Distributed Systems
Lab Exercise 1

Roll. No: CH.SC.U4CSE23224
Name: Manoj Mehra V

1. Develop a client-server communication program that allows a user-entered message to be passed from the client to the server and vice versa. Client-server communication should occur between different systems with different IP addresses.

Objective of the Exercise:

To understand socket programming by developing a client–server communication model where a message entered by the client is sent to the server, processed, and a response is returned back. The communication operates between two different systems using their IP addresses.

Program Code:

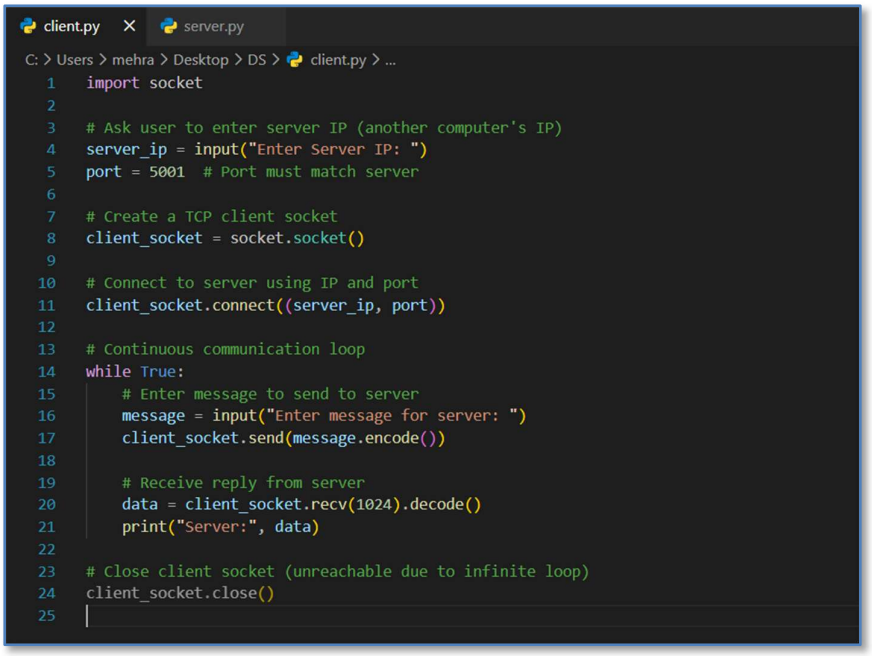
i) server.py:

server.py client.py

C: > Users > mehra > Desktop > DS > server.py > ...

```python
import socket

# Get the IP address of the current machine
host = socket.gethostbyname(socket.gethostname())
port = 5001  # Port used for communication

# Create a TCP socket
server_socket = socket.socket()

# Bind the socket to the host IP and port
server_socket.bind((host, port))

# Enable server to listen for incoming connections
server_socket.listen(1)

print("Server started...")
print("Server IP:", host)

# Accept a client connection
conn, address = server_socket.accept()
print("Connection from:", address)

# Continuous communication loop
while True:
    # Receive message from client
    data = conn.recv(1024).decode()
    if not data:
        break
    print("Client:", data)

    # Server enters reply
    reply = input("Enter reply to client: ")
    conn.send(reply.encode())

# Close connection
conn.close()
```

ii) client.py:



Explanation of the Code:

Server:

- The server creates a TCP socket and binds it to its own IP address and a fixed port.
- It listens for incoming connections and waits for a client to connect.
- Once connected, it receives messages from the client using recv().
- The server reads a reply from the user and sends it back using send().
- This message exchange continues in a loop until the client closes the connection.

Client:

- The client creates a TCP socket and connects to the server using the server's IP and port.It listens for incoming connections and waits for a client to connect.
- Once connected, it receives messages from the client using recv().
- The server reads a reply from the user and sends it back using send().
- This message exchange continues in a loop until the client closes the connection.

Inputs and Outputs Examples:

Server:

```
Server started...
Server IP: 11.12.2.179
Connection from: ('11.12.2.179', 55360)
```

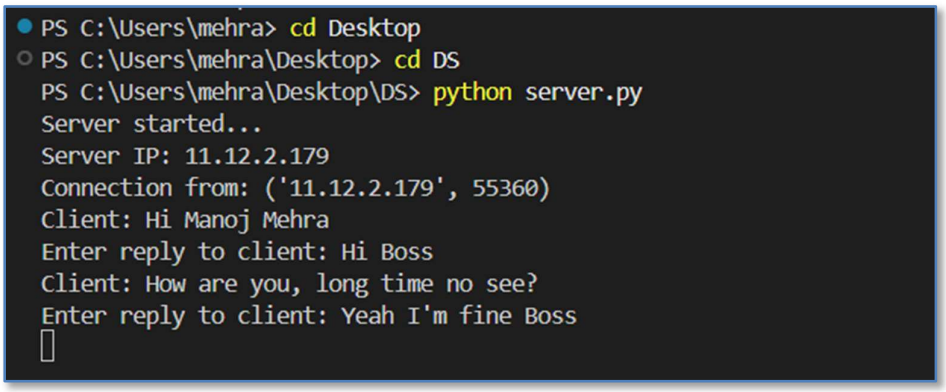
```
Client: Hi Manoj Mehra
Enter reply to client: Hi Boss
Client: How are you, long time no see?
Enter reply to client: Yeah I'm fine Boss
```

Client:

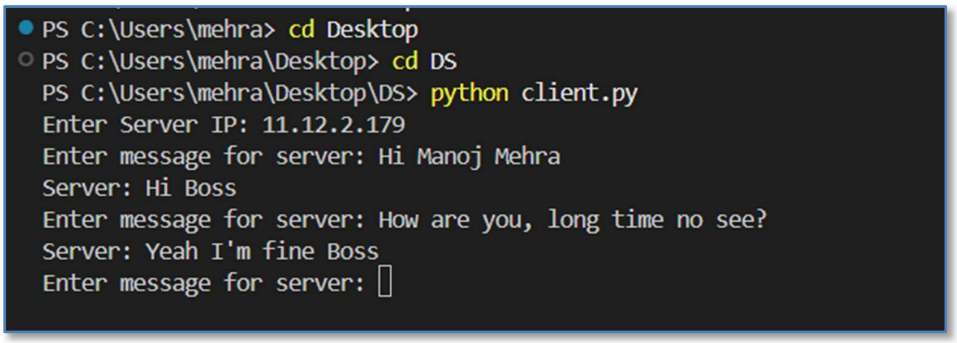
```
Enter Server IP: 11.12.2.179
Enter message for server: Hi Manoj Mehra
Server: Hi Boss
Enter message for server: How are you, long time no see?
Server: Yeah I'm fine Boss
```

Screenshots of Output:

Server:



Client:



Conclusion:

Through this exercise, we successfully implemented basic client–server communication using Python sockets. This helped understand network data transmission, socket creation, connection setup, and message handling between two machines.

2.Write a program that lists the available files from a local folder to the client using peer-to-peer communication. The list should not be a dummy one; it should fetch real files from a specific folder and display them to the client.

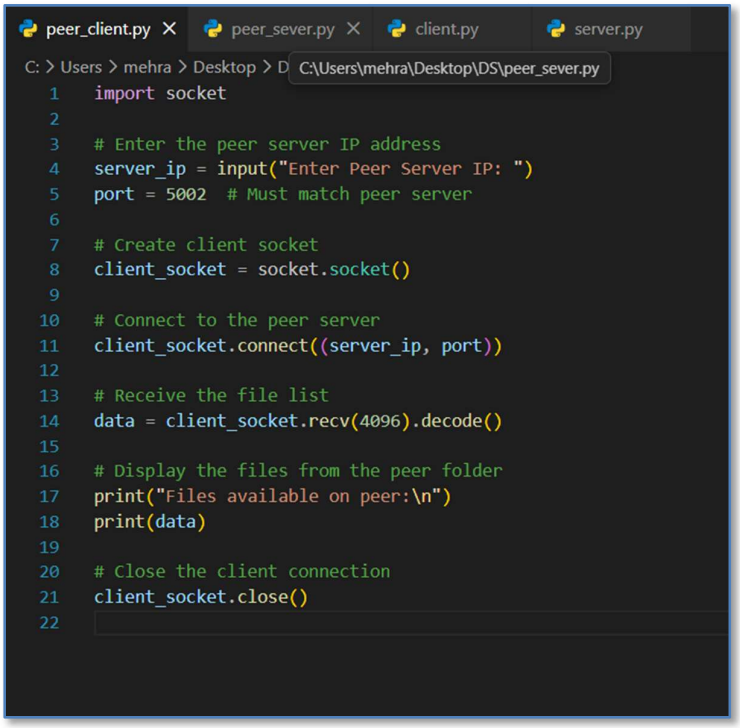
Objective of the Exercise:

To build a peer-to-peer (P2P) system where a client connects to another machine (peer) and retrieves the actual list of files present in a specific folder—NOT hardcoded, but fetched dynamically.

Program Code:

i)peer_server.py:

i)peer_client.py:



Explanation of the Code:

Peer_server:

- The peer server creates a TCP socket and waits for a peer client to connect.
- It uses os.listdir() to read the actual files from the specified folder.
- These filenames are combined into a single string.
- The server sends this complete file list to the connected client.

Peer_client:

- The peer client connects to the peer server using its IP and port.
- After connection, it receives the file list sent by the server.
- It prints these filenames, showing the actual contents of the server's folder.

Inputs and Outputs Examples:

Server:

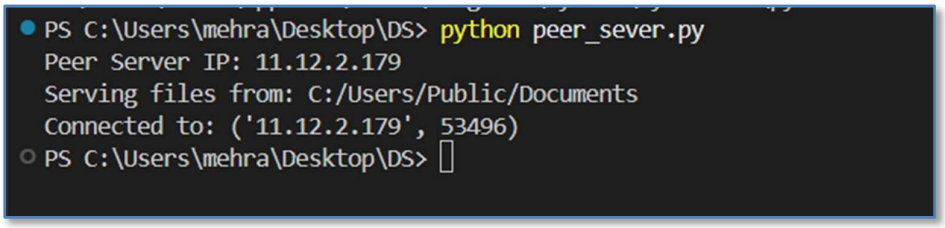
```
Peer Server IP: 11.12.2.179
Serving files from: C:/Users/Public/Documents
Connected to: ('11.12.2.179', 53496
```

Client:

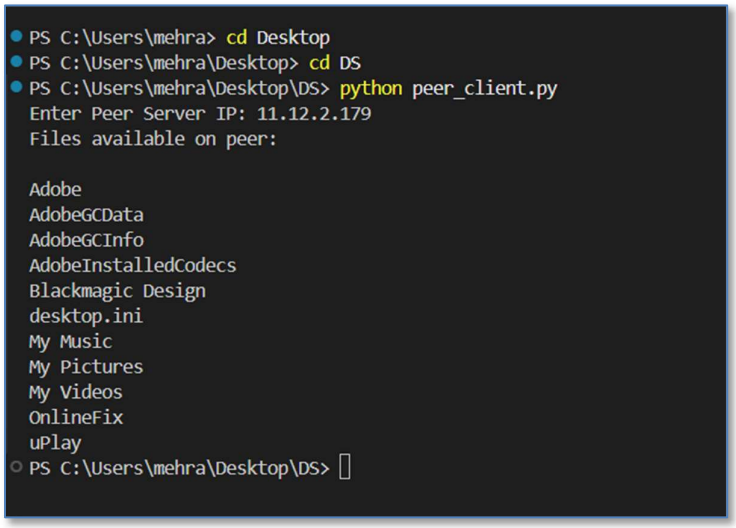
```
Enter Peer Server IP: 11.12.2.179

Files available on peer:

Adobe
AdobeGCData
AdobeGCInfo
AdobeInstalledCodecs
Blackmagic Design
desktop.ini
My Music
My Pictures
My Videos
OnlineFix
uPlay
```

Screenshots of Output:

Peer_Server:



Peer_Client:



Conclusion:

This program demonstrates peer-to-peer communication and real-time file discovery in a shared folder. It reinforces skills in directory handling, socket communication, and network-based data exchange.

--x--