**IPC: Sockets**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 9(a)**

**Title-** Write two programs (server/client) and establish a socket to communicate.

**Objectives:**

1. To learn about fundamentals of IPC through C socket programming.
2. Learn and understand the OS interaction with socket programming.
3. Use of system call and IPC mechanism to write effective application programs.
4. To know the port numbering and process relation.
5. To knows the iterative and concurrent server concept.

**Theory:**

A very basic one-way Client and Server setup where a client connects, sends messages to server and the server shows them using socket connection. Java API networking package (java.net) takes care of all of that, making network programming very easy for programmers

CLIENT-SIDE PROGRAMMING:

Establish a Socket Connection

* To connect to other machine, we need a socket connection.
* A socket connection means the two machines have information about each other’s network location (IP Address) and TCP port. The java.net.Socket class represents a Socket.
* To open a socket: Socket socket = new Socket (“127.0.0.1”, 5000)

• First argument – IP address of Server. (127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).

• Second argument – TCP Port. (Just a number representing which

application to run on a server. For example, HTTP runs on port 80.

Port number can be from 0 to 65535) To communicate over a socket

connection, streams are used to both input and output the data. Closing

the connection. The socket connection is closed explicitly once the

message to server is sent.

SERVER-SIDE PROGRAMMING:

Establish a Socket Connection

To write a server application two sockets are needed.

* A ServerSocket which waits for the client requests (when a client makes a new Socket())
* A plain old Socket socket to use for communication with the client getOutputStream() method is used to send the output through the socket. Close the Connection After finishing, it is important to close the connection by closing the socket as well as input/output streams

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| **SR. NO.** | **Variable/Function** | **Data Type** | **Use** |
| 1 | ss | ServerSocket | Create a socket for server side  communication. |
| 2 | s | Socket | Socket is created |
| 3 | dos | DataOutputStream | Output Stream |
| 4 | dis | DataInputStream | Input Stream |
| 5 | str | String | String to display message from  clients. |

**Program:  
Server:**

import java.io.\*;

import java.net.\*;

public class server {

    public static void main(String[] args) {

        try {

            // Create a server socket that listens on port 12345

            ServerSocket serverSocket = new ServerSocket(12345);

            System.out.println("Server is listening on port 12345...");

            // Wait for a connection from the client

            Socket socket = serverSocket.accept();

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

            // Get input and output streams for communication

            BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

            // Read the message from the client

            String clientMessage = input.readLine();

            System.out.println("Received from client: " + clientMessage);

            // Send a response back to the client

            String response = "Hello, Client! Message received: " + clientMessage;

            output.println(response);

            // Close the connection

            socket.close();

            serverSocket.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

**Client:**

import java.io.\*;

import java.net.\*;

public class client {

    public static void main(String[] args) {

        try {

            // Connect to the server on localhost (127.0.0.1) at port 12345

            Socket socket = new Socket("localhost", 12345);

            System.out.println("Connected to the server!");

            // Get input and output streams for communication

            BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

            // Get the response from the server

            BufferedReader serverInput = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            // Send a message to the server

            System.out.print("Enter a message for the server: ");

            String message = input.readLine();

            output.println(message);

            // Read and display the server's response

            String serverMessage = serverInput.readLine();

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

            // Close the connection

            socket.close();

        } catch (IOException e) {

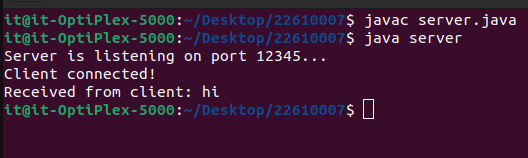
            e.printStackTrace();

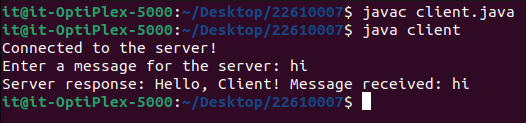
        }

    }

}

**Output:**

****

****

**Conclusion:**

Java can be used to establish communication between two programs on remote or same machine using sockets and system calls.