Government College of Engineering, Jalgaon

(An Autonomous Institute of Government of Maharashtra)

Name : Viraj Koli **Semester :** V **PRN :** 2241032

Class : T. Y. B.Tech Computer **Academic Year :** 2024-25 **Subject : CNTL**

CourseTeacher : Mrs. Prajakta Sawale

Date of Performance : **Date of Completion :**

Practical No : 1 (A)

Writeup	Correctness of program.	Documentation of program	Viva	Attendence For practical	Timely completion	Total	Dated sign of course teacher
4	2	2	5	2	5	20	

Aim : **UNIX Sockets: WAP program in C/C++ /Python/Java sockets API.**

a. TCP sockets

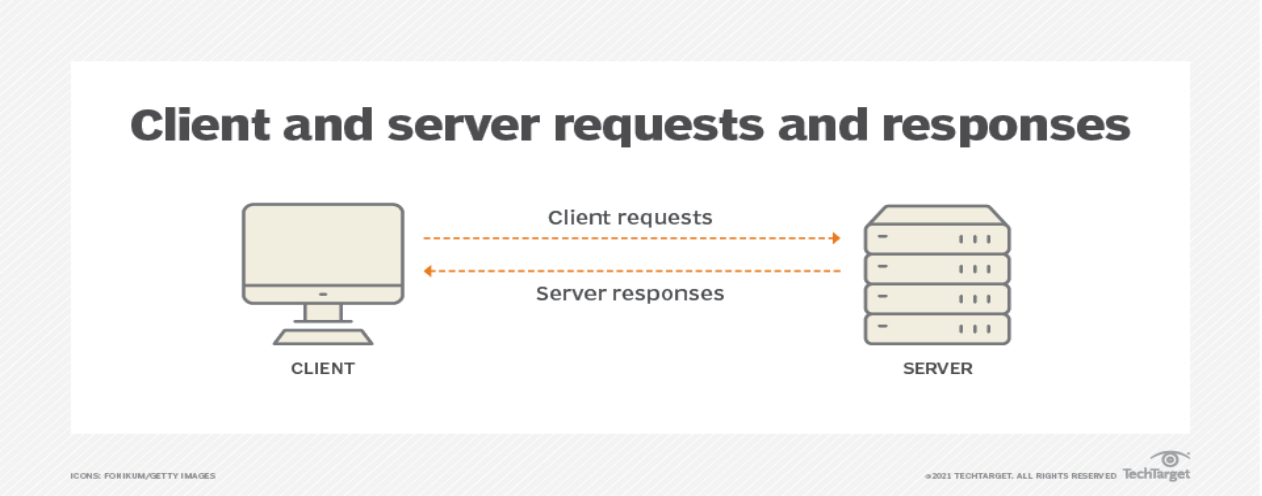
b. UDP sockets

Server accepts operation and floating point numbers from the clients; performs arithmetic operations and sends the result back to the client. Server applications must handle at least five clients simultaneously. Both the server and client should display input and output numbers as well as the operation. The server and client processes should be run on different machines. During evaluation , students will demonstrate via creating multiple client processes on different machines.

Software Used :- Idle Python 3

Theory :-

While two network applications could start at the same time, it is not practical to expect this. As a result, it's more logical to design network applications to carry out complementary operations one after the other, rather than at the same time. The server starts first and waits to receive data, followed by the client, which initiates the connection by sending the first packet. Once this initial communication is established, both the client and server can send and receive data.



What are Unix Sockets?

Imagine a two-way tunnel, not of dirt but of data, built within the digital terrain of your Linux system. That's essentially what a Unix socket represents. It's a software endpoint facilitating bidirectional communication between processes, regardless of their location within the system or even beyond its borders. Unix sockets offer two distinct flavors:

Socket Types and Their Roles

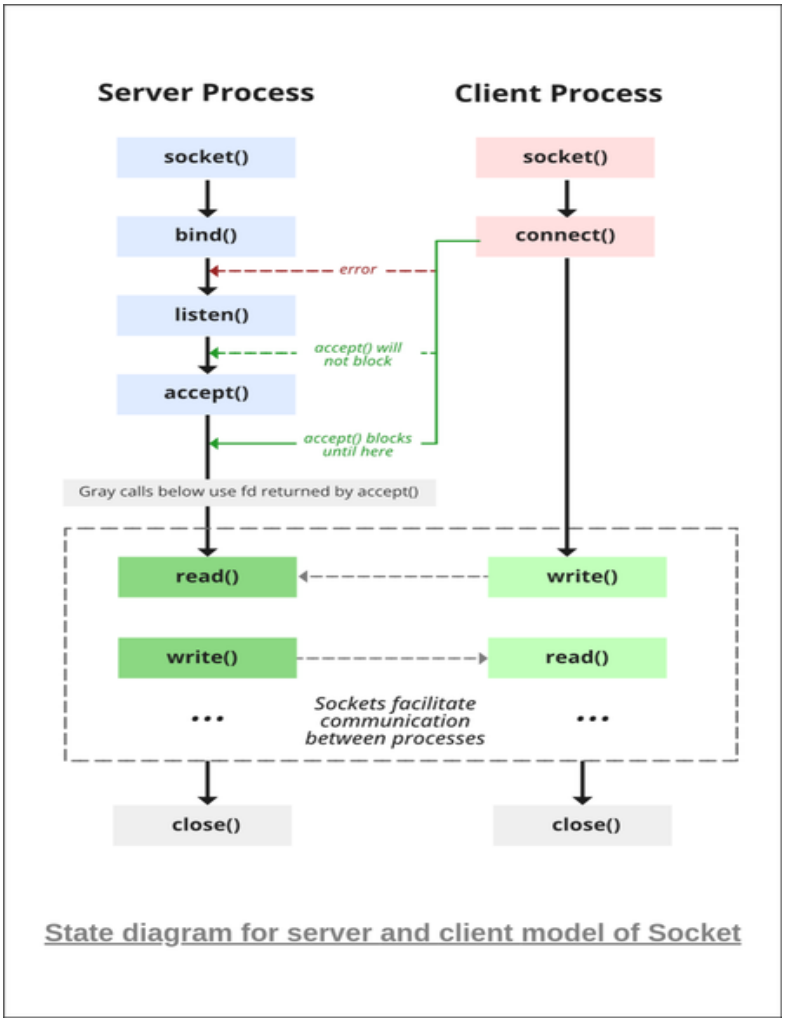
The type of socket dictates its communication style:

1. Stream Sockets (SOCK_STREAM) : Think of them as flowing rivers, ensuring reliable, sequential data transmission. Imagine sending a document, with each byte arriving in the correct order, much like the pages of a book.

2. Datagram Sockets (SOCK_DGRAM) : Picture them as mail delivery, where individual packets carry the data. While faster, they lack guaranteed delivery and order, like postcards mailed independently.

Creating UNIX Sockets

Sockets enable communication between two separate processes, either on the same machine or across different machines or networks. Essentially, they provide a method for exchanging data between computers. In Unix systems, all input/output operations are performed by reading from or writing to a file descriptor. A socket acts as the endpoint for communication, with each process having its own socket and corresponding socket descriptor to handle these operations. This article will explore how to create a socket in a Unix environment and implement both client- and server-side programs.



State diagram for server and client model of Socket

Creating TCP server :-

1. **socket()**:
   - The server process starts by creating a socket using the socket() system call. This creates an endpoint for communication.
2. **bind()**:
   - Next, the server binds the socket to a specific IP address and port using the bind() function. This ensures that the server listens for incoming connections on a particular address and port.
   - If this step fails (e.g., if the port is already in use), an error occurs, and the process cannot continue.
3. **listen()**:
   - The server enters the listening state with the listen() function. This prepares the socket to accept incoming connections by specifying a backlog of pending connections.
4. **accept()**:
   - When a client attempts to connect, the server uses the accept() function to accept the incoming connection. This function blocks (i.e., waits) until a client makes a connection request.
   - Once the connection is accepted, a new socket descriptor is created, which is used for further communication with the client.
   - If the connection is successful, the server is now ready to read and write data through this new socket.
5. **read() / write()**:
   - The server reads data from the client using the read() function and sends data using the write() function. These operations are performed on the socket descriptor returned by accept().
6. **close()**:
   - Finally, when the communication is complete, the server closes the socket using the close() function, terminating the connection.

Creating a TCP Client :-

1. **socket()**:

   - Similar to the server, the client first creates a socket using the socket() system call.

2. **connect()**:

   - The client initiates a connection to the server using the connect() function. It provides the server's IP address and port number to establish a connection.

- Once the connection is successfully established, the client can proceed with sending and receiving data.

3. **write() / read()**:

   - The client writes data to the server using the write() function and reads responses using the read() function. These functions enable data exchange between the client and server.

4. **close()**:

   - After the communication is done, the client closes the socket using the close() function, ending the connection.

Conclusion :-

To conclude, sockets provide a highly efficient method for establishing strong communication. Inter-process communication (IPC) within the same host has strengthened Unix socket programming in C. It enables the creation of reliable, high-performance communication channels for processes running on the same machine using Unix domain sockets. The client starts a TCP socket, sets up the server address, and connects to receive messages.

Name & Sign of Course Teacher
Mrs.Prajakta DSawle