### IPC: Sockets

Subject:- Unix Operating System System Lab Class :- TYIT

**Name: - SHRENIK JADHAV**

**OM GHARGE**

**PRN: -2020BTEIT00011,2020BTEIT00041**

**Assignment No 9a**

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

**Objective:**

1. To learn about fundamentals of IPC through C socket programming.
2. Learn and understand the OS intraction with socket programming.
3. Use of system call and IPC mechanism to write effective application programs.
4. To know the port numbersing 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 | DatOutputStream | 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 and bind it to a port

ServerSocket serverSocket = new ServerSocket(1234);

// Wait for a client to connect

System.out.println("Waiting for a client to connect...");

Socket clientSocket = serverSocket.accept();

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

// Create input and output streams for the socket

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

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

// Read data from the client and print it

String message = in.readLine();

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

// Send a response back to the client

out.println("Message received by the server.");

// Close the streams and sockets

in.close();

out.close();

clientSocket.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 {

// Create a socket and connect to the server

Socket socket = new Socket("localhost", 1234);

// Create input and output streams for the socket

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

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

// Send a message to the server

String message = "Hello, server!";

out.println(message);

System.out.println("Sent to server: " + message);

// Receive and print the response from the server

String response = in.readLine();

System.out.println("Received from server: " + response);

// Close the streams and socket

in.close();

out.close();

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

run-----Compile the Java files using the javac command:

bash

* javac Server.java

javac Client.java

This will generate the corresponding .class files.

* Run the server program in one terminal window by executing the following command:

bash

* java Server

The server will start and wait for a client to connect.

* Open another terminal window and run the client program by executing the following command:

bash

* java Client

The client will connect to the server, send a message, and receive a response.

* You should see the output from the server and client programs in their respective terminal windows.





**Conclusion-**

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

**Reference-**

<http://www.prasannatech.net/2008/07/socket-programming-tutorial.html>