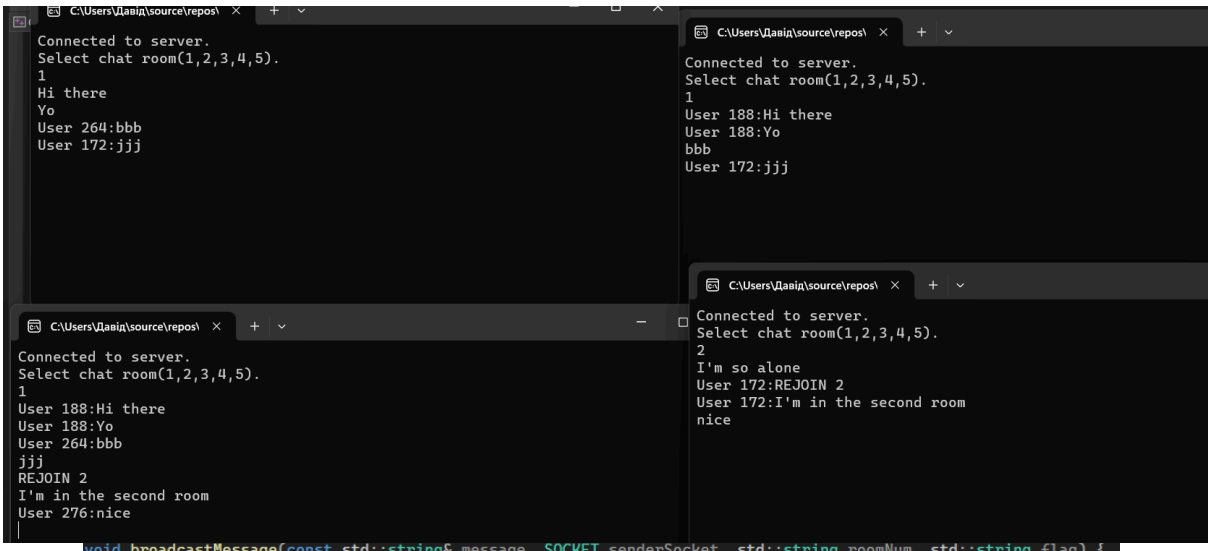
Decline/accept file:
When some user broadcasts file all users can reject or accept file by entering “y” or “n” two times

Use cases screenshots:



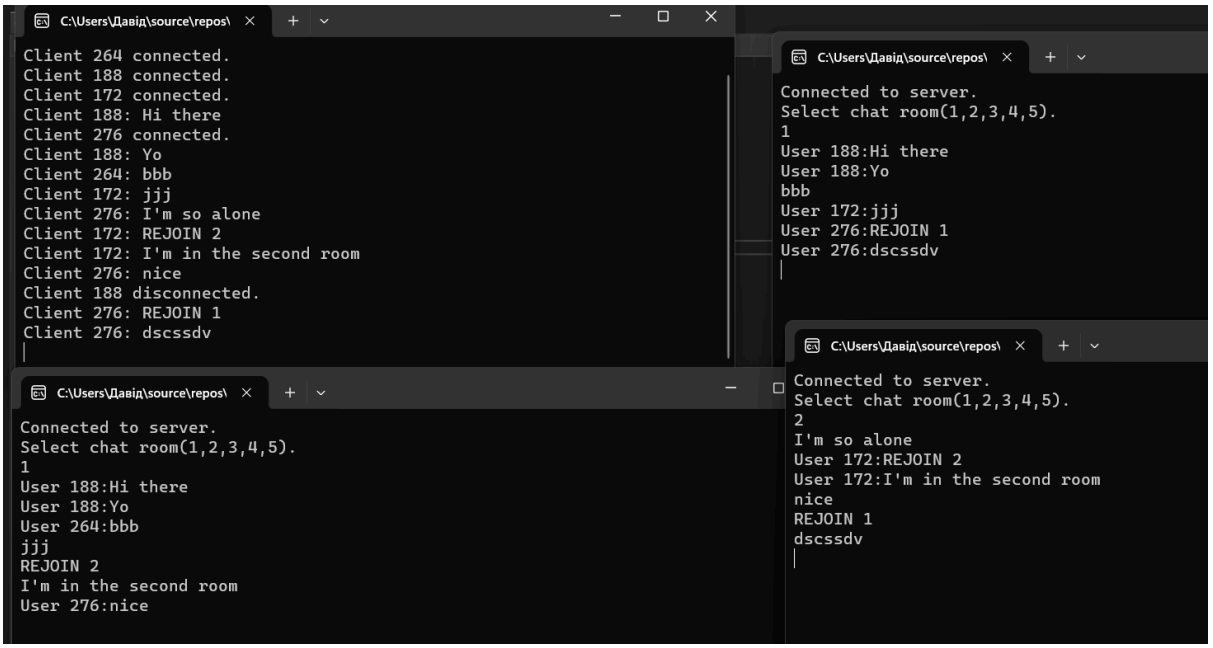

```
if (tokens[0] == "REJOIN") {
    for (int i = 0; i < roomset[roomNum].size(); i++)
    {
        if (roomset[roomNum][i] == clientSocket) {
            roomset[roomNum].erase(roomset[roomNum].begin() + i);
            break;
        }
    }
    roomNum = tokens[1];
    roomset[roomNum].push_back(clientSocket);
}
```

As you can see users can only communicate with users that are with one room

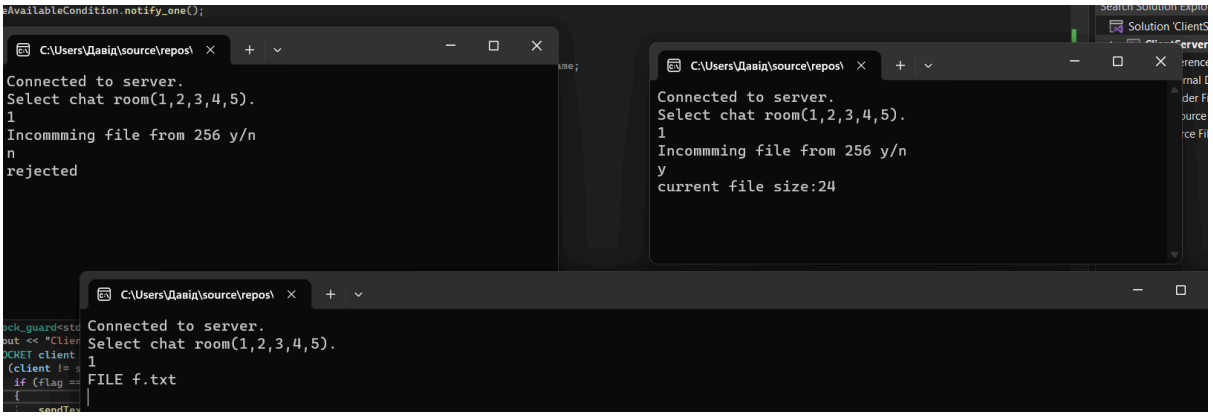


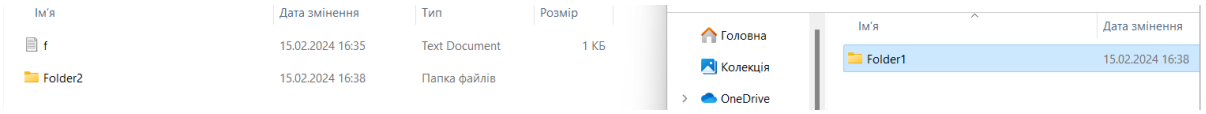

```
void broadcastMessage(const std::string& message, SOCKET senderSocket, std::string roomNum, std::string flag) {
    std::lock_guard<std::mutex> lock(consoleMutex);
    std::cout << "Client " << senderSocket << ": " << message << std::endl;
    for (SOCKET client : roomset[roomNum]) {
        if (client != senderSocket) {
            if (flag == "text")
            {
                sendText(std::to_string(senderSocket), client);
                sendText(message, client);
            }
            else {
                sendText(std::to_string(senderSocket), client);
                sendText("FILEINCOMING", client);
                sendText(message, client);
                PutToClient(client, message);
            }
        }
    }
}
```

One of the users decided to REJOIN to the second room to talk with the user 276, while user 188 and and 264 remained in the first room



As we can see here user 188 disconnected from the server by force but still no errors happening, users can freely reconnect and communicate with each other.






```
void receiveMessages(SOCKET clientSocket) {
    while (true) {
        std::string user = receiveText(clientSocket);
        std::string message = receiveText(clientSocket);
        if (message == "error") {
            std::cerr << "Server disconnected.\n";
            break;
        }
        if (message == "FILEINCOMING") {
            consoleMutex.lock();
            std::cout << "Incommming file from "<<user<<" y/n" << std::endl;
            consoleMutex.unlock();
            std::string name = receiveText(clientSocket);
            std::unique_lock<std::mutex> lock(dataMutex);
            dataCondition.wait(lock, [] { return flag != ""; });
            GetF(name, clientSocket, flag);
            if (flag != "y"){
                consoleMutex.lock();
                std::cout << "rejected" << std::endl;
                consoleMutex.unlock();
            }
            flag = "";
        }
        else {
            consoleMutex.lock();
            std::cout << "User " << user << ":" << message << std::endl;
            consoleMutex.unlock();
        }
    }
}
```

```
void GetFromClient(const int clientSocket)
{
    std::string name = receiveText(clientSocket);
    std::streamsize fileSize;
    if (recv(clientSocket, (char*)&fileSize, sizeof(std::streamsize), 0) == SOCKET_ERROR)
    {
        std::lock_guard<std::mutex> lock(consoleMutex);
        std::cout << WSAGetLastError() << std::endl;

    }
    std::ofstream outFile("C:\\Users\\Давід\\source\\repos\\Te\\ClientServer2\\ClientServer2\\" + name, std::ios::binary);
    std::streamsize totalReceived = 0;
    while (totalReceived < fileSize)
    {
        char buffer[BUFSIZE];
        std::streamsize bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        outFile.write(buffer, bytesReceived);
        totalReceived += bytesReceived;
        std::lock_guard<std::mutex> lock(consoleMutex);
        std::cout << "current file size:" << totalReceived << std::endl;
    }
    outFile.close();
}
```

In this screenshot we can see file sending with one client accepting the file ,while another rejects the file. The one who accepted the file has it in his folder.