Ramdeobaba University, Nagpur
Department of Computer Science and Engineering
Session: 2025-26

Subject: Computer Network

Name : Prachi Sharnagat
Section : A2 B2
Roll no : 20

Lab 9: TCP Socket Programming – Multi-Client Chat System

Github link :

https://github.com/Prachi-Sharnagat/Python-SocketChat

Objective :

The objective of this laboratory exercise was to implement a robust, connection-oriented chat application using TCP/IP sockets in Python. The program demonstrates the fundamental principles of the Client–Server model, including socket creation, binding, listening, and accepting connections.

Application Architecture

The application consists of two main Python programs that work in tandem:

1. `server.py` – **Multi-Client Handler**

The server is designed to handle multiple clients concurrently using **threads**.

- **Main Thread:**
  Continuously calls `server.accept()` to detect and accept new client connections.
  Upon connection, it spawns a **new thread** dedicated to that client.

- **Worker Threads:**
  Each thread executes the `handle_client()` function.
  This allows each client to chat independently without blocking others.
- **Broadcasting:**
  Messages received from one client are **broadcasted** to all connected clients using a shared `clients` set, enabling group chat functionality.

```
import threading

import socket



PORT = 5050

SERVER = "localhost"

ADDR = (SERVER, PORT)

FORMAT = "utf-8"

DISCONNECT_MESSAGE = "!DISCONNECT"



server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server.bind(ADDR)



clients = set()

clients_lock = threading.Lock()




def handle_client(conn, addr):

    print(f"[NEW CONNECTION] {addr} Connected")
```

```
    try:
        connected = True
        while connected:
            msg = conn.recv(1024).decode(FORMAT)
            if not msg:
                break

            if msg == DISCONNECT_MESSAGE:
                connected = False

            print(f"[{addr}] {msg}")
            with clients_lock:
                for c in clients:
                    c.sendall(f"[{addr}] {msg}".encode(FORMAT))

    finally:
        with clients_lock:
            clients.remove(conn)

        conn.close()



def start():
```

```
    print('[SERVER STARTED]!')

    server.listen()

    while True:

        conn, addr = server.accept()

        with clients_lock:

            clients.add(conn)

        thread = threading.Thread(target=handle_client, args=(conn,
addr))

        thread.start()




start()
```

2. `client.py` – **Asynchronous Communicator**

The client application also uses threading to allow sending and receiving messages simultaneously.

- **Main Thread (Sender):**
  Captures user input using `input()` and sends messages using `client.send()`.
- **Worker Thread (Receiver):**
  Runs `receive_messages()` in the background to continuously listen for broadcast messages from the server using `client.recv()`.

This architecture ensures **non-blocking communication** and real-time chat functionality.

```
import socket
```

```
import time



PORT = 5050

SERVER = "localhost"

ADDR = (SERVER, PORT)

FORMAT = "utf-8"

DISCONNECT_MESSAGE = "!DISCONNECT"




def connect():

    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    client.connect(ADDR)

    return client




def send(client, msg):

    message = msg.encode(FORMAT)

    client.send(message)




def start():

    answer = input('Would you like to connect (yes/no)? ')

    if answer.lower() != 'yes':
```

```
        return


    connection = connect()
    while True:
        msg = input("Message (q for quit): ")


        if msg == 'q':
            break


        send(connection, msg)


    send(connection, DISCONNECT_MESSAGE)
    time.sleep(1)
    print('Disconnected')



start()
```

Outcome:

A fully functional multi-client chat system that demonstrates concurrent socket communication using Python's socket and threading modules.