**Theory Document: Socket Programming in Computer Networks**

**1. Introduction to Socket Programming**

Socket programming enables communication between two systems (client and server) over a network. A **socket** acts as an endpoint for sending and receiving data.

**2. What is a Socket?**

A **socket** is a software structure that allows a program to connect to another program on a network.

Each socket is bound to:

* An **IP Address**
* A **Port Number**

**Analogy:** A socket is like a telephone jack that allows two devices to connect and communicate.

**3. Types of Sockets**

| **Type** | **Description** | **Protocol** |
| --- | --- | --- |
| **Stream Sockets** | Provide reliable, two-way, connection-based communication. | **TCP** |
| **Datagram Sockets** | Provide fast, unreliable, connectionless communication. | **UDP** |

**4. Client-Server Model**

In most network communications, one program acts as the **server** and waits for connections, while another acts as the **client** and initiates the connection.

**Server Responsibilities:**

* Wait for client requests
* Process request
* Send response

**Client Responsibilities:**

* Initiate connection
* Send request
* Receive response

**5. Socket API Functions**

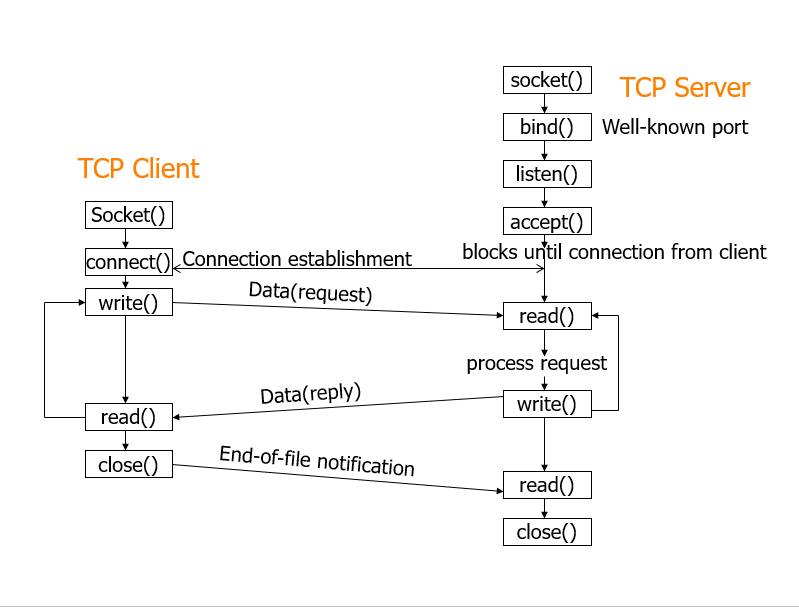
**For TCP (Stream Socket)**

**Server Side (C/POSIX):**

1. socket() – Create socket
2. bind() – Bind socket to IP and port
3. listen() – Listen for incoming connections
4. accept() – Accept client connection
5. read() / recv() – Receive data
6. write() / send() – Send data
7. close() – Close the socket

**Client Side (C/POSIX):**

1. socket() – Create socket
2. connect() – Connect to server
3. read() / recv() – Receive data
4. write() / send() – Send data
5. close() – Close the socket



**7. Socket Programming in Java (TCP Example)**

**Server.java**

**import java.io.\*;**

**import java.net.\*;**

**public class Server {**

**public static void main(String[] args) throws IOException {**

**ServerSocket serverSocket = new ServerSocket(8080);**

**System.out.println("Server started, waiting for client...");**

**Socket clientSocket = serverSocket.accept();**

**System.out.println("Client connected!");**

**BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));**

**PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);**

**// Receive message from client**

**String msg = in.readLine();**

**System.out.println("Client: " + msg);**

**// Send reply to client**

**out.println("Hello from Server");**

**clientSocket.close();**

**serverSocket.close();**

**}**

**}**

**Client.java  
import java.io.\*;**

**import java.net.\*;**

**public class Client {**

**public static void main(String[] args) {**

**try {**

**Socket socket = new Socket("localhost", 8080);**

**BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));**

**PrintWriter out = new PrintWriter(socket.getOutputStream(), true);**

**// Send message to server**

**out.println("Hello Server");**

**// Receive server response**

**String response = in.readLine();**

**System.out.println("Server: " + response);**

**socket.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**8. UDP Socket Programming**

In UDP, there's no connection establishment. Messages (datagrams) are sent directly.

**Important Functions:**

* sendto()
* recvfrom()

UDP is faster but does not guarantee delivery.

**Implement a TCP-based chat application.**