DHARSHINI.S 241901026

**EX NO:2** DATE:19/08/2025

**STUDY OF SOCKET PROGAMMING IN PYTHON**

**AIM:**

To study the concepts of socket programming in python.

**DESCRIPTION:**

**INTRODUCTION:**

In networks, the services provided to the user follow the traditional client/server

model. One computer acts as a server to provide a certain service and another computer

represents the client side which makes use of this service. In order to communicate over

the network a network socket comes into play, mostly only referred to as a socket.

**SOCKET DEFINITION:**

A network socket is an endpoint of a two-way communication link between two

programs or processes - client and server in our case - which are running on the network.

This can be on the same computer as well as on different systems which are connected

via the network. Both client/server communicate with each other by writing to or reading

from the network socket. The technical equivalent in reality is a telephone

number of the telephone line, or a contract in case of cell phones.

**SOCKET PROGRAMMING IN PYTHON:**

Python's core networking library is Socket Module.

Python's socket module has both class-based and instances-based methods.

Class-based method is an intuitive approach which doesn't need an instance of a

socket object.

For example, in order to print a machine's IP address, you don't need a socket

object. Instead, just call the socket's class-based methods.

In instance-based method, if some data needed to be sent to a server application,

it is more intuitive to create a socket object to perform that explicit operation.

This module has everything you need to build socket servers and clients.

ALOGRITHM:

**Server Algorithm:**

1. **Initialize Socket:**
   * Create a socket using TCP/IP protocol (SOCK\_STREAM) and IPv4 addressing (AF\_INET).
2. **Bind the Socket:**
   * Bind the server socket to a specific address (localhost) and port number (e.g., 55555).
3. **Listen for Connections:**
   * Set the server socket to listen for incoming connections from clients. Specify the backlog queue size (e.g., 3).
4. **Accept Client Connections:**
   * Wait for a client to connect. Once a client connects, accept the connection, which returns a new socket (clientfd) for communication and the address of the client.
5. **Receive Data from Client:**
   * Receive the message from the client socket, decode it from bytes to a string.

Process and Respond:

* + (Optional) Process the received message as needed (e.g., echo the message back).
  + Send the response (or echoed message) back to the client via the same socket.

1. **Check for Continuation:**
   * Ask the server operator if they want to continue accepting more clients or shut down the server.
2. **Close Connection:**
   * After responding to the client, close the individual client connection.
   * If the server is to stop, break out of the loop and close the server socket.

**Client Algorithm:**

1. **Initialize Socket:**
   * Create a socket using TCP/IP protocol (SOCK\_STREAM) and IPv4 addressing (AF\_INET).
2. **Connect to the Server:**
   * rnumber (e.g., 55555).
3. **Send Data to Server:**
   * Accept user input (the message to send) and convert it to bytes.
   * Send the message to the server via the established socket connection.
4. **Receive Server Response:**
   * Wait for a response from the server. This will typically be the same message that was sent (in the case of an echo server).
5. **Display Response:**
   * Decode the received message from bytes to string and display it to the user.
6. **Close the Connection:**
   * Once the response is received and displayed, closed.

**Code:**

import socket

sockfd=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

print('Socket Created')

sockfd.bind(('localhost',55555))

sockfd.listen(3)

print('Waiting for connections')

while True:

clientfd,addr=sockfd.accept()

receivedMsg=clientfd.recv(1024).decode()

print("Connected with ",addr)

print("Message Received from Client: ",receivedMsg)

clientfd.send(bytes(receivedMsg,'utf-8'))

print("Message reply sent to Client!")

print("Do you want to continue(type y or n):")

choice=input()

if choice=='n':

break

import socket

clientfd=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

clientfd.connect(('localhost',55555))

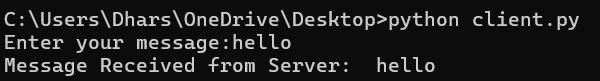
name=input("Enter your message:")

clientfd.send(bytes(name,'utf-8'))

print("Message Received from Server: ",clientfd.recv(1024).decode())

A screen shot of a computer

AI-generated content may be incorrect.



**Result:**

The server echoes the client's message back after receiving it, and the client displays the server's echoed response.