CS 461

Lab Assignment 5

Name: Gandhi Dhruv Vipulkumar

Institute ID: 202151053

Date: 15-10-2024

**Q. Implement Distributed Chat Application**

**Server.py:**

import socket  
import threading  
import sqlite3  
  
# Database connection  
conn = sqlite3.connect('chat.db', check\_same\_thread=False)  
cursor = conn.cursor()  
  
# Creating tables if they don't exist  
cursor.execute(  
    '''CREATE TABLE IF NOT EXISTS users (username TEXT PRIMARY KEY, password TEXT)''')  
cursor.execute(  
    '''CREATE TABLE IF NOT EXISTS messages (sender TEXT, recipient TEXT, message TEXT)''')  
  
# Dictionary to hold online clients and groups  
clients = {}  
groups = {}  # Dictionary to store groups and their members  
  
# Function to broadcast messages to all members of a group  
  
def broadcast\_group(message, room, sender=None):  
    if room in groups:  
        for client\_socket in groups[room]:  
            if client\_socket != sender:  
                client\_socket.send(f"Group {room}: {message}".encode())  
  
# Function to handle each client  
  
def handle\_client(client\_socket, client\_address):  
    username = None  
  
    # User authentication (login/signup)  
    while True:  
        try:  
            choice = client\_socket.recv(1024).decode()  
            if choice == "signup":  
                username, password = client\_socket.recv(  
                    1024).decode().split(':')  
                try:  
                    cursor.execute(  
                        "INSERT INTO users (username, password) VALUES (?, ?)", (username, password))  
                    conn.commit()  
                    client\_socket.send(  
                        "Signup successful! You can now start chatting.".encode())  
                except sqlite3.IntegrityError:  
                    client\_socket.send(  
                        "Username already exists. Try a different one.".encode())  
            elif choice == "login":  
                username, password = client\_socket.recv(  
                    1024).decode().split(':')  
                cursor.execute(  
                    "SELECT password FROM users WHERE username=?", (username,))  
                stored\_password = cursor.fetchone()  
                if stored\_password and stored\_password[0] == password:  
                    client\_socket.send(  
                        "Login successful! Welcome to the chat.".encode())  
  
                    # Add the user to the clients dictionary  
                    clients[username] = client\_socket  
                    break  
                else:  
                    client\_socket.send("Invalid credentials.".encode())  
        except:  
            client\_socket.close()  
            return  
  
    # Handling messaging after login/signup  
    while True:  
        try:  
            message = client\_socket.recv(1024).decode()  
  
            if message.startswith("/private"):  
                \_, recipient, msg = message.split(' ', 2)  
                if recipient in clients:  
                    clients[recipient].send(  
                        f"Private from {username}: {msg}".encode())  
                    cursor.execute(  
                        "INSERT INTO messages (sender, recipient, message) VALUES (?, ?, ?)", (username, recipient, msg))  
                    conn.commit()  
                else:  
                    client\_socket.send("User not online.".encode())  
  
            elif message.startswith("/group"):  
                \_, room, msg = message.split(' ', 2)  
  
                if room not in groups:  
                    groups[room] = []  
                if client\_socket not in groups[room]:  
                    groups[room].append(client\_socket)  
  
                # Broadcast the message to all group members  
                broadcast\_group(f"{username}: {msg}",  
                                room, sender=client\_socket)  
  
            elif message == "/logout":  
                client\_socket.send("You have logged out.".encode())  
                client\_socket.close()  
  
                # Remove user from clients and groups when they logout  
                if username in clients:  
                    del clients[username]  
                for group in groups.values():  
                    if client\_socket in group:  
                        group.remove(client\_socket)  
                break  
        except:  
            # Handle disconnection  
            client\_socket.close()  
            if username in clients:  
                del clients[username]  
            for group in groups.values():  
                if client\_socket in group:  
                    group.remove(client\_socket)  
            break  
  
# Main server function to accept incoming connections  
  
def start\_server():  
    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
    server\_socket.bind(('127.0.0.1', 12345))  
    server\_socket.listen(5)  
    print("Server is listening...")  
  
    while True:  
        client\_socket, client\_address = server\_socket.accept()  
        print(f"New connection from {client\_address}")  
        thread = threading.Thread(  
            target=handle\_client, args=(client\_socket, client\_address))  
        thread.start()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    start\_server()

**Client.py**

import socket  
import threading  
  
# Function to receive messages  
  
def receive\_messages(client\_socket):  
    while True:  
        try:  
            message = client\_socket.recv(1024).decode()  
            print(message)  
        except Exception as e:  
            print(f"An error occurred while receiving message: {e}")  
            client\_socket.close()  
            break  
  
# Function to handle sending messages  
  
def send\_messages(client\_socket):  
    while True:  
        message = input()  
        if message.startswith("/private"):  
            recipient = input("Recipient: ")  
            msg = input("Message: ")  
            client\_socket.send(f"/private {recipient} {msg}".encode())  
        elif message.startswith("/group"):  
            room = input("Room name: ")  
            msg = input("Message: ")  
            client\_socket.send(f"/group {room} {msg}".encode())  
        elif message == "/logout":  
            client\_socket.send(message.encode())  
            break  
  
# Function to start the client  
  
def start\_client():  
    client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
    client\_socket.connect(('127.0.0.1', 12345))  
  
    print("Welcome to the chat app!")  
    auth\_choice = input(  
        "Do you want to login or signup? (login/signup): ").strip()  
    client\_socket.send(auth\_choice.encode())  
  
    if auth\_choice == "signup":  
        username = input("Choose a username: ")  
        password = input("Choose a password: ")  
        client\_socket.send(f"{username}:{password}".encode())  
    elif auth\_choice == "login":  
        username = input("Username: ")  
        password = input("Password: ")  
        client\_socket.send(f"{username}:{password}".encode())  
  
    # Receive confirmation message (Signup/Login success)  
    response = client\_socket.recv(1024).decode()  
    print(response)  
  
    if "successful" in response:  
        # Once signup/login is successful, allow sending and receiving messages  
        print("Welcome to the chat room. You can now send messages!")  
  
        # Start a thread for receiving messages  
        receive\_thread = threading.Thread(  
            target=receive\_messages, args=(client\_socket,))  
        receive\_thread.start()  
  
        # Handle sending messages  
        send\_messages(client\_socket)  
  
    # Close the socket connection after logout  
    client\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    start\_client()

**Code Explanation:**

** Server:**

* The server listens for client connections.
* When a client connects, it spawns a new thread to handle communication with that client.
* It broadcasts messages to all connected clients except the sender.
* If a client disconnects, it removes that client from the list.

** Client:**

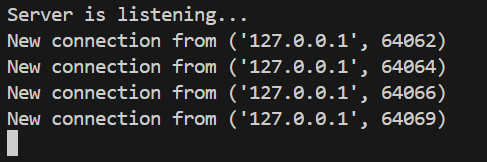
* The client connects to the server and spawns a thread to listen for incoming messages.
* The user can send messages to the server, which will be broadcasted to other clients.
* If the user types exit, the client disconnects from the server.

**Key Features:**

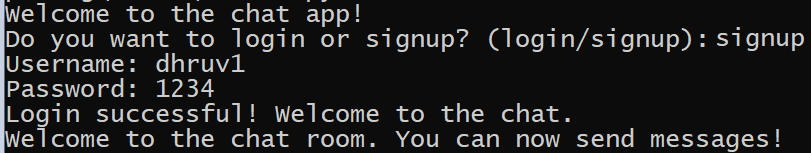
1. User Authentication (Signup/Login)
2. Private Chat (Direct Messaging)
3. Group Chatrooms
4. Multi-server Support (Distributed Setup)

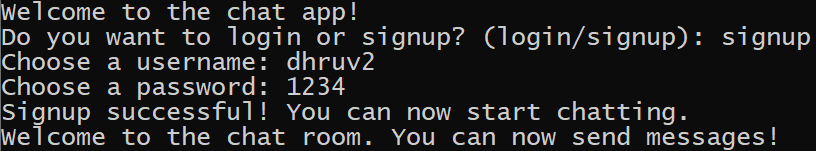
**Testing Phase:**

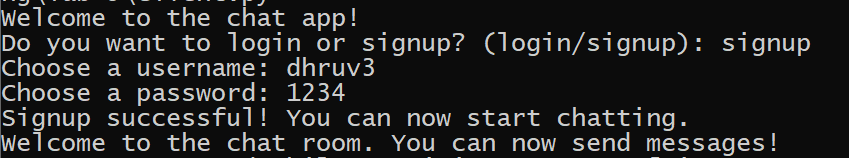
**1) Start the server:**

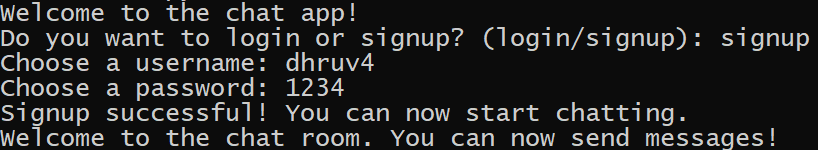
****

**2) Signup and Login 4 different users dhruv1-4:**

****

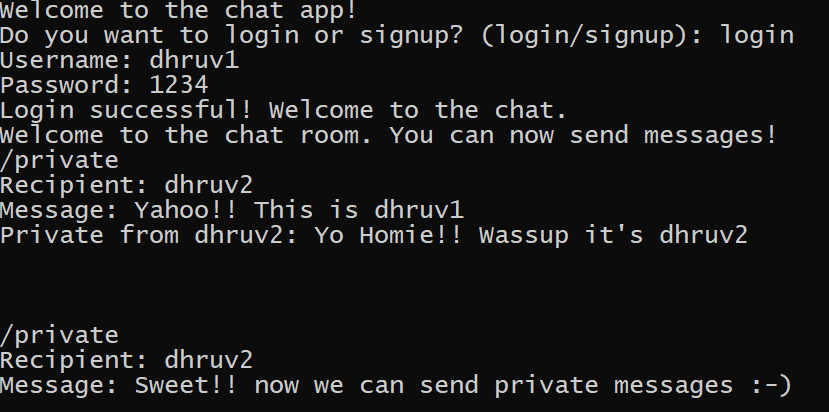
****

****

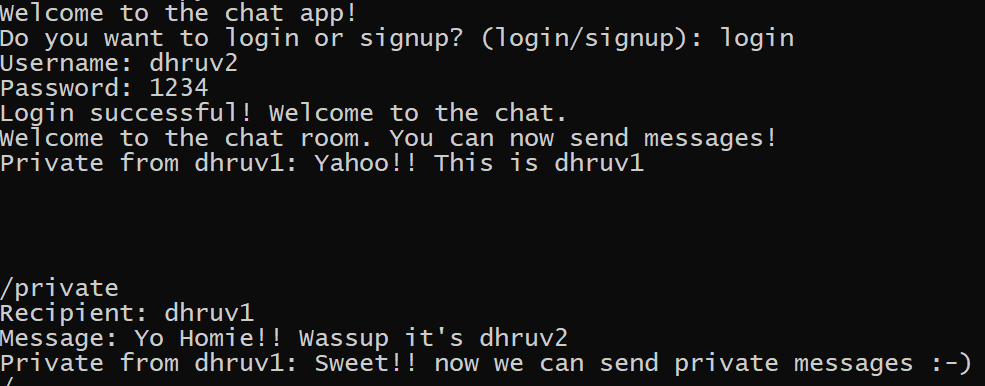
****

**3) dhruv1 send private message to dhruv2**

**From dhruv1:**

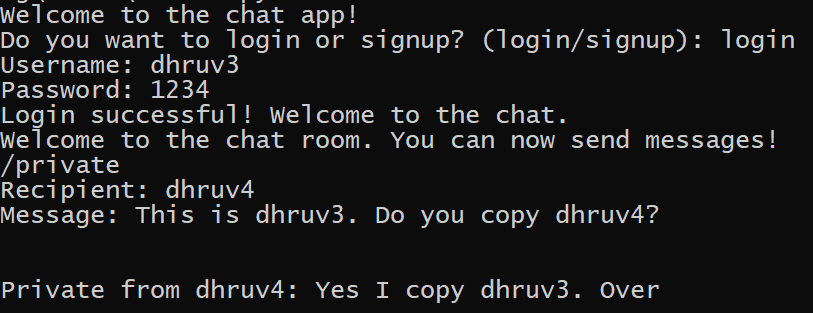
****

**From dhruv2:**

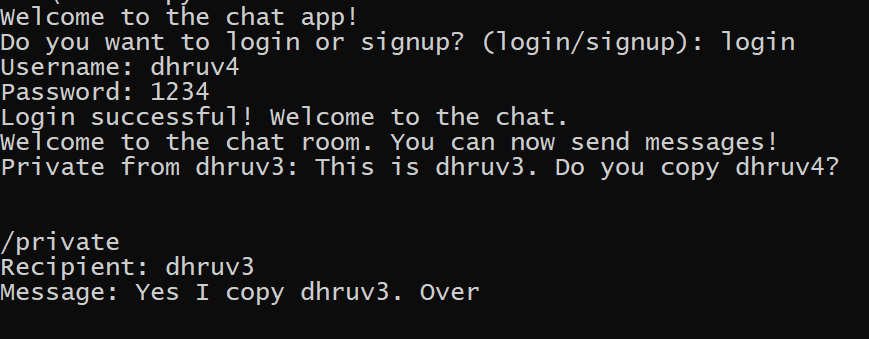
****

**4) sending message from dhruv3 to dhruv4:**

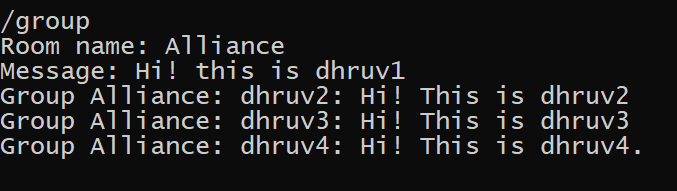
**(From dhruv3)**

****

**(From dhruv4)**

****

**5) Create a group named “Alliance” and send messages:**

****

**Conclusion:** Successfully implemented chat application in python.