Experiment No: 13 **Date:**

SOCKET PROGRAMMING

AIM: Implement inter-process communication in Java using Socket Programming.

THEORY:

Introduction to Inter-Process Communication (IPC) using Socket Programming:

Inter-Process Communication (IPC) refers to the mechanisms and techniques used by processes running on a computer to exchange data and synchronize their actions. Socket programming is a popular approach to IPC in computer networks, allowing processes on different devices to communicate over a network. Sockets provide an interface for sending and receiving data between processes, enabling a wide range of network-based applications and services.

Advantages of IPC using Socket Programming:

1. **Platform Independence:** Socket programming enables IPC between processes running on different operating systems and hardware platforms, making it versatile and widely compatible.
2. **Flexibility:** Sockets support various communication protocols such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol), offering flexibility in data transmission based on application requirements.
3. **Scalability:** Socket-based IPC can scale to accommodate large numbers of processes and devices, making it suitable for both small-scale and enterprise-level applications.
4. **Network Transparency:** IPC using sockets abstracts the underlying network details, allowing processes to communicate over local or remote networks seamlessly without needing to manage low-level networking operations.

Disadvantages of IPC using Socket Programming:

1. **Complexity:** Socket programming can be complex, especially for developers unfamiliar with network programming concepts and protocols. It requires careful handling of network errors, data formatting, and protocol-specific considerations.
2. **Overhead:** Socket-based IPC may introduce additional overhead due to network communication, especially when transferring large amounts of data or establishing and maintaining network connections.
3. **Security Risks:** IPC over networks introduces security risks such as data interception, eavesdropping, and unauthorized access if proper encryption and authentication measures are not implemented.

4. **Performance Impact:** Network latency, bandwidth limitations, and congestion can impact the performance of IPC using sockets, particularly in scenarios with high data throughput or real-time communication requirements.

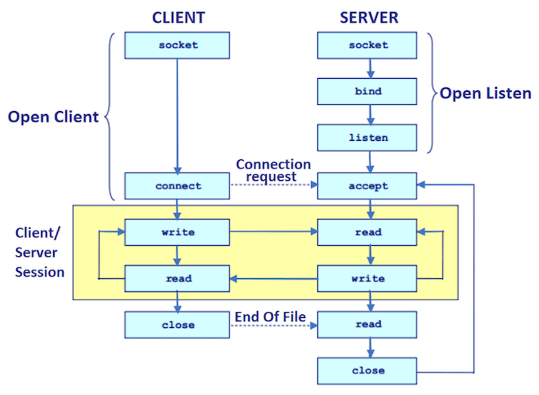
Real-World Application of IPC using Socket Programming:

1. **Client-Server Applications:** Socket programming is widely used in client-server applications, where processes on client devices communicate with server processes to request and receive services, such as web servers, email servers, and chat applications.
2. **Distributed Systems:** IPC using sockets enables communication between processes in distributed systems spanning multiple machines or devices, facilitating coordination and data exchange in cloud computing environments, IoT (Internet of Things) networks, and distributed databases.
3. **Network Protocols:** Many network protocols and services rely on socket programming for IPC, including HTTP (Hypertext Transfer Protocol) for web communication, SMTP (Simple Mail Transfer Protocol) for email transmission, and FTP (File Transfer Protocol) for file sharing.

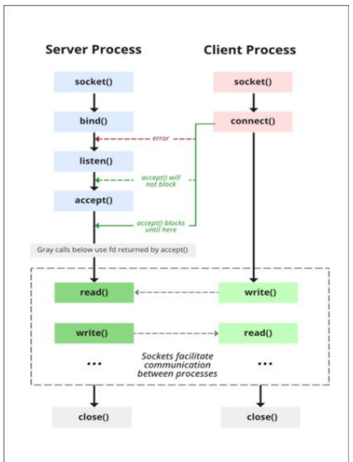
Example:

Consider a chat application where users can exchange messages in real-time over a network. The application consists of a client component running on users' devices and a server component hosted on a central server. The client and server processes communicate using TCP sockets established over the network.

When a user sends a message, the client process initiates a TCP connection to the server process using a socket. The server receives the message, processes it, and then broadcasts it to all connected clients. Each client receives the message through its own TCP socket connection and displays it to the user.



SOCKET API



State diagram for server and client model of Socket

CODES:

//Server side Java Code

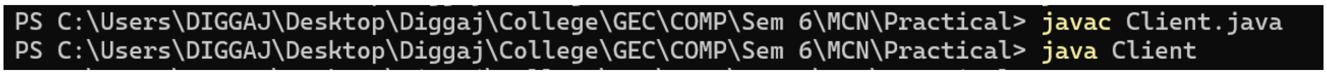
```
import java.io.*;
import java.net.*;
public class Server {
   public static void main(String[] args) {
      int portNumber = 12345; // Port number the server will listen on
      try {
         // Create a ServerSocket object
         ServerSocket serverSocket = new ServerSocket(portNumber);
         System.out.println("Server is running and waiting for client connection...");
         // Wait for a client connection
         Socket clientSocket = serverSocket.accept();
         System.out.println("Client connected: " + clientSocket.getInetAddress());
         // Get the input stream of the client socket
         InputStream inputStream = clientSocket.getInputStream();
         BufferedReader in = new BufferedReader(new InputStreamReader(inputStream));
         // Read the message from the client
         String message = in.readLine();
         System.out.println("Received message from client: " + message);
         // Close the sockets
         clientSocket.close();
         serverSocket.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

//Client side Java Code

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {
        String serverAddress = "172.26.10.28"; // Server's IP address
        int serverPort = 12345; // Server's port number

        try {
            // Create a socket connecting to the server
            Socket socket = new Socket(serverAddress, serverPort);

            // Get the output stream of the socket
            OutputStream outputStream = socket.getOutputStream();

            // Create a PrintWriter to write messages to the output stream
            PrintWriter out = new PrintWriter(outputStream, true);

            // Send a message to the server
            out.println("Hello, server! , I am Diggaj Ugvekar");

            // Close the socket
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

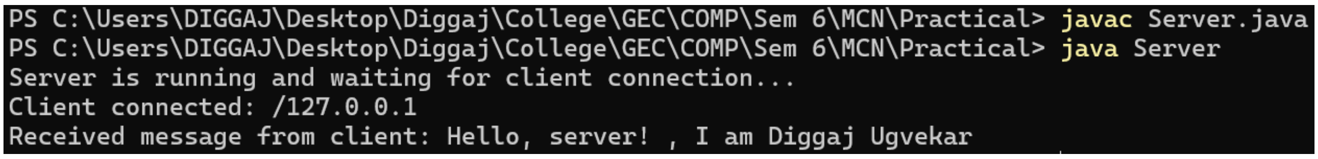
OUTPUT:

Client side code output



```
PS C:\Users\DIGGAJ\Desktop\Diggaj\College\GEC\COMP\Sem 6\MCN\Practical> javac Client.java
PS C:\Users\DIGGAJ\Desktop\Diggaj\College\GEC\COMP\Sem 6\MCN\Practical> java Client
```

Server side code output



```
PS C:\Users\DIGGAJ\Desktop\Diggaj\College\GEC\COMP\Sem 6\MCN\Practical> javac Server.java
PS C:\Users\DIGGAJ\Desktop\Diggaj\College\GEC\COMP\Sem 6\MCN\Practical> java Server
Server is running and waiting for client connection...
Client connected: /127.0.0.1
Received message from client: Hello, server! , I am Diggaj Ugvekar
```

CONCLUSION:

The inter-process communication in Java using Socket Programming was studied, implemented and verified successfully.