CONTENT BEYOND SYLLABUS

1.Lab Manual: Developing a TCP Client-Server Application using Python Socket Programming

Lab Setup:

1. **Setting up the Environment:**
   - Ensure that you have a Linux environment with Python installed.
   - Open a terminal.
2. **Creating the Project Directory:**
   - Create a new directory for your project.

bash

```
2. mkdir tcp_message_lab
   cd tcp_message_lab
```

Part 1: Server Side

Step 1: Writing the Server Code

1. Create a file named `server.py` in the project directory.

python

```
1. # server.py
   import socket

   PORT = 8080
   MAX_BUFFER_SIZE = 1024

   def main():
       server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       server_socket.bind(('0.0.0.0', PORT))
       server_socket.listen(3)

       print("Server listening on port", PORT)

       client_socket, client_address = server_socket.accept()
       print("Accepted connection from", client_address)

       data = client_socket.recv(MAX_BUFFER_SIZE).decode('utf-8')
       print("Received message from client:", data)

       client_socket.close()
       server_socket.close()

   if___name__== "__main__":
       main()
```

Step 2: Running the Server Code

1. Run the server.

   bash

1. python server.py

Part 2: Client Side

Step 1: Writing the Client Code

1. Create a file named `client.py` in the project directory.

   python

1.
```
# client.py
import socket

PORT = 8080
MAX_BUFFER_SIZE = 1024

def main():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('127.0.0.1', PORT)

    try:
        client_socket.connect(server_address)
        print("Connected to server on port", PORT)

        message = input("Enter a message to send to the server: ")
        client_socket.sendall(message.encode('utf-8'))

    finally:
        client_socket.close()

if___name__== "__main__":
    main()
```

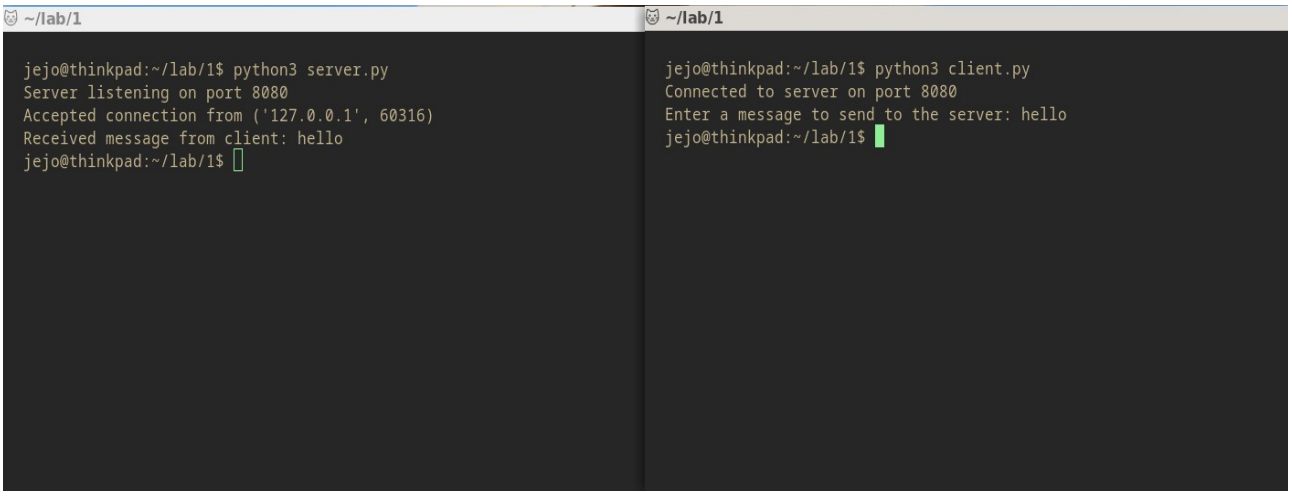
Step 2: Running the Client Code

1. Run the client (in a separate terminal).

   bash

```
python client.py
```

OUTPUT:



```
jejo@thinkpad:~/lab/1$ python3 server.py
Server listening on port 8080
Accepted connection from ('127.0.0.1', 60316)
Received message from client: hello
jejo@thinkpad:~/lab/1$
```

```
jejo@thinkpad:~/lab/1$ python3 client.py
Connected to server on port 8080
Enter a message to send to the server: hello
jejo@thinkpad:~/lab/1$
```

2. Lab Manual: Developing a UDP Client-Server Application using Python Socket Programming

Lab Setup:

1. **Setting up the Environment:**
   - Ensure that you have a UNIX-like operating system.
   - Open a terminal.
2. **Creating the Project Directory:**

   bash

2.
   ```
   mkdir udp_message_lab
   cd udp_message_lab
   ```

Part 1: Server Side

Step 1: Writing the Server Code

1. Create a file named `udp_server.py` in the project directory.

   python

1.
   ```
   # udp_server.py
   import socket

   PORT = 8080
   MAX_BUFFER_SIZE = 1024

   def main():
       sockfd = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
       server_addr = ('0.0.0.0', PORT)
       sockfd.bind(server_addr)

       print(f"UDP Server is listening on port {PORT}...")

       while True:
           data, client_addr = sockfd.recvfrom(MAX_BUFFER_SIZE)
           message = data.decode('utf-8')
           print(f"Message from client: {message}")

       sockfd.close()

   if___name__== "__main__":
       main()
   ```

Step 2: Running the Server Code

1. Run the server.

   bash

1.
   ```
   python udp_server.py
   ```

Part 2: Client Side

Step 1: Writing the Client Code

1. Create a file named `udp_client.py` in the project directory.

   python

```
1. # udp_client.py
   import socket

   PORT = 8080
   SERVER_IP = '127.0.0.1'
   MAX_BUFFER_SIZE = 1024

   def main():
       sockfd = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
       server_addr = (SERVER_IP, PORT)

       while True:
           message = input("Enter a message to send to the server: ")
           sockfd.sendto(message.encode('utf-8'), server_addr)

       sockfd.close()

   if___name__== "__main__":
       main()
```

Step 2: Running the Client Code

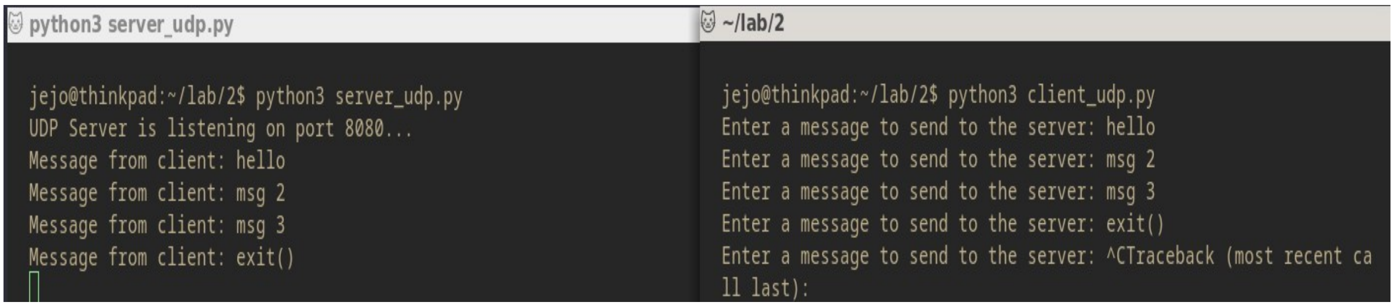
1. Run the client (in a separate terminal).

   bash

```
1. python udp_client.py
```

In both the server and client code, the `socket` module is used for creating and managing sockets. The logic is similar to the C code provided, demonstrating the use of UDP for communication between the client and the server. The `recvfrom` method is used to receive data from the client, and `sendto` is used to send data to the server.

OUTPUT:



```
jejo@thinkpad:~/lab/2$ python3 server_udp.py
UDP Server is listening on port 8080...
Message from client: hello
Message from client: msg 2
Message from client: msg 3
Message from client: exit()
```

```
jejo@thinkpad:~/lab/2$ python3 client_udp.py
Enter a message to send to the server: hello
Enter a message to send to the server: msg 2
Enter a message to send to the server: msg 3
Enter a message to send to the server: exit()
Enter a message to send to the server: ^CTraceback (most recent ca
ll last):
```