

Date: 06/4/2021

Lab Assignment No: 08

Aim: Socket Programming using TCP.

Lab Outcome Attained: LO 4: -To implement client-server socket programs.

Theory:

What is a socket?

- A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.
- Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data.

What is Socket Programming?

- Socket programming is a way of connecting two nodes on a network to communicate with each other.
- One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection.
- Server forms the listener socket while client reaches out to the server.

How Server Socket Works?

- Typically, a server runs on one computer and accesses a socket that is bound to a specific port.
- The server waits for a different computer to make a connection request.

- The client computer knows the hostname of the server computer and the port number on which the server is listening.
- The client computer identifies itself, and if everything goes right the server permits the client computer to connect.

Executable Codes in Java:

□ Client Side:

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

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

```
class TCPClient
```

```
{ public static void main(String argv[]) throws
```

```
Exception
```

```
{
```

```
String sentence;
```

```
String modifiedSentence;
```

```
BufferedReader inFromUser = new BufferedReader( new InputStreamReader(System.in));
```

```
Socket clientSocket = new Socket("localhost", 6789);
```

```
DataOutputStream outToServer = new  
DataOutputStream(clientSocket.getOutputStream());
```

```
BufferedReader inFromServer = new  
BufferedReader(new  
InputStreamReader(clientSocket.getInputStream()));
```

```
sentence = inFromUser.readLine();
```

```
outToServer.writeBytes(sentence + '\n');
```

```
modifiedSentence = inFromServer.readLine();
```

```
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();
}
}
```

□ Server Side:

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

class TCPServer
{
    public static void main(String argv[]) throws
    Exception
    {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true)
        {
            Socket connectionSocket = welcomeSocket.accept();
            inFromClient = new BufferedReader(new
            InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
```

```
OutputStream(connectionSocket.getOutputStream());
clientSentence = inFromClient.readLine();
System.out.println("Received: " + clientSentence);
capitalizedSentence = clientSentence.toUpperCase() + '\n';
outToClient.writeBytes(capitalizedSentence);
    }
}
}
```

Explanation of Code of Client Side:

import java.net.*;

- It includes the ServerSocket class, which implements a socket that servers can use to listen for and accept connections to clients.
- The java.net package provides classes for network support.
- In particular, it contains the Socket and ServerSocket classes.
The clientSocket object of this program is derived from the Socket class.

Socket clientSocket=new Socket("localhost",6789);

- To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.
- Here, we are using 6789 port number for the communication between the client and server. You may also choose any other port number. -

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

- Java BufferedReader class is used to read the text from a characterbased input stream.

- To get the server's response, it reads from the `BufferedReader` object `inFromUser`.
- The above line creates the stream object `inFromUser` of type `BufferedReader`.
- The input stream is initialized with `System.in`, which attaches the stream to the standard input. The command allows the client to read text from its keyboard.

```
DataOutputStream outToServer =new  
DataOutputStream(clientSocket.getOutputStream()); BufferedReader  
inFromServer = new BufferedReader(new  
InputStreamReader(clientSocket.getInputStream()));
```

- It gets the socket's output stream and opens a `DataOutputStream` on it.
- Similarly, the next statement gets the socket's input stream and opens a `BufferedReader` on it.
- To send data through the socket to the server, the Client needs to write to the `DataOutputStream`.
- The above two lines create stream objects that are attached to the socket.
- The `outToServer` stream provides the process output to the socket.
- The `inFromServer` stream provides the process input from the socket.

```
clientSocket.close();
```

- Client closes the connection with the server.
- Server Never gets closed, it always remain open.

```
sentence=inFromUser.readLine();
```

- The above line places a line typed by user into the string `sentence`.
- The string `sentence` continues to gather characters until the user ends the line by typing a carriage return.
- The line passes from standard input through the stream `inFromUser` into the string `sentence`.
- The `readline()` method reads one line from the file and returns it as a string.

- String returned by readline will contain newline character at the end. 🚩
outToServer.writeBytes(sentence + '\n');
- The above line sends the string sentence augmented with a carriage return into the outToServer stream.
- The augmented sentence flows through the client's socket and into the TCP pipe. The client then waits to receive characters from the server.
- “\n” is the ‘separator’ between the messages.
- ‘\n’ is considered to be terminated by any one of a line feed.
-

Explanation of Code of Server Side:

ServerSocket listener=new ServerSocket(6789);

- To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6789 port number for the communication between the client and server. 🚩 Socket
connectionSocket=listener.accept(); - The accept() method waits for the client.
- If clients connect with the given port number, it returns an instance of Socket.
- It invokes the ServerSocket's accept() method to listen on the configured port for a client connection.
- When a client connects to the server, the accept() method returns a Socket through which the server can communicate with the client.

BufferedReader inFromClient =new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream())); DataOutputStream
outToClient =new DataOutputStream(connectionSocket.getOutputStream());

- Create Input Stream attached to the socket – Create Output Stream attached to the Socket.

- Client reads line from standard input (inFromClient stream), sends to server via socket (outToClient stream) - Server reads line from socket.

```
"C:\Users\VRUDDHI PARAG TOLIA\.jdk\openjdk-16.0.1\bin\java.exe" "C:\Users\VRUDDHI PARAG TOLIA\.jdk\openjdk-16.0.1\bin\java.exe"
Vruddhi
Received: Vruddhi
|
Vruddhi
FROM SERVER: VRUDDHI
Process finished with exit code 0
|
```

CONCLUSION: Thus, the concept of Socket Programming (Client – Server Socket Programs) using CP was understood and the implementation for the same was done during the lab.