|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Electronics Application Using Python Programming** | **Semester:** | **V** |
| **Date of Performance:** | **03 / 09 / 2024** | **Batch No:** | **APPA-2** |
| **Faculty Name:** | **Prof. Deepa Jain** | **Roll No:** | **16014022096** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **/25** |

**Experiment No: 6**

**Title: Study of Socket Programming**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| To understand the socket concept in networking |

|  |
| --- |
| **COs to be achieved:** |
| **CO1**: Understand socket based and web service approaches to inter process communications. |

|  |
| --- |
| **Theory** |
| **Introduction:**  A Socket is an end point in a communication link between separate processes. In Python sockets are objects which provide a way of exchanging information between two processes in a straight forward and platform independent manner. In this chapter we will introduce the basic idea of socket communications and then presents a simple socket server and client application  Socket to Socket Communication:  When two operating system level processes wish to communicate, they can do so via sockets. Each process has a socket which is connected to the others socket. One process can then write information out to the socket, while the second process can read information in from the socket. Associated with each socket are two streams, one for input and one for output. Thus, to pass information from one process to another, you write that information out to the output stream of one socket object and read it from the input stream of another socket object (assuming the two sockets are connected).  To set up the connection, one process must be running a program that is waiting for a connection while the other must try to connect up to the first program. The first is referred to as a server socket while the second just as a socket. For the second process to connect to the first (the server socket) it must know what machine the first is running on and which port it is connected to.    For example, in the above diagram the server socket connects to port 8084. In turn the client socket connects to the machine on which the server is executing and to port number 8084 on that machine. |

|  |
| --- |
| **Tools required:** |
| Any python editor tool |

|  |
| --- |
| **Code:** |
| * Write a Python Client Server application using socket library   CODES:  SERVER 1  import socket  ob = socket.socket()  ob.bind(('localhost', 2301))  ob.listen(4)  print("server is ready to listen")  clientobject, add=ob.accept()  print("server is ready to accept the connection")  print("connected with this address: ", add)  ob.close()  CLIENT 1  import socket  ob = socket.socket()  ob.connect(('localhost', 2301))  print("client is ready to accept the connection")  ob.close()  SERVER 2  import socket  ob = socket.socket()  ob.bind(('localhost', 2301))  ob.listen(4)  print("server is ready to listen")  clientobject, add=ob.accept()  print("server is ready to accept the connection")  print("connected with this address: ", add)  gotmsg = clientobject.recv(1024)  gotmsg.decode('utf-8')  print(gotmsg)  ob.close()  CLIENT 2  import socket  ob = socket.socket()  ob.connect(('localhost', 2301))  print("client is ready to accept the connection")  msg = 'hello dazai san here'  ob.send(msg.encode('utf-8'))  ob.close()  SERVER 3  import socket  ob = socket.socket()  ob.bind(('localhost', 2301))  ob.listen(4)  print("server is ready to listen")  clientobject, add=ob.accept()  print("server is ready to accept the connection")  print("connected with this address: ", add)  conn = True  while conn:      gotmsg = clientobject.recv(1024)      gotmsg.decode('utf-8')      print(gotmsg)      if len(gotmsg) == 0:          conn = False  ob.close()  CLIENT 3  import socket  ob = socket.socket()  ob.connect(('10.0.104.42', 2301))  print("client is ready to send data")  conn = True  while conn:      msg = input("enter your message: ")      if msg == 'no':          conn = False      else:          ob.send(msg.encode('utf-8'))  ob.close()  SERVER 4  import socket  host = 'localhost'  port = 1234  server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  server\_socket.bind((host,port))  server\_socket.listen(4)  print(**f**"server listening on {host} : {port} ")  client\_socket, add = server\_socket.accept()  print(**f**"connected by {add}")  while True:      data = client\_socket.recv(1024)      if not data:          break      print(**f**"client :  {data.decode()}")      msg = input("Server: ")      client\_socket.sendall(msg.encode())  server\_socket.close()  client\_socket.close()    CLIENT 4  import socket  host = 'localhost'  port = 1234  client\_socket = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)  client\_socket.connect((host, port))  while True:      message = input("Client: ")      client\_socket.sendall(message.encode())      data = client\_socket.recv(1024)      print(**f**"Server: {data.decode()}")      if message.lower() == 'exit' or data.decode().lower() == 'exit':          break  client\_socket.close() |

|  |
| --- |
| **Output:** |
| OUTPUT 1      OUTPUT 2      OUTPUT 3      OUTPUT 4      OUTPUT 4 (client on laptop and server on mobile): |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. Explain different types of Sockets and its domain   ANS:  **Stream Sockets (SOCK\_STREAM)**: These provide reliable, two-way, connection-based communication (like a phone call). They use TCP (Transmission Control Protocol) to send data.  **Datagram Sockets (SOCK\_DGRAM)**: These provide connectionless communication (like sending a letter). They use UDP (User Datagram Protocol) and are faster but less reliable.  **Socket Domains:**  **AF\_INET**: Used for IPv4 Internet protocols (e.g., sending data over the internet using an IP address).  **AF\_INET6**: Used for IPv6 Internet protocols.  **AF\_UNIX**: Used for communication between processes on the same machine (local inter-process communication).   1. Explain Socketserver module and how it is different from socket module   ANS:  **Socket module**: Provides basic functions for creating and managing network connections. You handle most details (like managing connections and threads) manually.  **SocketServer module**: Simplifies the creation of server applications. It provides classes like TCPServer and UDPServer that automatically manage connections and handle incoming requests. It's higher-level and more user-friendly than socket, making it easier to create multi-client servers. |

|  |
| --- |
| **Conclusion:** |
| We have successfully explored the fundamentals of socket programming, learning how to establish connections between client and server applications. We successfully implemented and tested socket-based communication to facilitate inter-process communication. This enhanced our understanding of networking concepts in Python. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |