TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Lab Report

of

Distributed System

on

Client-Server Implementation Using Python

Submitted By:

Prinsa Joshi

(THA076BCT031)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

June, 2023

TITLE: SIMPLE CLIENT-SERVER IMPLEMENTATION USING PYTHON

THEORY:

Client-server architecture is a fundamental concept in networking and distributed computing. It is widely used in various applications, including web servers, database systems, email systems, and more. The basic components of a client-server implementation include:

- Client: A client is a program or device that requests services from a server. It can be a desktop application, a web browser, a mobile app, or any other entity that needs to access resources or perform tasks provided by a server. The client initiates a connection to the server, sends requests for specific services, and waits for the server's response.
- Server: A server is a program or device that provides services to clients. It listens for incoming requests from clients, processes those requests, and sends back the corresponding responses. Servers have dedicated hardware and software resources to handle multiple client connections simultaneously. They are designed to be robust, scalable, and capable of serving numerous clients concurrently.
- Communication Protocol: To facilitate communication between the client and server, a communication protocol is used. A protocol defines the rules and formats for exchanging messages between the client and server, ensuring that both parties understand and interpret the messages correctly. Some commonly used protocols in client-server communication include HTTP (Hypertext Transfer Protocol), TCP/IP (Transmission Control Protocol/Internet Protocol), and SMTP (Simple Mail Transfer Protocol).

Steps in Client-Server Interaction:

The interaction between a client and server typically follows these steps:

- Establishing Connection: The client establishes a network connection with the server.
- Sending Requests: Once the connection is established, the client sends requests to the server.
- Processing Requests: The server receives the client's requests and processes

them according to the requested services.

- Generating Responses: After processing the request, the server generates the corresponding response.
- Sending Responses: The server sends the response back to the client through the established connection.
- Closing the Connection: Once the interaction is complete, either the client or server (or both) can initiate the closure of the connection.

Client-server architecture provides several benefits, including:

- Scalability: Servers can handle multiple client connections simultaneously, allowing for scalable systems that can serve a large number of users.
- Centralized Resources: Servers consolidate resources and provide centralized access, enabling efficient management and utilization of shared resources.
- Security: By centralizing access control and authentication, client-server architectures can enhance security measures, protecting sensitive data and limiting unauthorized access.
- Maintenance and Updates: Servers can be updated or maintained independently, ensuring minimal disruption to clients accessing the services.
- Fault Tolerance: Servers can be designed with redundancy and fault tolerance mechanisms to ensure

CODE:

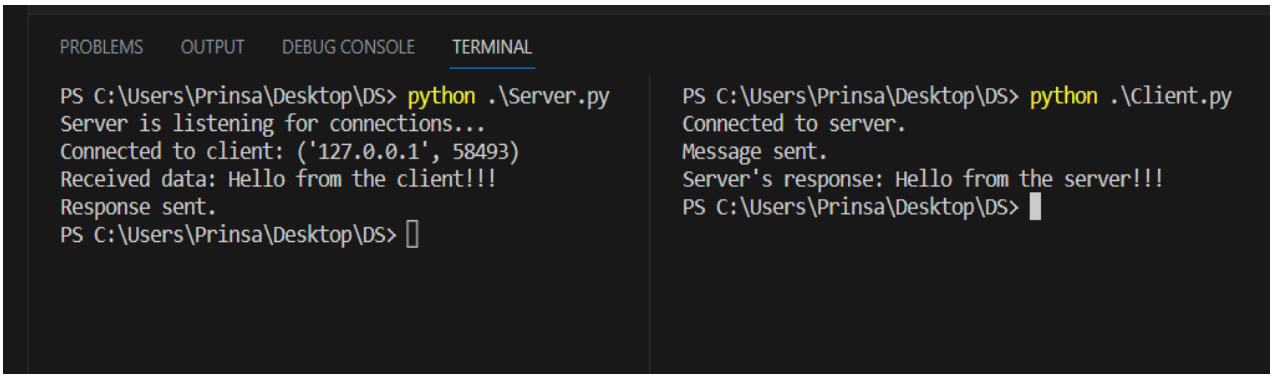
Server.py

```python
import socket

# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Define the host and port
host = 'localhost'
port = 12345

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

# Listen for incoming connections
server_socket.listen(1)
print("Server is listening for connections...")

# Accept a client connection
client_socket, addr = server_socket.accept()
print("Connected to client:", addr)

# Receive data from the client
data = client_socket.recv(1024).decode()
print("Received data:", data)

# Send a response back to the client
response = "Hello from the server!!!"
client_socket.send(response.encode())
print("Response sent.")

# Close the connection
client_socket.close()
server_socket.close()
```

Client.py

```python
import socket

# Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Define the host and port to connect to
host = 'localhost'
port = 12345

# Connect to the server
client_socket.connect((host, port))
print("Connected to server.")

# Send data to the server
message = "Hello from the client!!!"
client_socket.send(message.encode())
print("Message sent.")

# Receive the server's response
response = client_socket.recv(1024).decode()
print("Server's response:", response)

# Close the connection
client_socket.close()

```

OUTPUT:



CONCLUSION:

Through this experiment, I have learned the basics of client-server architecture and its implementation. I gained knowledge about establishing network connections, sending and receiving messages between clients and servers, and the importance of communication protocols. This experience has deepened my understanding of networked systems and their practical applications.