Practical - 7

Aim: To implement Socket Programing

Scenario:
An organization named Albert Enterprise has established two departments for better performance of the company, as each department will be having some specific set of tasks to perform. So, this will reduce the time and increase the efficiency of the work. As both the departments are dependent on each other, they need to communicate more frequently. To solve the problem, the IT department has suggested the option to create a chat application using socket programming which will work only in the office premises. So, help the IT professionals to create the chat application.

Make sure that the application has the below mentioned features:

1) Department 1 will be set as the SERVER while department 2 will be set as a CLIENT device.

2) The message received by CLIENT or SERVER must be displayed with time stamp.

3) If any of the device irrespective of CLIENT or SERVER has sent the message that the "**quit**", then connection should be closed on both the ends.

4) There is no restriction on the protocol selection, you can use UDP or TCP. Justify the reason for selection of the specific protocol.

Expected Output:

```
----------
  SERVER
----------
SERVER is listening..
Connection accepted from ('192.168.1.6', 53792)
CLIENT [2024-10-06 21:13:08]:  Hi I am Client 1
ENTER TEXT: I am Server
CLIENT [2024-10-06 21:13:32]:  My task is done
ENTER TEXT: quit
CLIENT [2024-10-06 21:13:36]:
```

```
----------
  CLIENT
----------
Hello there! msg from SERVER
ENTER TEXT: Hi I am Client 1
SERVER [2024-10-06 21:13:17]:  I am Server
ENTER TEXT: My task is done
SERVER [2024-10-06 21:13:36]:  quit
```

Server Code:

```
import socket
import threading
from datetime import datetime

# Function to handle client connection
def handle_client(client_socket):
    while True:
        # Receive message from the client
        message = client_socket.recv(1024).decode()

        # Get current timestamp
        timestamp = datetime.now().strftime('%Y-%m-%d
%H:%M:%S')

        if message.lower() == 'quit':
            print(f"[{timestamp}] Client disconnected")
            client_socket.close()
            break

        print(f"[{timestamp}] Client: {message}")
        # Send a reply to the client
        server_message = input("Enter message to client:
")
        client_socket.send(server_message.encode())

        if server_message.lower() == 'quit':
            print("Closing connection.")
            client_socket.close()
            break

# Start the server
def start_server():
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```

```
    server_socket.bind(('0.0.0.0', 12345))  # Bind to all
network interfaces on port 12345
    server_socket.listen(1)
    print("Server is listening...")

    while True:
        client_socket, addr = server_socket.accept()
        print(f"Connection established with {addr}")

        # Handle the client in a new thread
        client_thread =
threading.Thread(target=handle_client,
args=(client_socket,))
        client_thread.start()

# Run the server
if __name__ == "__main__":
    start_server()
```

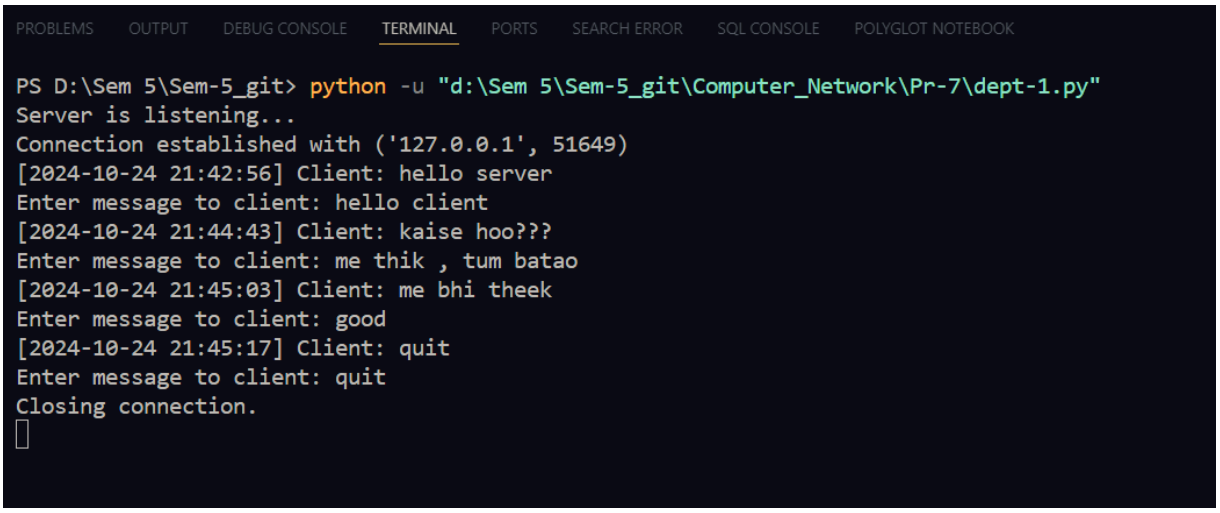
Client Code:

```
import socket

from datetime import datetime

def start_client():

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

    client_socket.connect(('127.0.0.1', 12345))  # Connect to the server
(replace '127.0.0.1' with server IP)

    while True:

        # Send message to server

        message = input("Enter message to server: ")

        client_socket.send(message.encode())

        # Get current timestamp

        timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        if message.lower() == 'quit':
```

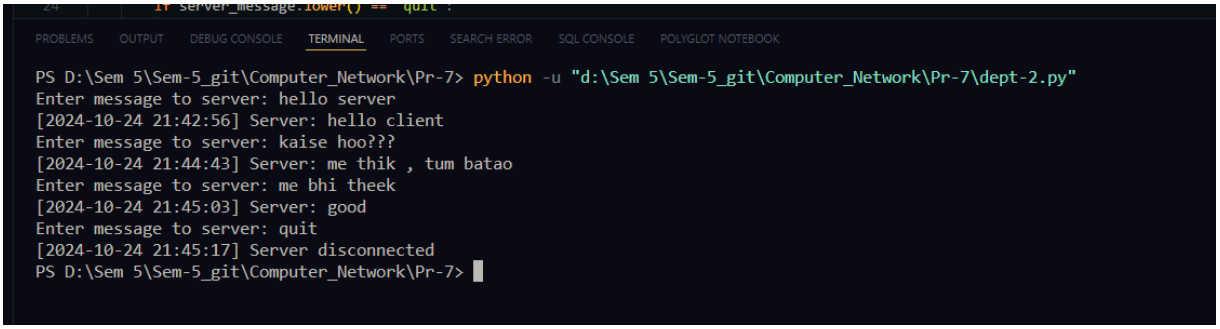
```
            print("Closing connection.")
            client_socket.close()
            break

        # Receive response from server
        server_message = client_socket.recv(1024).decode()

        if server_message.lower() == 'quit':
            print(f"[{timestamp}] Server disconnected")
            client_socket.close()
            break
        print(f"[{timestamp}] Server: {server_message}")

# Run the client
if __name__ == "__main__":
    start_client()
```

Output:

Server:



Client:



Conclusion:

When we configure the server and client side code correctly they can communicate with each other through CLI