Exp. No: 12
10.10.25

a) Implement echo client server using TCP/UDP sockets.

Aim:

To implement echo client server using TCP/UDP sockets.

Algorithm:

```
import socket
import threading
def handle-client (client-socket, client-address):
    Print(f"[+] New connection from {client address}")
    while True:
        try:
            msg = client_socket.recv (1024).decode()
            if not msg:
                break
            Print (f"[client {clientaddress }] {msg}")
            client-socket. sendall(f"server received. {msg}".encode())
        except connection Reset Error:
            break
    Print (f"[-] connection closed {client-address}")
    Client_socket. close()
def start_server(host = "127.0.0.1", port =5000):
    server_socket = socket.socket (socket .AF.INET, socket. SOCK.STREAM)
    server_socket. bind((host , port))
```

```
server-socket.listen(5)
print(f"[SERVER] Listening on {host}:{port}..

While True:
    client-socket, client-address = server-
                                    socket.accept
    client-thread = threading-Thread(
         target = handle-Client, args=(client
                                  socket, client_ad
    )
    client_thred. Start()
```

Client code

```
def start-client (server-host="127.0.0.1", se

    client-socket = socket. socket (saket
                         socket.sock-STR

    client-socket.connect((server-host
    print(f"[CLIENT] connected to server
                               {server_port}")

    try:
      while True:
        msg = input("Enter message
                          to exit

        if msg.lower() == "quit"
              break
        client-socket. sender (msg.
        response = client=socket.recv(
        print(f"[SERVER REsponse]
    finally:
        client-socket. close()
        print("[CLIENT] Disconnecte
if -name- =="main-":
    import sys
```

```
if len(sys.argv) > 1 and sys.argv[1] == "server":
        start_server()
    else:
        start_client()
```

Sample Input and Output

Step 1: Run the Server

```
$ Python chat-program.py server
```

Server Output:

```
[SERVER] Listening on 127.0.0.1:5000...
[+] New connection from ('127.0.0.1', 60628)
[client ('127.0.0.1', 60628)] Hello Server!
[client ('127.0.0.1', 60628)] How are you?
[-] connection closed (127.0.0.1', 60628)
```

Step 2: Run the Client

```
$ Python chat-program.py
```

Client Interaction:

```
[CLIENT] connected to server 127.0.0.1:5000
Enter message (or 'quit' to exit): Hello server
[SERVER RESPONSE] Server received: Hello server!
Enter message (or 'quit' to exit): How are you?
[SERVER RESPONSE] server received How are you?
Enter message (or 'quit' to exit): quit
[CLIENT] Disconnected.
```

10/10

[illegible] 24/4/25

Result:

The Echo Client - Server and chat program were successfully implemented using TCP sockets.

End to End Communication

Aim:
To implement a UDP-based Echo (ping) server program using socket programming that measures Round Trip time (RTT) for each packet and demonstrates end to end comm at the Transport layer.

Program Code:

```
import socket
import time
import sys
def udp-ping-server(host="127.0.0.1", port=1
    server_socket = socket. socket(socket. AF_
                          socket. sock_DGRAM)
    server socket. bind ((host, port))
    print(f"[SERVER] Listening on {host}:
              {port}")
    While True:
        msg. client_address = server_socket
                                   recv from (1024)
        Print (f"[Server] Received '{msg.d
                     from {client_address}")
        server_socket send to(msg, client_a
```

client

```
def udp-ping-client(server_host="127.0.
                     server_port=12000, coun
    client_socket = socket. socket(socket. A
                          Socket. soc
```

```
client_socket.settimeOut(1)
for i in range (1, count+1):
    msg = f"ping {i} {time.time()}"
    start = time.time()
    client_socket.Sendto(msg.encode(),
                         (server_host, server_port))
  try:
    data _ = client_socket.recvfrom(1024)
    end = time.time()
    rtt = (end_start) * 1000
    print(f"Reply from {server_host}:{server_port}
                {data.decode()} RTT={rtt:.2f} ms")
  except socket.timeout:
        print(f"Request {i} timed out")
 Client_socket.close()
      if len(sys.argv) > 1 and sys.argv[1]=="server":
            udp_ping_server()
      else:
            udp_ping_client()
```

Sample Input and Output:

```
[SERVER] listening on 127.0.0.1:12000
[SERVER] Received 'ping 1 1728575342.123' from
                         ('127.0.0.1', 60642)
[SERVER] Received 'ping 2' 1728575343.125' from
                         ('127.0.0.1', 60642)
[SERVER] Received 'ping 3' 1728575344.127' from
                         ('127.0.0.1', 60642)
```

Step 2: Run the client

```
$ Python Udp-ping.py
```

{ 14/x/25 $\frac{10}{10}$

Result:

UDP echo (ping) Client-Server program was successfully implemented.