**EXPERIMENT NO: 5**

***AIM:***To study TCP client server communication.

***THEORY:***

The **Transmission Control Protocol** (**TCP**) is one of the main [protocols](https://en.wikipedia.org/wiki/Communications_protocol) of the [Internet protocol suite](https://en.wikipedia.org/wiki/Internet_protocol_suite). It originated in the initial network implementation in which it complemented the [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP). Therefore, the entire suite is commonly referred to as *TCP/IP*. TCP provides [reliable](https://en.wikipedia.org/wiki/Reliability_(computer_networking)), ordered, and [error-checked](https://en.wikipedia.org/wiki/Error_detection_and_correction) delivery of a stream of [octets](https://en.wikipedia.org/wiki/Octet_(computing)) between applications running on hosts communicating by an IP network. Major Internet applications such as the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), [email](https://en.wikipedia.org/wiki/Email), [remote administration](https://en.wikipedia.org/wiki/Remote_administration), and [file transfer](https://en.wikipedia.org/wiki/File_transfer) rely on TCP. Applications that do not require reliable data stream service may use the [User Datagram Protocol](https://en.wikipedia.org/wiki/User_Datagram_Protocol) (UDP), which provides a [connectionless](https://en.wikipedia.org/wiki/Connectionless_communication) [datagram](https://en.wikipedia.org/wiki/Datagram) service that emphasizes reduced [latency](https://en.wikipedia.org/wiki/Latency_(engineering)) over reliability.

***NETWORK FUNCTION:***

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the Transport Layer of the Internet model. An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required packet fragmentation on the transmission medium. At the transport layer, the protocol handles all handshaking and transmission details and presents an abstraction of the network connection to the application. At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets may be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, the source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details.

***PROTOCOL OPERATION:***

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (*connection establishment*) before entering the *data transfer* phase. After data transmission is completed, the *connection termination* closes established virtual circuits and releases all allocated resources. A TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the *Internet socket*. During the lifetime of a TCP connection the local end-point undergoes a series of state changes:

**LISTEN :**(server) represents waiting for a connection request from any remote TCP and port.

**SYN-SENT:** (client) represents waiting for a matching connection request after having sent a connection request.

**SYN-RECEIVED:** (server) represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

**ESTABLISHED:** (both server and client) represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

**FIN-WAIT-1:** (both server and client) represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

**FIN-WAIT-2:** (both server and client) represents waiting for a connection termination request from the remote TCP.

**CLOSE-WAIT:** (both server and client) represents waiting for a connection termination request from the local user.

**CLOSING:** (both server and client) represents waiting for a connection termination request acknowledgment from the remote TCP.

**LAST-ACK:** (both server and client) represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

**TIME-WAIT:** (either server or client) represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.

**CLOSED:** (both server and client) represents no connection state at all.

**TCP SERVER:**

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();

BufferedReader inFromClient =

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

clientSentence = inFromClient.readLine();

System.out.println("Received: " + clientSentence);

capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence); }}}

**TCP CLIENT:**

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();

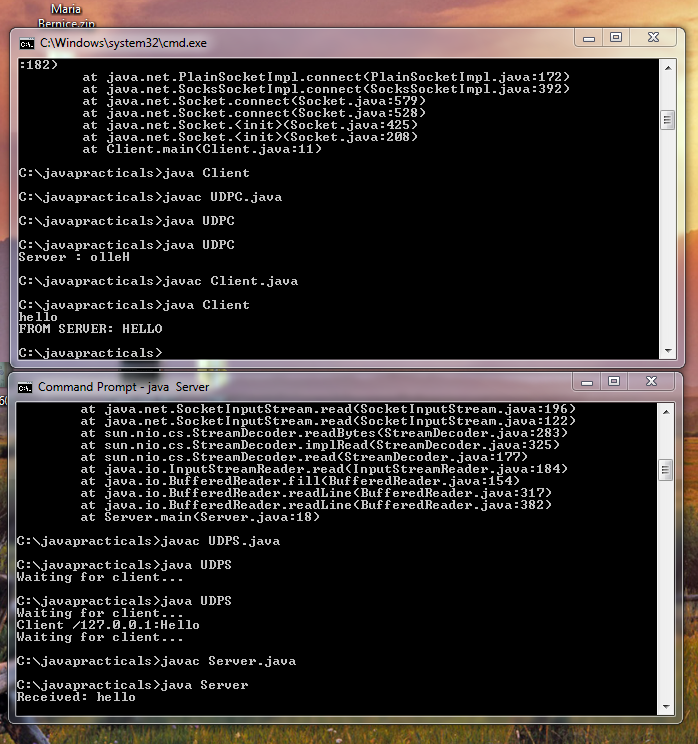
outToServer.writeBytes(sentence + '\n');

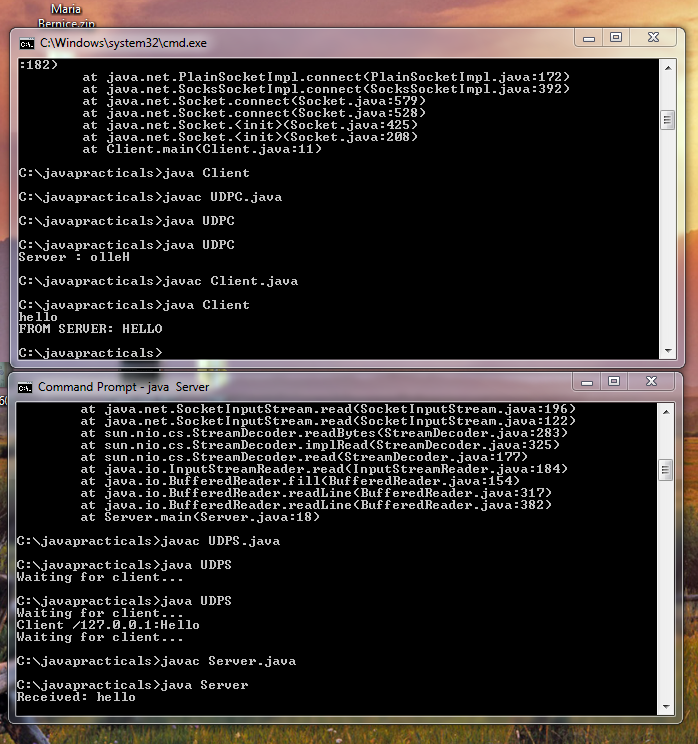
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();}}

**OUTPUT:**

****

****

***CONCLUSION:***

Hence we have studied that the Transmission Control Protocol (TCP) binding establishes a two-way connection between a server and a single client. The “Client” in a TCP/IP connection is the computer or device that “dials the phone” and the “Server” is the computer that is “listening” for calls to come in. In other words, the Client needs to know the IP Address of whatever Server it wants to connect to and it also needs to know the port number that it wants to send and receive data through after a connection has been established. The Server only has to listen for connections and either accept them or reject them when they are initiated by a client. Once a connection through a TCP/IP port has been established between a TCP/IP client and a TCP/IP server, data can be sent in either direction exactly the same way that data is sent through any other type of port on a PC (serial, parallel, etc.).