

Distributed Systems

Lab Exercise 1

Roll. No: CH.SC.U4CSE23224

Name: Manoj Mehra V

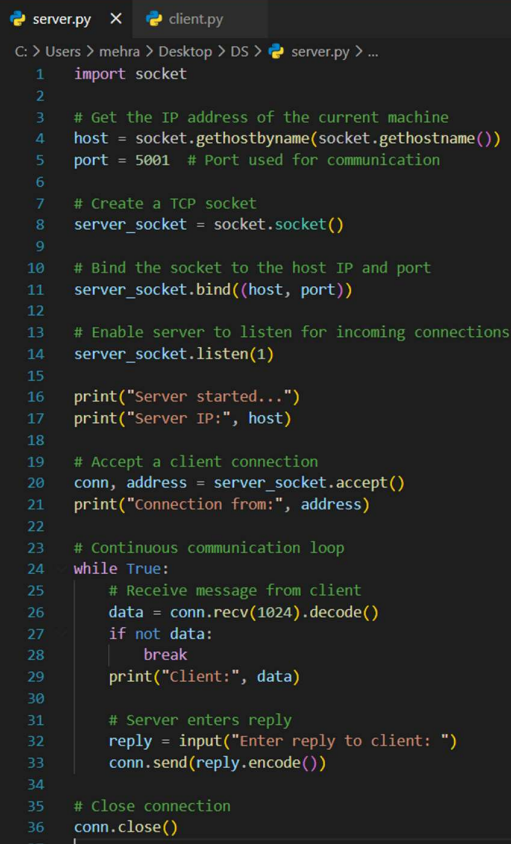
1. Develop a client-server communication program that allows a user-entered message to be passed from the client to the server and vice versa. Client-server communication should occur between different systems with different IP addresses.

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.

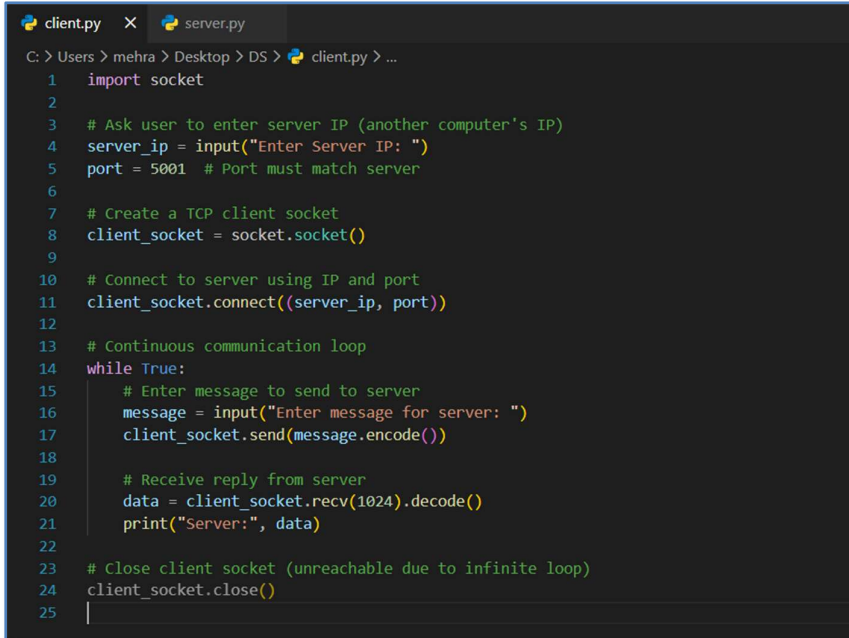
Program Code:

i) server.py:



```
server.py X client.py
C: > Users > mehra > Desktop > DS > server.py > ...
1  import socket
2
3  # Get the IP address of the current machine
4  host = socket.gethostname(socket.gethostname())
5  port = 5001 # Port used for communication
6
7  # Create a TCP socket
8  server_socket = socket.socket()
9
10 # Bind the socket to the host IP and port
11 server_socket.bind((host, port))
12
13 # Enable server to listen for incoming connections
14 server_socket.listen(1)
15
16 print("Server started...")
17 print("Server IP:", host)
18
19 # Accept a client connection
20 conn, address = server_socket.accept()
21 print("Connection from:", address)
22
23 # Continuous communication loop
24 while True:
25     # Receive message from client
26     data = conn.recv(1024).decode()
27     if not data:
28         break
29     print("Client:", data)
30
31     # Server enters reply
32     reply = input("Enter reply to client: ")
33     conn.send(reply.encode())
34
35 # Close connection
36 conn.close()
37
```

ii) client.py:



```
client.py x server.py
C: > Users > mehra > Desktop > DS > client.py > ...
1 import socket
2
3 # Ask user to enter server IP (another computer's IP)
4 server_ip = input("Enter Server IP: ")
5 port = 5001 # Port must match server
6
7 # Create a TCP client socket
8 client_socket = socket.socket()
9
10 # Connect to server using IP and port
11 client_socket.connect((server_ip, port))
12
13 # Continuous communication loop
14 while True:
15     # Enter message to send to server
16     message = input("Enter message for server: ")
17     client_socket.send(message.encode())
18
19     # Receive reply from server
20     data = client_socket.recv(1024).decode()
21     print("Server:", data)
22
23 # Close client socket (unreachable due to infinite loop)
24 client_socket.close()
25 |
```

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:

```
PS C:\Users\mehra> cd Desktop
PS C:\Users\mehra\Desktop> cd DS
PS C:\Users\mehra\Desktop\DS> python server.py
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:

```
PS C:\Users\mehra> cd Desktop
PS C:\Users\mehra\Desktop> cd DS
PS C:\Users\mehra\Desktop\DS> python client.py
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
Enter message for server: █
```

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.

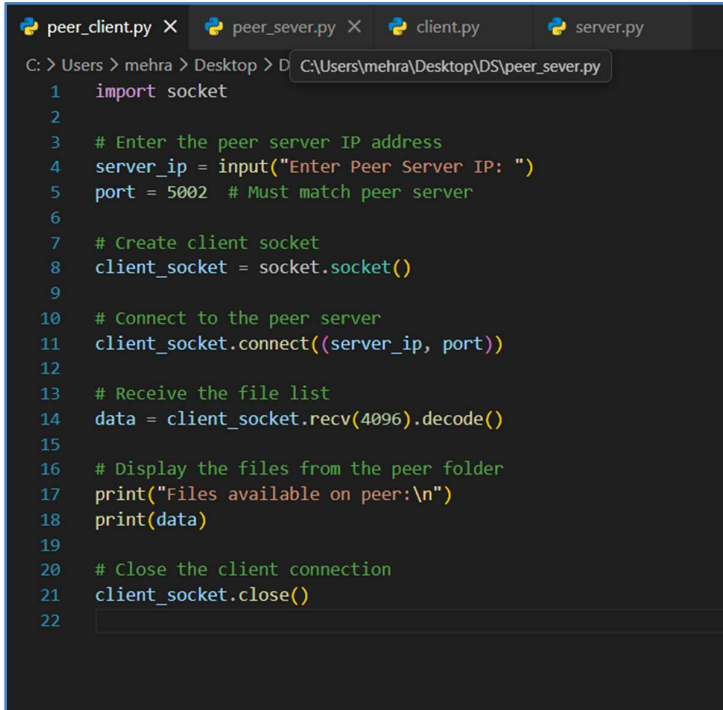
Program Code:

i) peer_server.py:



```
peer_sever.py X client.py peer_client.py server.py
C: > Users > mehra > Desktop > DS > peer_sever.py > ...
1 import socket
2 import os
3
4 # Get local machine IP
5 host = socket.gethostname(socket.gethostname())
6 port = 5002
7
8 # Folder to list files from
9 folder_path = "C:/Users/Public/Documents"
10
11 # Create server socket
12 server_socket = socket.socket()
13
14 # Bind to IP and port
15 server_socket.bind((host, port))
16
17 # Start listening for connections
18 server_socket.listen(1)
19
20 print("Peer Server IP:", host)
21 print("Serving files from:", folder_path)
22
23 # Accept client connection
24 conn, address = server_socket.accept()
25 print("Connected to:", address)
26
27 # Get actual file names from folder
28 files = os.listdir(folder_path)
29
30 # Convert list to string format
31 file_list = "\n".join(files)
32
33 # Send file names to client
34 conn.send(file_list.encode())
35
36 # Close connection
37 conn.close()
38
```

i)peer_client.py:



```
peer_client.py X peer_sever.py X client.py server.py
C: > Users > mehra > Desktop > D C:\Users\mehra\Desktop\DS\peer_sever.py
1 import socket
2
3 # Enter the peer server IP address
4 server_ip = input("Enter Peer Server IP: ")
5 port = 5002 # Must match peer server
6
7 # Create client socket
8 client_socket = socket.socket()
9
10 # Connect to the peer server
11 client_socket.connect((server_ip, port))
12
13 # Receive the file list
14 data = client_socket.recv(4096).decode()
15
16 # Display the files from the peer folder
17 print("Files available on peer:\n")
18 print(data)
19
20 # Close the client connection
21 client_socket.close()
22
```

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
AdobeGCDData
AdobeGCInfo
AdobeInstalledCodecs
Blackmagic Design
desktop.ini
My Music
My Pictures
My Videos
OnlineFix
uPlay

Screenshots of Output:

Peer_Server:

```
PS C:\Users\mehra\Desktop\DS> python peer_sever.py
Peer Server IP: 11.12.2.179
Serving files from: C:/Users/Public/Documents
Connected to: ('11.12.2.179', 53496)
PS C:\Users\mehra\Desktop\DS> █
```

Peer_Client:

```
PS C:\Users\mehra> cd Desktop
PS C:\Users\mehra\Desktop> cd DS
PS C:\Users\mehra\Desktop\DS> python peer_client.py
Enter Peer Server IP: 11.12.2.179
Files available on peer:

Adobe
AdobeGCDData
AdobeGCInfo
AdobeInstalledCodecs
Blackmagic Design
desktop.ini
My Music
My Pictures
My Videos
OnlineFix
uPlay
PS C:\Users\mehra\Desktop\DS> █
```

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.