Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL405**	**Course Name:**	**Skills Based Python Programming Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**7**
Title of the Experiment:	**To implement a simple socket for basic information exchange between server and client.**
Date of Performance:	**04/03/2025**
Date of Submission:	**04/03/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mr. Raunak Joshi

Signature :

Date:

Aim: To implement a simple socket for basic information exchange between server and client.

Theory:

Sockets provide a way for programs to communicate with each other over a network. In Python, socket programming enables the exchange of data between a server and one or more clients. This is fundamental for network communication, including web servers, chat applications, and remote procedure calls.

Introduction to Socket Programming:

A socket is an endpoint in a communication channel used to send and receive data over a network. Socket programming involves the creation, configuration, and usage of sockets to enable communication between devices or processes over a network.

Python's socket module provides a low-level interface for network communication using the Berkeley sockets API, which is compatible with most operating systems.

Types of Sockets:

1. **Stream Sockets (SOCK_STREAM):**
    - Uses Transmission Control Protocol (TCP).
    - Reliable, connection-oriented communication.
    - Ensures data delivery and maintains order.
    - Suitable for applications like web servers, file transfers, and messaging.
2. **Datagram Sockets (SOCK_DGRAM):**
    - Uses User Datagram Protocol (UDP).
    - Connectionless and faster but unreliable (no guarantee of delivery or order).
    - Suitable for real-time applications like gaming and video streaming.

This theory focuses on TCP sockets (SOCK_STREAM) for reliable communication.

Socket Communication Model:

1. **Server Side:**
    - Creates a socket.
    - Binds the socket to an IP address and port.
    - Listens for incoming connections.
    - Accepts a connection from a client.
    - Receives and sends data.
    - Closes the connection.
2. **Client Side:**
    - Creates a socket.
    - Connects to the server using its IP address and port.
    - Sends and receives data.
    - Closes the connection.

Python's `socket` Module:

The `socket` module provides the necessary functions to implement networking:

- **`socket.socket()`**: Creates a socket object.
- **`socket.bind()`**: Binds the socket to an address (IP and port).
- **`socket.listen()`**: Listens for incoming connections (server side).
- **`socket.accept()`**: Accepts a connection (server side).
- **`socket.connect()`**: Initiates a connection (client side).
- **`socket.send()` and `socket.recv()`**: Sends and receives data.
- **`socket.close()`**: Closes the socket.

Basic Server-Client Architecture:

In a basic information exchange system:

- The **server** waits for a client to connect, then sends and receives data.
- The **client** connects to the server, exchanges data, and disconnects.

This architecture is suitable for:

- Chat applications.
- Simple request-response systems.
- Data sharing between networked systems.

Key Concepts:

- **AF_INET:** Address family for IPv4.
- **SOCK_STREAM:** Socket type for TCP communication.
- **bind()**: Assigns an IP and port to the socket.
- **listen()**: Puts the server in listening mode.
- **accept()**: Waits for an incoming client connection.
- **connect()**: Initiates a connection from client to server.
- **recv() and sendall()**: For receiving and sending data

Implementation:

client.py

```
import socket

PORT = 5050
SERVER = '192.168.21.212'
SERVER = socket.gethostbyname (socket.gethostname ())
ADDR = (SERVER, PORT)
FORMAT = 'utf-8'
HEADER = 64
DISCONNECT_MESSAGE = '!DISCONNECT'

client = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
client.connect (ADDR)

def send (messages):
    message = messages.encode (FORMAT)
    messageLength = len (message)
    sendLength = str (messageLength).encode (FORMAT)
```

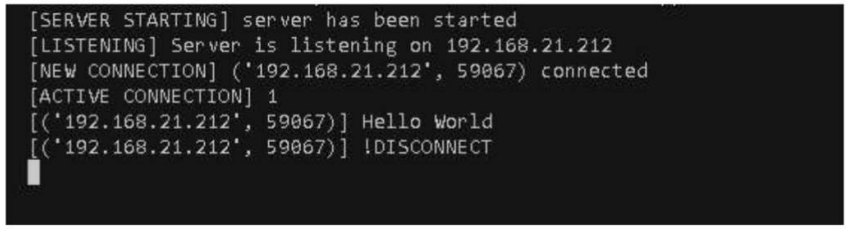
```
    sendLength += b' ' * (HEADER - len (sendLength))
    client.send (sendLength)
    client.send (message)
    print (client.recv (2045).decode (FORMAT))

if __name__ == '__main__':
    send ('Hello World')
    send (DISCONNECT_MESSAGE)
```

server.py

```
import socket, threading

PORT = 5050
SERVER = socket.gethostbyname (socket.gethostname ())
ADDR = (SERVER, PORT)
FORMAT = 'utf-8'
HEADER = 64
DISCONNECT_MESSAGE = '!DISCONNECT'

server = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
server.bind (ADDR)

def clientHandling (connection, address):
    print (f'[NEW CONNECTION] {address} connected')
    isConnected = True
    while isConnected:
        messageLength = connection.recv (HEADER).decode (FORMAT)
        if messageLength:
            messageLength = int (messageLength)
            message = connection.recv (messageLength).decode (FORMAT)
            if message == DISCONNECT_MESSAGE:
                isConnected = False
            print (f'[{address}] {message}')
            connection.send ('Message Recieved'.encode (FORMAT))
    connection.close ()

def start ():
```

```
    server.listen ()
    print (f'[LISTENING] Server is listening on {SERVER}')
    while True:
        connection, address = server.accept ()
        thread = threading.Thread (target=clientHandling, args=(connection,
address))
        thread.start ()
        print (f'[ACTIVE CONNECTION] {threading.active_count () - 1}')


print ('[SERVER STARTING] server has been started')
start ()
```

Output -



Conclusion:

Socket programming is essential for network communication, enabling data exchange between server and client applications. Using Python's socket module, developers can efficiently build networking solutions for a variety of applications, ranging from messaging systems to remote servers.