EXPERIMENT NO. 9

Socket Programming

Aim: i. Write a program to create the stream socket to implement Client-Server model.
ii. Perform the communication using TCP socket.

System Software Requirements: JDK, Windows-10 / Ubuntu.

Theory:

The process-to-process communication service provided by transport layer is achieved most commonly the client-server paradigm. A process on the local host, called a client, needs services from a process usually on the remote host, called a server. The operating systems today support both multiuser and multiprogramming environments. A remote computer can run several server programs at the same time, just as several local computers can run one or more client programs at the same time. For communication, define the local host, local process, remote host, and remote process. The local host and the remote host are defined using IP addresses and the processes by second identifiers, called port numbers. A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection. A socket is supposed to behave like a terminal or a file, it is an abstraction. It is an object that is created and used by the application program. The communication between a client process and a server process is communication between two sockets, created at two ends, as shown in Fig. The client thinks that the socket is the entity that receives the request and gives the response; the server thinks that the socket is the one that has a request and needs the response. The combination of an IP address and a port number is called a socket address. The client socket address defines the client process uniquely and the server socket address defines the server process uniquely.

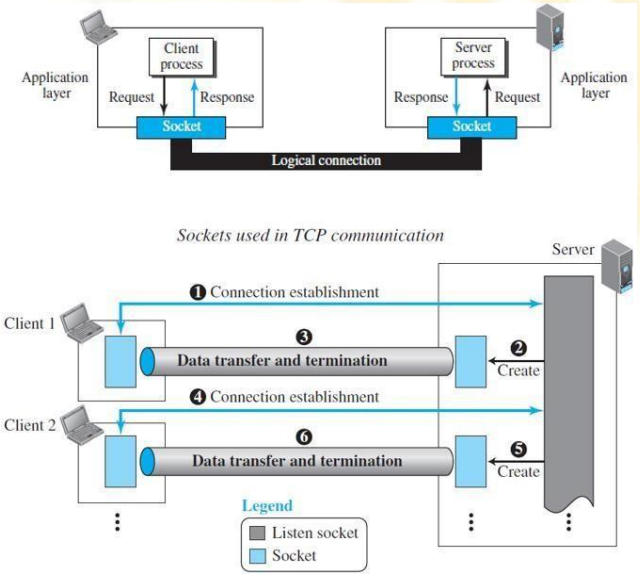


Fig. Sockets in TCP

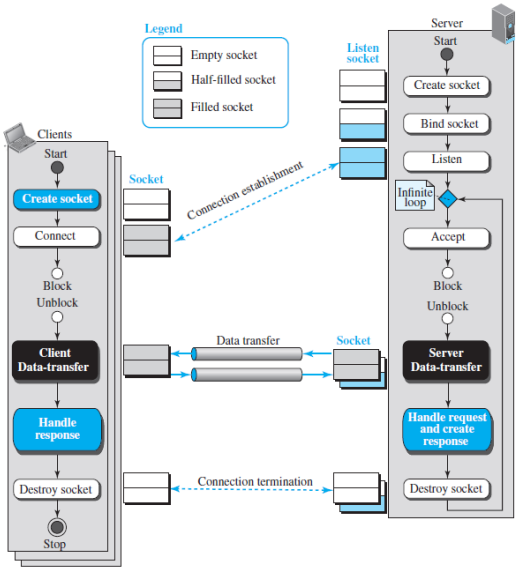


Fig. Client-Server model using Stream Socket

Transmission Control Protocol (TCP):
TCP allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their bytes across the Internet. This imaginary environment is depicted in Figure. The sending process produces (writes to) the stream and the receiving process consumes (reads from) it. TCP provides connection-oriented, reliable, byte-stream service. TCP requires that two ends first create a logical connection between themselves by exchanging some connection-establishment packets. This phase, which is sometimes called handshaking, establishes some parameters between the two ends, including the size of the data packets to be exchanged, the size of buffers to be used for holding the chunks of data until the whole message arrives, and so on. After the handshaking process, the two ends can send chunks of data in segments in each direction. By numbering the bytes exchanged, the continuity of the bytes can be checked. For example, if some bytes are lost or corrupted, the receiver can request the resending of those bytes, which makes TCP a reliable protocol.

Sockets Used in TCP:
The TCP server uses two different sockets, one for connection establishment and the other for data transfer. The first one is called as listen socket and the second the socket. The reason for having two types of sockets is to separate the connection phase from the data exchange phase. A server uses a listen socket to listen for a new client trying to establish connection. After the connection is established, the server creates a socket to exchange data with the client and finally to terminate the connection. The client uses only one socket for both connection establishment and data exchange.

Server Process:
The server makes a passive open, in which it becomes ready for the communication, but it waits until a client process makes the connection. It creates a socket and binds it, but these two commands create the listen socket to be used only for the connection establishment phase. The server process then calls the listen procedure, to allow the operating system to start accepting the clients, completing the connection phase, and putting them in the waiting list to be served. The server process now starts a loop and serves the clients one by one. In each iteration, the server process issues the accept procedure that removes one client from the waiting list of the connected clients for serving. If the list is empty, the accept procedure blocks until there is a client to be served. When the accept procedure returns, it creates a new socket for data transfer. The server process now uses the client socket address obtained during the connection establishment to fill the remote socket address field in the newly created socket. At this time the client and server can exchange data.

Client Process:
The client process makes an active open. In other words, it starts a connection. It creates an empty socket and then issues the send command, which fully fills the socket, and sends the request. The client then issues a receive command, which is blocked until a response arrives from the server. The response is then handled and the socket is destroyed. The client data-transfer box needs to be defined for each specific case.

Name of the Student: Vaishnavi A patil
Roll No.: 122A1115
Class: TE-CE
Batch: D2

EXPERIMENT NO. 9

Socket Programming

Aim:
i. Write a program to create the stream socket to implement Client-Server model.
ii. Perform the communication using TCP socket.

System Software Requirements: JDK, Windows-10 / Ubuntu

Program:

SERVER SIDE PROGRAM-

```
import java.net.*;
import java.io.*;
class MyServer
{
public static void main(String args[])throws Exception
{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop"))
{
str=din.readUTF();
System.out.println("client says: "+str);

str2=br.readLine();
dout.writeUTF(str2); dout.flush();
}
din.close();
```

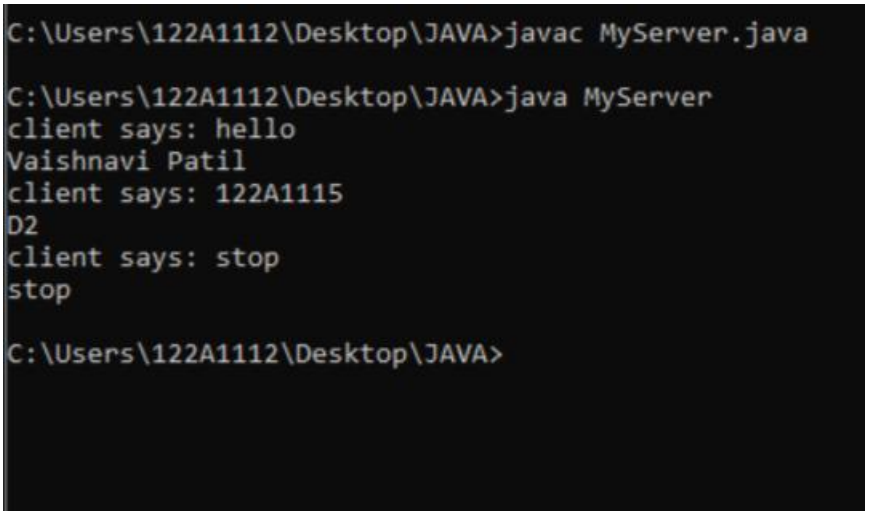
```
s.close();
ss.close();
}
}
```

CLIENT SIDE PROGRAM-

```
import java.net.*;
import java.io.*;
class MyClient
{
public static void main(String args[])throws Exception
{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2=""; while(!str.equals("stop"))
{
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}
}
```

Output:

```
C:\Users\122A1112\Desktop\JAVA>javac MyClient.java

C:\Users\122A1112\Desktop\JAVA>java MyClient
hello
Server says: Vaishnavi Patil
122A1115
Server says: D2
stop
Server says: stop

C:\Users\122A1112\Desktop\JAVA>
```



Conclusion:

The implementation of the TCP-based client-server communication was successfully completed. The server was able to establish a reliable connection with the client, exchange data, and terminate the connection gracefully. This demonstrates how TCP provides a stable and reliable transport layer for real-time data communication in networked applications.