By:
Adil Zainul Syed

Socket Network Programming – Client-Server Chat Application

Objective:
To implement client-server communication using TCP sockets where multiple clients can send and receive messages from a central server.

Tools Required:
• Language: Python 3
• Libraries: socket, threading

Theory:
Socket programming allows communication between two processes — typically on different machines — over a network. A server waits for client requests, while a client initiates a connection. TCP sockets ensure reliable, ordered, and error-checked communication.

Algorithm:

Server:
1. Create a socket using socket.socket().
2. Bind it to an IP address and port.
3. Listen for incoming connections using listen().
4. Accept client connections in a loop.
5. For each client, start a new thread to handle messages concurrently.
6. Broadcast messages to all connected clients.

Client:
1. Create a socket and connect to the server's IP and port.
2. Send messages to the server.
3. Continuously listen for messages from the server.

Program Code:

a) Server (server.py):

```
import socket

import threading
```

```
clients = []
client_ids = {}   # Map socket to client ID
client_count = 0  # Incremental ID counter

def handle_client(client_socket):
    client_id = client_ids[client_socket]
    while True:
        try:
            message = client_socket.recv(1024).decode()
            if not message:
                break
            formatted_message = f"{client_id}: {message}"
            print(formatted_message)
            broadcast(formatted_message, client_socket)
        except:
            print(f"{client_id} disconnected.")
            clients.remove(client_socket)
            del client_ids[client_socket]
            client_socket.close()
            break

def broadcast(message, sender_socket):
    for client in clients:
        if client != sender_socket:
            client.send(message.encode())

def main():
    global client_count
```

```
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('localhost', 12345))
    server.listen(5)
    print("Server listening on port 12345...")

    while True:
        client_socket, addr = server.accept()
        client_count += 1
        client_id = f"Client-{client_count} (socket-{client_socket.fileno()})"
        client_ids[client_socket] = client_id
        clients.append(client_socket)
        print(f"Connected to {addr} as {client_id}")
        client_socket.send(f"You are connected as {client_id}".encode())

        threading.Thread(target=handle_client, args=(client_socket,)).start()

if __name__ == "__main__":
    main()
```

b) Client (client.py):

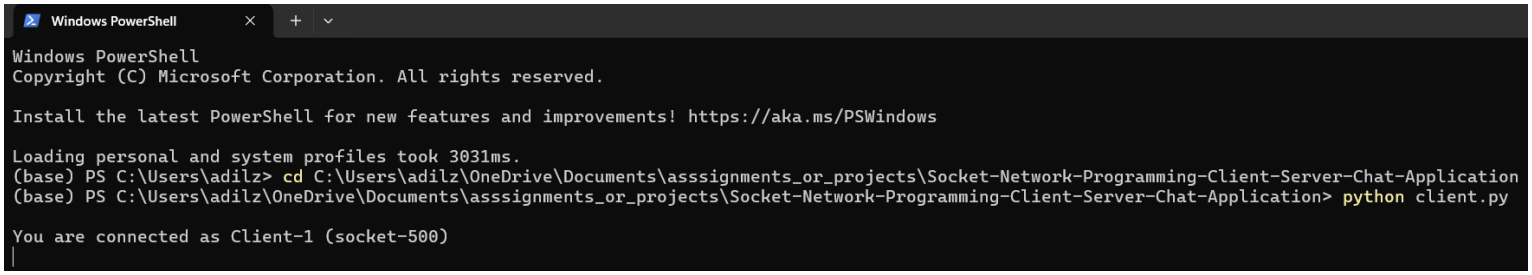
```
import socket
import threading

def receive_messages(client_socket):
    while True:
        try:
            message = client_socket.recv(1024).decode()
            if not message:
                break
            print("\n" + message)
```

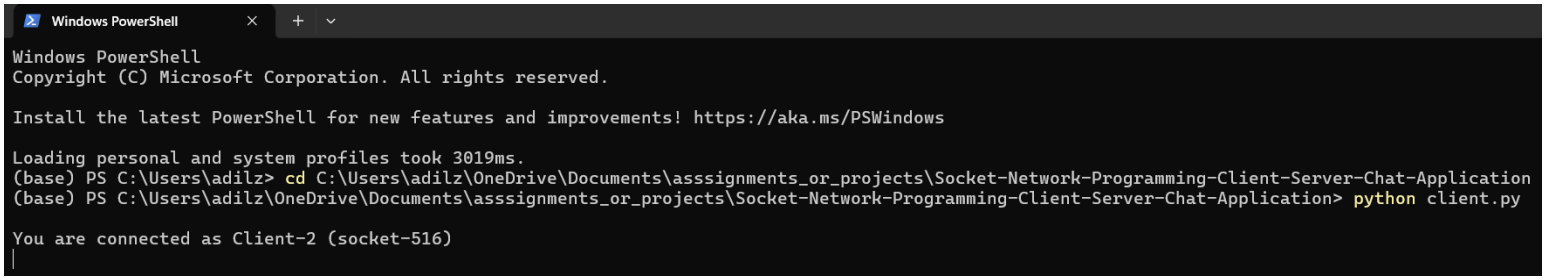
```
        except:
            print("Disconnected from server.")
            client_socket.close()
            break

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('localhost', 12345))

    threading.Thread(target=receive_messages, args=(client,)).start()

    while True:
        message = input("")
        client.send(message.encode())

if __name__ == "__main__":
    main()
```

Output:

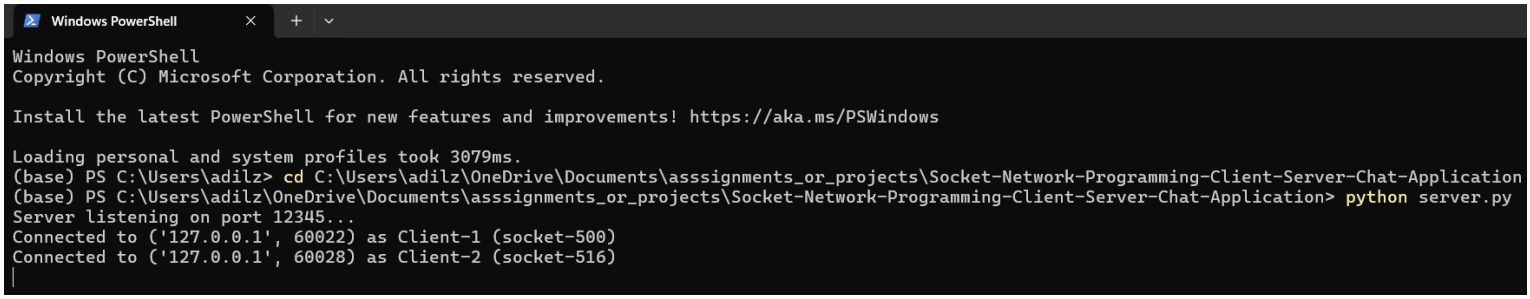
a) Setting up the Server side –

b) Setting up the Client side (2 Clients, same for both) –

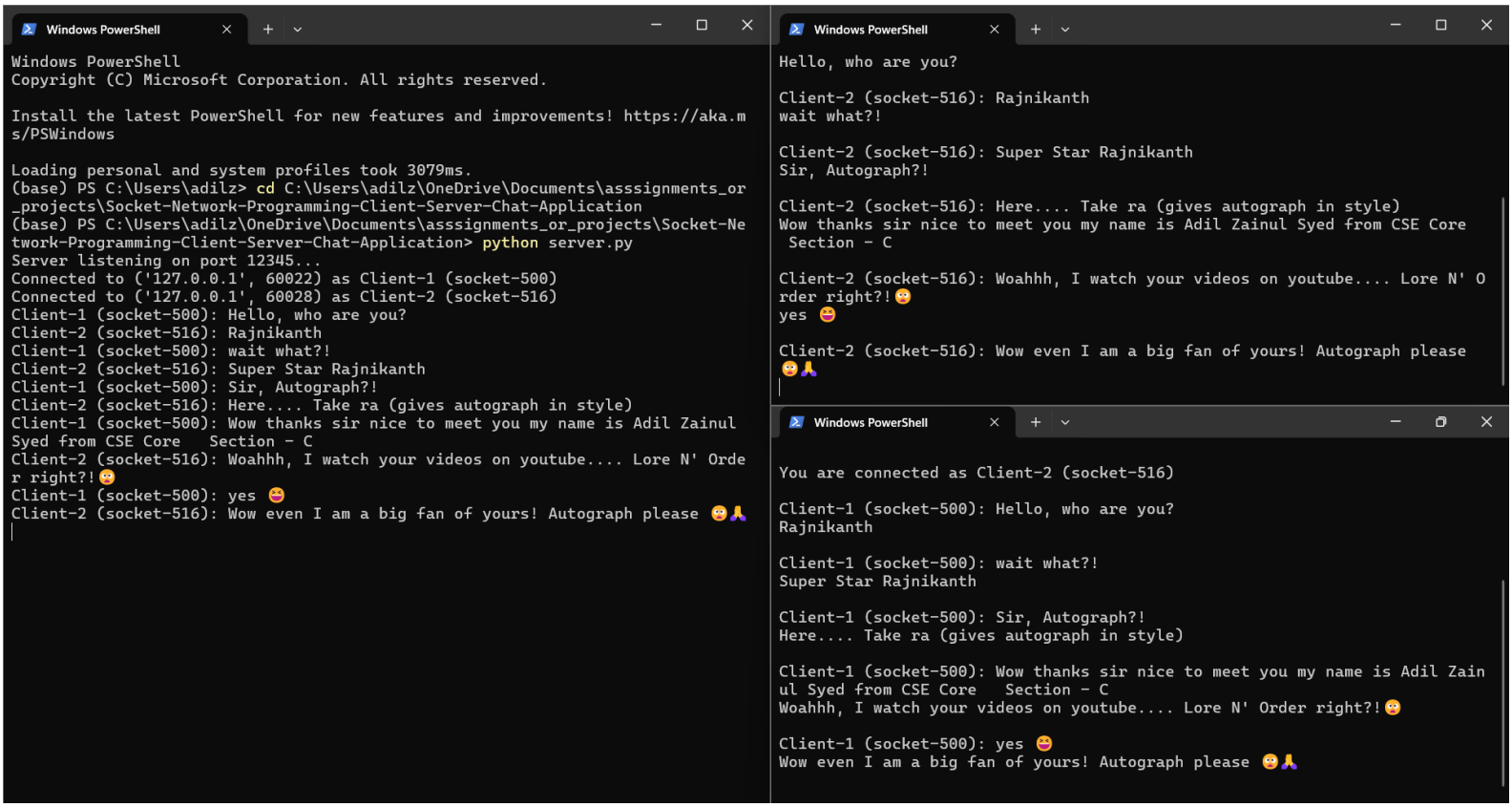




c) On the Server side after establishment of network connection between 2 clients -



d) Chatting between the clients after establishment of connection on the network –



Conclusion:

Successfully implemented client-server communication using sockets where multiple clients can interact with each other through a central server in real time.