20CYS402 – Distributed Systems & Cloud Computing

Lab 1

Name: Chitra Harini **Date:** 18/06/2025

Roll no: CH.EN.U4CYS22010 **Lab - 1**

Github Link:

Question 1.1: Construct a program for Client-Server communication to transmit user-entered messages from client to server and vice versa using different IP addresses.

Objective

To implement socket-based client-server communication where messages are exchanged between two systems using different IP addresses.

Program Code

<u>Server (server.py)</u>

```
import socket
server_ip = "0.0.0.0"
server_port = 5001
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port))
server_socket.listen(1)
print("Server is waiting for connection...")
conn, addr = server_socket.accept()
print("Connected by", addr)
while True:
    msg = conn.recv(1024).decode()
    if msg.lower() == 'exit':
        print("Client exited.")
        break
    print("Client:", msg)
    reply = input("Server: ")
    conn.sendall(reply.encode())
    if reply.lower() == 'exit':
        break
conn.close()
```

<u>Client (client.py)</u>

```
import socket
server_ip = input("Enter Server IP Address: ")
server_port = 5001
```

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_ip, server_port))
while True:
    msg = input("Client: ")
    client_socket.sendall(msg.encode())
    if msg.lower() == 'exit':
        break
    reply = client_socket.recv(1024).decode()
    print("Server:", reply)
    if reply.lower() == 'exit':
        break
client_socket.close()
```

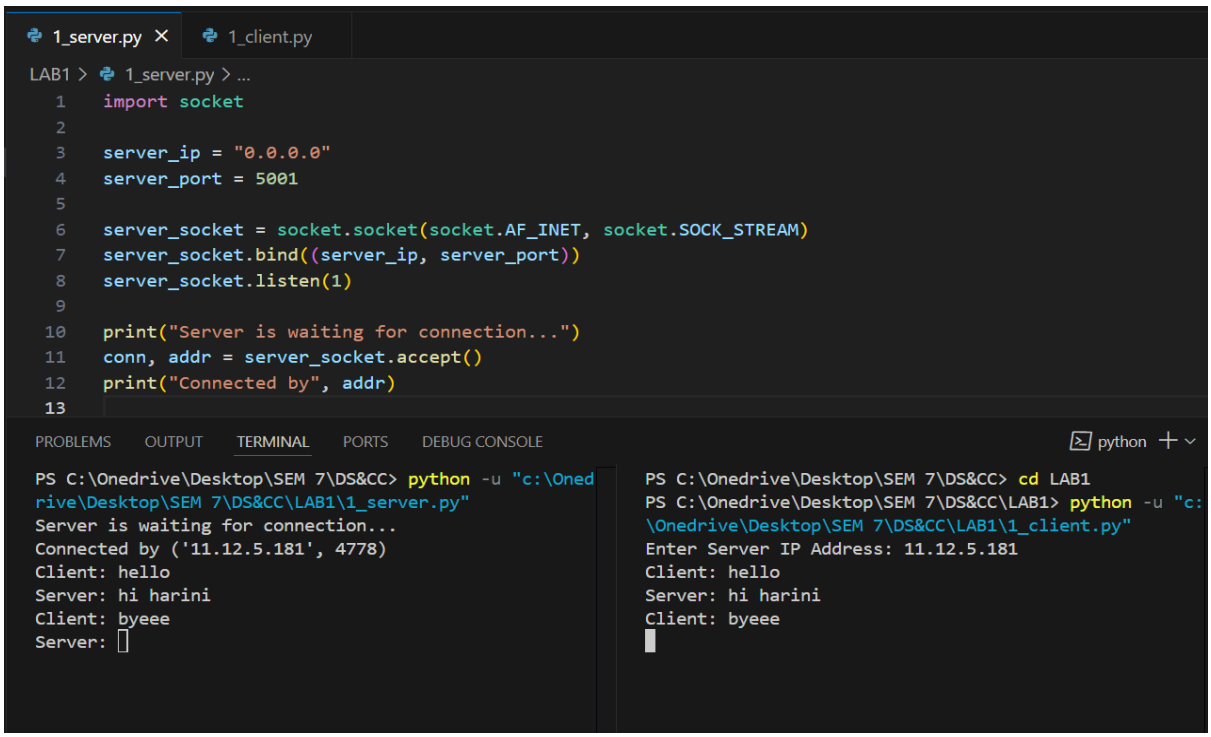
Explanation

- The server binds to a port and listens for client connections.
- The client connects to the server using its IP address.
- Once connected, both client and server can send and receive messages.
- The loop continues until either party types "exit".

Input/Output Example

Client Input: Hello
Server Output: Client: Hello
Server Input: Hi
Client Output: Server: Hi

Screenshot



Conclusion

This task demonstrated how to establish a TCP socket connection between two machines using different IPs, enabling bidirectional communication.

Question 1.2: Peer-to-Peer Communication to List Local Files

Objective

To develop a peer-to-peer application where one peer requests a file list from another peer's local directory using socket programming.

Program Code

Server.py

```
import socket
import os
def start_file_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('192.168.56.1', 5000))
    server_socket.listen(1)
    print("File server running...")
    conn, addr = server_socket.accept()
    print(f"Connected by {addr}")
    files = os.listdir('.')  # List files in current directory
    file_list = "\n".join(files)
    conn.send(file_list.encode())
    conn.close()
start_file_server()
```

Client.py

```
import socket
def connect_to_peer():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('192.168.56.1', 5000))
    data = client_socket.recv(4096).decode()
    print("Available files on server:\n", data)
    client_socket.close()
connect_to_peer()
```

Explanation

- The program acts as both server and client using threads.
- One peer connects to another and sends a "LIST" command.
- The receiving peer responds with a list of files in its current directory.

- This models a simple peer-to-peer interaction.

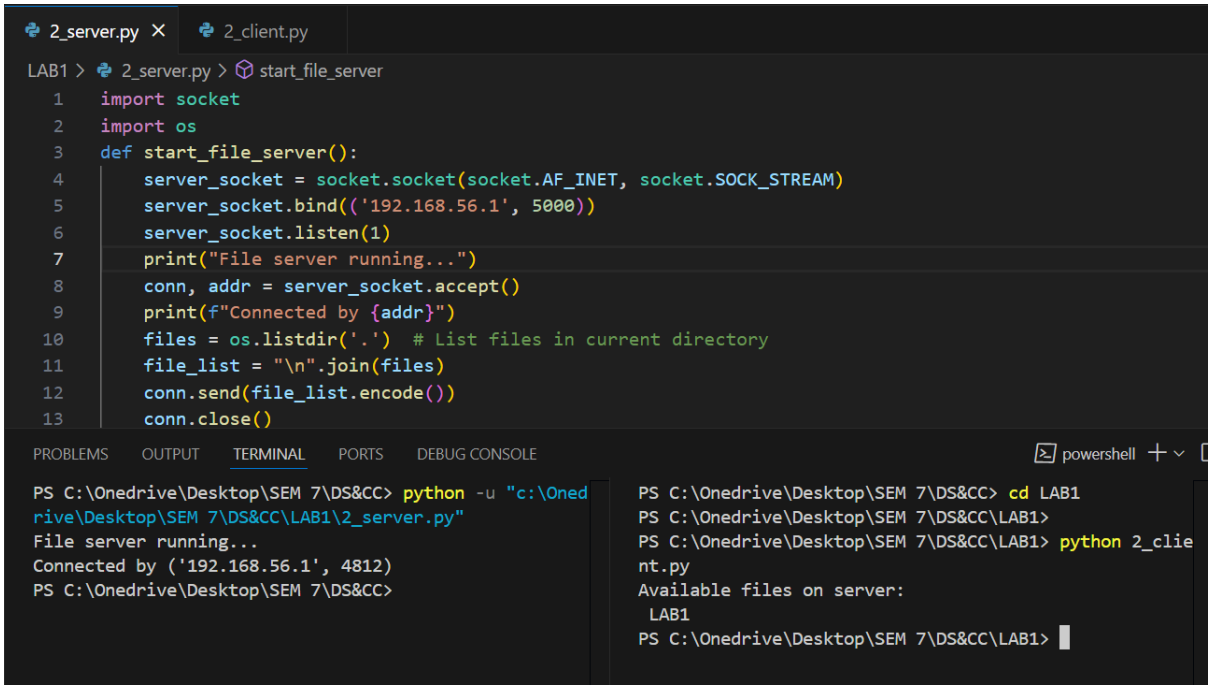
Input/Output Example

Input (to peer): LIST

Output (from peer):

LAB1

Screenshot



Conclusion

This exercise helped implement peer-to-peer communication and directory listing using socket programming and multithreading, simulating real-world P2P behavior.

Question 1.3: Remote Procedure Call (RPC) Based Age Calculator

Objective

To implement a distributed system using RPC where a client sends a date of birth, and the server calculates and returns the age.

Program Code

<u>RPC Server (rpc_server.py)</u>

```
from xmlrpc.server import SimpleXMLRPCServer
from datetime import datetime
def calculate_age(dob_str):
    dob = datetime.strptime(dob_str, '%Y-%m-%d')
    today = datetime.now()
    age = today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))
    return age
```

```
server = SimpleXMLRPCServer(("192.168.56.1", 5000))
print("RPC Server is listening on port 5000...")
server.register_function(calculate_age, "get_age")
server.serve_forever()
```

RPC Client (rpc_client.py)

```
import xmlrpc.client
proxy = xmlrpc.client.ServerProxy("http://192.168.56.1:5000/")
dob = input("Enter your Date of Birth (YYYY-MM-DD): ")
age = proxy.get_age(dob)
print(f"Your age is: {age}")
```

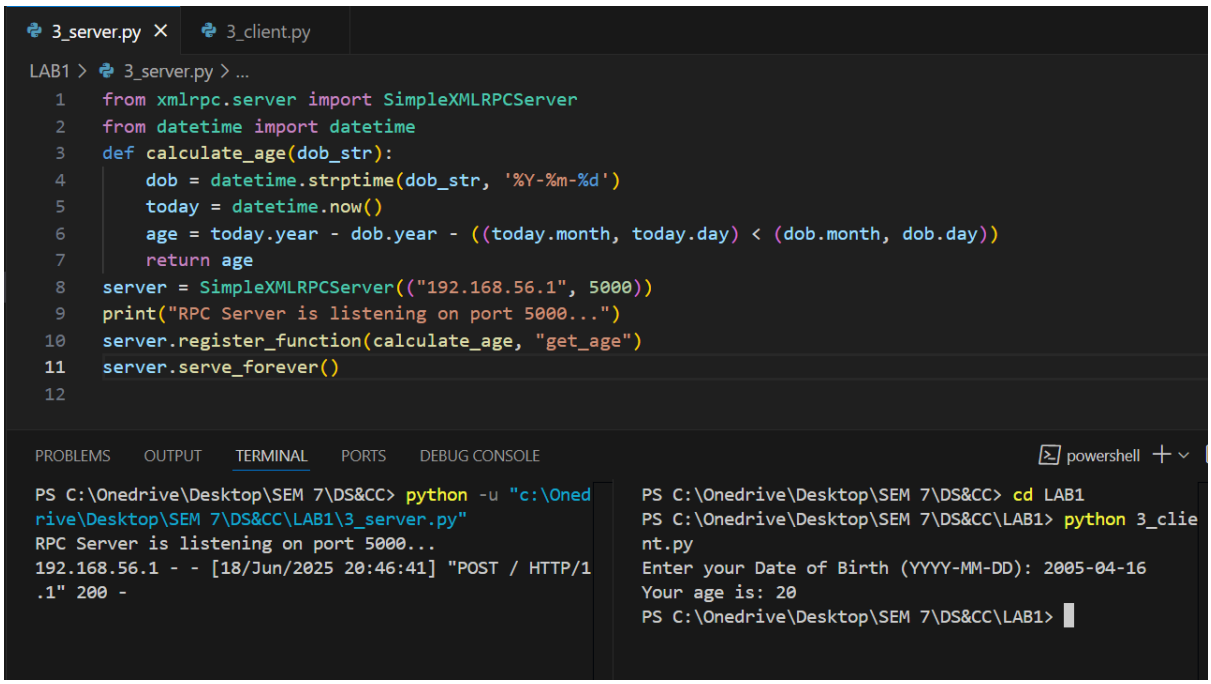
Explanation

- The server runs an XML-RPC service that exposes the get_age function.
- The client connects to this server and passes a date of birth.
- The server calculates age and returns it as a response.

Input/Output Example

Input (DOB): 2005-04-16
Output (Age): 25

Screenshot



Conclusion

This task demonstrated how to use XML-RPC for implementing a simple distributed system. It illustrated how functions can be executed remotely and results returned over a network.