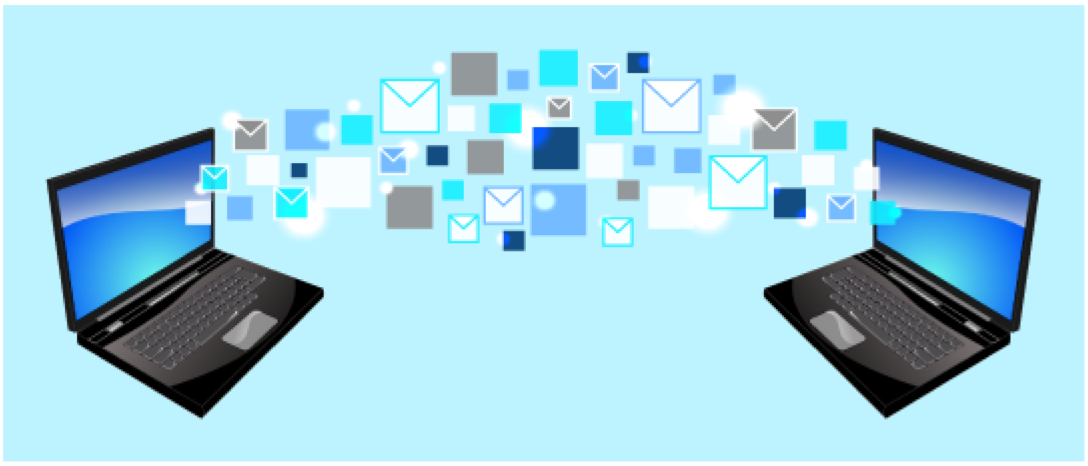
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Ramapuram Campus, Chennai- 600089.
FACULTY OF ENGINEERING AND TECHNOLOGY
Department of Computer Science & Engineering

18CSC302J / COMPUTER NETWORKS

LAB MANUAL



CLASS	**:**	**B.Tech. [U.G]**
YEAR	**:**	**III YEAR**
SEM.	**:**	**V**
SOFTWARE REQUIREMENT	**:**	**JAVA any Versions**

Prepared By

Dr. M.Azhagiri AP/CSE

Mrs. R.Sathya AP/CSE

Mr. S.P. Maniraj AP/CSE

LIST OF EXPERIMENTS

1. Study of necessary header files with respect to socket programming.
2. Study of Basic Functions of Socket Programming
3. Simple TCP/IP Client Server Communication
4. UDP Echo Client Server Communication
5. Concurrent TCP/IP Day-Time Server
6. Half Duplex Chat Using TCP/IP
7. Full Duplex Chat Using TCP/IP
8. Implementation of File Transfer Protocol
9. Remote Command Execution Using UDP
10. ARP implementation Using UDP
11. Study of IPV6 Addressing & Sub netting
12. Implementation of NAT
13. Implementation of VPN
14. Communication Using HDLC
15. Communication Using PPP

Ex.No: 1	**Study of necessary header files with respect to socket programming**
Date:	

Aim:

To Study of necessary header files with respect to socket programming

Description:

1. stdio.h:

Has standard input and output library providing simple and efficient buffered stream IO interface.

2. unistd.h:

It is a POSIX standard for open system interface. [Portable Operating System Interface

3. string.h:

This header file is used to perform string manipulation operations on NULL terminated strings.(Bzero -0 the m/y)

4. stdlib.h:

This header file contains the utility functions such as string conversion routines, memory allocation routines, random number generator, etc.

5. sys/types.h:

Defines the data type of socket address structure in unsigned long.

6. sys/socket.h:

The socket functions can be defined as taking pointers to the generic socket address structure

called sockaddr.

7. netinet/in.h:

Defines the IPv4 socket address structure commonly called Internet socket address structure called sockaddr_in.

8. netdb.h:

Defines the structure hostent for using the system call gethostbyname to get the network host entry.

9. time.h:

Has structures and functions to get the system date and time and to perform time manipulation functions. We use the function ctime(), that is defined in this header file , to calculate the current date and time.

10. sys/stat.h:

Contains the structure stat to test a descriptor to see if it is of a specified type. Also it is used to display file or file system status.stat() updates any time related fields.when copying from 1 file to

another.

11. sys/ioctl.h:

Macros and defines used in specifying an ioctl request are located in this header file. We use the function ioctl() that is defined in this header file. ioctl() function is used to perform ARP cache operations.

12. pcap.h:

Has function definitions that are required for packet capturing. Some of the functions are pcap_lookupdev(),pcap_open_live() and pcap_loop(). pcap_lookupdev() is used to initialize the network device.The device to be sniffed is opened using the pcap_open_live(). Pcap_loop() determines the number of packets to be sniffed.

13. net/if_arp.h:

Contains the definitions for Address Resolution Protocol. We use this to manipulate the ARP request structure and its data members arp_pa,arp_dev and arp_ha. The arp_ha structure's data member sa_data[ ] has the hardware address.

14. errno.h:

It sets an error number when an error and that error can be displayed using perror function. It has symbolic error names. The error number is never set to zero by any library function.

15. arpa/inet.h:

This is used to convert internet addresses between ASCII strings and network byte ordered binary values (values that are stored in socket address structures). It is used for inet_aton, inet_addr, inet_ntoa functions

Result: Thus the Study of necessary header files with respect to socket programming has been done.

Ex.No: 2	Study of Basic Functions of Socket Programming
Date:	

Aim:

To discuss some of the basic functions used for socket programming.

1.**man socket**

NAME:
Socket – create an endpoint for communication.

SYNOPSIS:
#include<sys/types.h>
#include<sys/socket.h>
int socket(int domain,inttype,int protocol);

DESCRIPTION:
➢ Socket creates an endpoint for communication and returns a descriptor.
➢ The domain parameter specifies a common domain this selects the protocol family
which will be used for communication.
➢ These families are defined in <sys/socket.h>.

FORMAT:

NAME	PURPOSE
PF_UNIX,PF_LOCAL	Local Communication.
PF_INET	IPV4 Internet Protocols.
PF_IPX	IPX-Novell Protocols.
PF_APPLETALK	Apple Talk.

➢ The socket has the indicated type, which specifies the communication semantics.

TYPES:
1.SOCK_STREAM:
➢ Provides sequenced , reliable, two-way , connection based byte streams.
➢ An out-of-band data transmission mechanism, may be supported.Page | 4
2.SOCK_DGRAM:
➢ Supports datagram (connectionless, unreliable messages of a fixed maximum length).
3.SOCK_SEQPACKET:
➢ Provides a sequenced , reliable, two-way connection based data transmission path for
datagrams of fixed maximum length.
4.SOCK_RAW:
➢ Provides raw network protocol access.

5.SOCK_RDM:
➢ Provides a reliable datagram layer that doesn't guarantee ordering.
6.SOCK_PACKET:
➢ Obsolete and shouldn't be used in new programs.

2.man connect:

NAME:
connect – initiate a connection on a socket.

SYNOPSIS:
#include<sys/types.h>
#include<sys/socket.h>
int connect(int sockfd,const (struct sockaddr*)serv_addr,socklen_taddrlen);

DESCRIPTION:
➢ The file descriptor sockfd must refer to a socket.
➢ If the socket is of type SOCK_DGRAM then the serv_addr address is the address to
which datagrams are sent by default and the only addr from which datagrams are
received.
➢ If the socket is of type SOCK_STREAM or SOCK_SEQPACKET , this call attempts
to make a connection to another socket.
RETURN VALUE:
➢ If the connection or binding succeeds, zero is returned.
➢ On error , -1 is returned , and error number is set appropriately.

ERRORS:

EBADF	Not a valid Index.
EFAULT	The socket structure address is outside the user's address space.
ENOTSOCK	Not associated with a socket.
EISCONN	Socket is already connected.
ECONNREFUSED	No one listening on the remote address.

3.man accept
NAME:
accept/reject job is sent to a destination.
SYNOPSIS:
accept destination(s)
reject[-t] [-h server] [-r reason] destination(s)
DESCRIPTION:

➢ accept instructs the printing system to accept print jobs to the specified destination.
➢ The –r option sets the reason for rejecting print jobs.
➢ The –e option forces encryption when connecting to the server.

4. man send

NAME:
send, sendto, sendmsg - send a message from a socket.
SYNOPSIS:
#include<sys/types.h>
#include<sys/socket.h>
ssize_tsend(int s, const void *buf, size_tlen, int flags);
ssize_tsendto(int s, const void *buf, size_tlen, int flags, const struct sock_addr*to, socklen_ttolen);
ssize_tsendmsg(int s, const struct msghdr *msg, int flags);

DESCRIPTION:
➢ The system calls send, sendto and sendmsg are used to transmit a message to another
socket.
➢ The send call may be used only when the socket is in a connected state.
➢ The only difference between send and write is the presence of flags.
➢ The parameter is the file descriptor of the sending socket.

5.man recv
NAME:
recv, recvfrom, recvmsg – receive a message from a socket.
SYNOPSIS:
#include<sys/types.h>
#include<sys/socket.h>
ssize_trecv(int s, void *buf, size_tlen, int flags);
ssize_trecvfrom(int s, void *buf, size_tlen, int flags, struct sockaddr *from, socklen_t* from len);
ssize_trecvmsg(int s, struct msghdr *msg, int flags);
DESCRIPTION:
➢ The recvfrom and recvmsg calls are used to receive messages from a socket, and may
be used to recv data on a socket whether or not it is connection oriented.
➢ If from is not NULL, and the underlying protocol provides the srcaddr , this srcaddr is
filled in.
➢ The recv call is normally used only on a connection socket and is identical to recvfrom
with a NULL from parameter.

6.man read

NAME:
read, readonly, return

7.man write
NAME:
write- send a message to another user.
SYNOPSIS:
write user[ttyname]

DESCRIPTION:
➢ write allows you to communicate with other users, by copying lines from terminal to
....
➢ When you run the write and the user you are writing to get a message of the form:
Message from yourname @yourhost on yourtty at hh:mm:...
➢ Any further lines you enter will be copied to the specified user's terminal.
➢ If the other user wants to reply they must run write as well.

8. ifconfig
NAME:
ifconfig- configure a network interface.
SYNOPSIS:
Page | 6 ifconfig[interface]
ifconfig interface[aftype] options | address......
DESCRIPTION:
➢ ifconfig is used to configure the kernel resident network interfaces.
➢ It is used at boot time to setup interfaces as necessary.
➢ After that, it is usually only needed when debugging or when system tuning is needed.
➢ If no arguments are given, ifconfig displays the status of the currently active interfaces.

9. man bind
SYNOPSIS:
bind[-m keymap] [-lpsvpsv]

10. man htons/ man htonl
NAME:
htonl, htons, ntohl, ntohs- convert values between host and network byte order.
SYNOPSIS:
#include<netinet/in.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint32_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);

DESCRIPTION:

➢ The htonl() function converts the unsigned integer hostlong from host byte
order to
network byte order.
➢ The htons() converts the unsigned short integer hostshort from host byte
order to network
byte order.
➢ The ntohl() converts the unsigned integer netlong from network byte order to
host byte
order.

11. man gethostname
NAME:
gethostname, sethostname- get/set host name.
SYNOPSIS:
#include<unistd.h>
Page | 7 int gethostname(char *name,size_tlen);
int sethostname(const char *name,size_tlen);

DESCRIPTION:
➢ These functions are used to access or to change the host name of the current
processor.
➢ The gethostname() returns a NULL terminated hostname(set earlier by
sethostname()) in
the array name that has a length of len bytes.
➢ In case the NULL terminated then hostname does not fit ,no error is returned,
but the
hostname is truncated.
➢ It is unspecified whether the truncated hostname will be NULL terminated.

12. man gethostbyname
NAME:
gethostbyname, gethostbyaddr, sethostent, endhostent, herror, hstr – error – get
network host entry.

SYNOPSIS:
#include<netdb.h>
extern int h_errno;
struct hostent *gethostbyname(const char *name);
#include<sys/socket.h>
struct hostent *gethostbyaddr(const char *addr)int len, int type);
struct hostent *gethostbyname2(const char *name,intaf);

DESCRIPTION:
➢ The gethostbyname() returns a structure of type hostent for the given
hostname.
➢ Name->hostname or IPV4/IPV6 with dot notation.
➢ gethostbyaddr()- struct of type hostent / host address length
➢ Address types- AF_INET, AF_INET6.

➢ sethostent() – stay open is true(1).
➢ TCP socket connection should be open during queries.
➢ Server queries for UDP datagrams.
➢ endhostent()- ends the use of TCP connection.
➢ Members of hostent structure:
a) h_name
b) h_aliases
c) h_addrtype
d) h_length
e) h_addr-list
f) h_addr.

RESULT: Thus the basic functions used for Socket Programming was studied successfully.

Ex.No: 3	**Simple TCP/IP Client Server Communication**
Date:	

Aim: To establish communication between the Client and the Server to exchange messages

Algorithm

1. An object of ***ServerSocket*** is instantiated, and desired port number is specified, on which connection is going to take place.
2. The ***accept*** method of ***ServerSocket*** is invoked, in order to hold the server in listening mode. This method won't resume until a client is connected to the server through the given port number.
3. Now, on client side, an object of ***Socket*** is instantiated, and desired port number and IP address is specified for the connection.
4. An attempt is made, for connecting the client to the server using the specified IP address and port number. If attempt is successful, client is provided with a ***Socket*** that is capable of communicating to the respective server, with write and read methods. If unsuccessful, the desired exception is raised.
5. Since a client is connected to the server, ***accept*** method on the server side resumes, providing a ***Socket*** that is capable of communicating to the connected client.
6. Once the communication is completed, terminate the sockets on both, the server and the client side.

Server Program (MyServer.java)

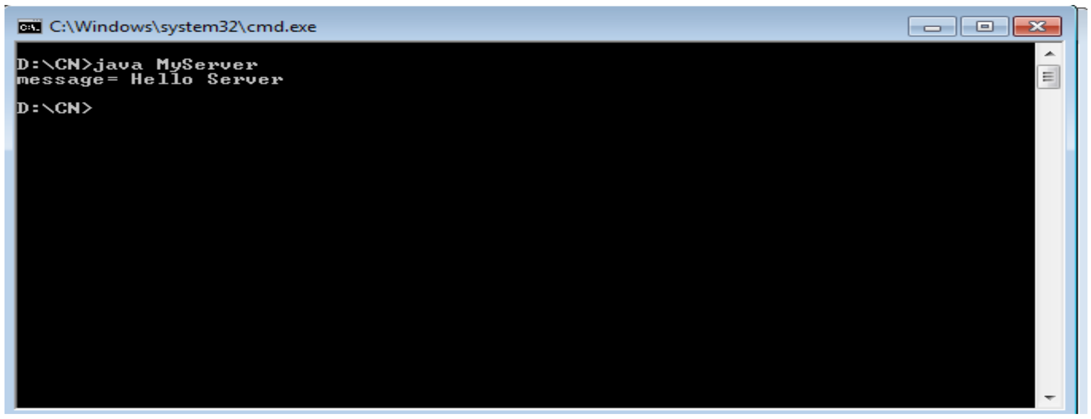
```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Client Program (MyClient.java)

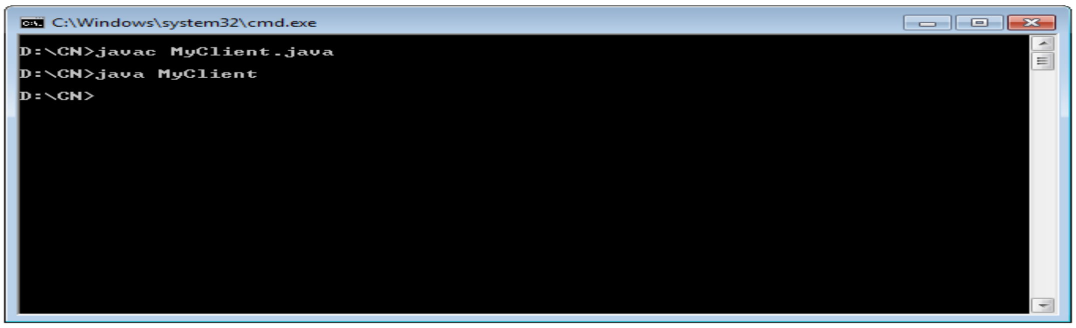
```
import java.io.*;
import java.net.*;
```

```
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Output



```
D:\CN>java MyServer
message= Hello Server

D:\CN>
```



```
D:\CN>javac MyClient.java

D:\CN>java MyClient

D:\CN>
```

RESULT: Thus the communication between the Client and the Server to exchange messages has been established

Ex.No: 4	**UDP Echo Client Server Communication**
Date:	

AIM:

To implement an Echo server and client in java using UDP sockets.

DESCRIPTION:

UDP is a connectionless protocol and the socket is created for client and server to transfer the data. Socket connection is achieved using the port number. Domain Name System is the naming convention that divides the Internet into logical domains identified in Internet Protocol version 4 (IPv4) as a 32-bit portion of the total address.

ALGORITHM:

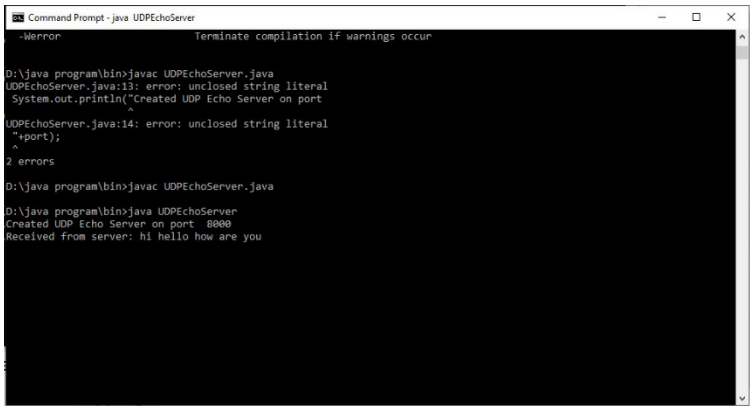
Server

1. Create two ports, server port and client port.
2. Create a datagram socket and bind it to the client port.
3. Create a datagram packet to receive client messages.
4. Wait for the client's data and accept it.
5. Read Client's message.
6. Get data from users.
7. Create a datagram packet and send a message through the server port.
8. Repeat steps 3-7 until the client has something to send.
9. Close the server socket.
10. Stop.

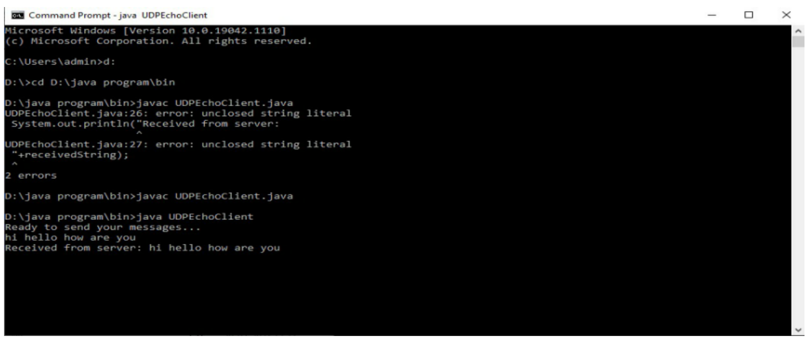
Client

1. Create two ports, server port and client port.
2. Create a datagram socket and bind it to the server port.
3. Get data from users.
4. Create a datagram packet and send data with server ip address and client port.
5. Create a datagram packet to receive server messages.
6. Read the server's response and display it.
7. Repeat steps 3-6 until there is some text to send.
8. Close the client socket.
9. Stop.

Output



```
  -Werror                      Terminate compilation if warnings occur


D:\java program\bin>javac UDPEchoServer.java
UDPEchoServer.java:13: error: unclosed string literal
 System.out.println("Created UDP Echo Server on port
                    ^
UDPEchoServer.java:14: error: unclosed string literal
 "+port);
 ^
2 errors

D:\java program\bin>javac UDPEchoServer.java

D:\java program\bin>java UDPEchoServer
Created UDP Echo Server on port  8000
Received from server: hi hello how are you
```



```
Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\admin>d:

D:\>cd D:\java program\bin

D:\java program\bin>javac UDPEchoClient.java
UDPEchoClient.java:26: error: unclosed string literal
 System.out.println("Received from server:
                    ^
UDPEchoClient.java:27: error: unclosed string literal
 "+receivedString);
 ^
2 errors

D:\java program\bin>javac UDPEchoClient.java

D:\java program\bin>java UDPEchoClient
Ready to send your messages...
hi hello how are you
Received from server: hi hello how are you
```

RESULT: Thus UDP Echo Client Server Communication has been implemented

Ex.No: 5	**Concurrent TCP/IP Day-Time Server**
Date:	

AIM: To implement date and time display from client to server using TCP Sockets

DESCRIPTION: TCP Server gets the system date and time and opens the server socket to read the client details. Client sends its address to the server. Then the client receives the date and time from server to display. TCP socket server client connection is opened for communication. After the date time is displayed the server client connection is closed with its respective streams to be closed.

ALGORITHM:

Server

1.Create a server socket and bind it to the port.

2.Listen for new connections and when a connection arrives, accept it.

3. Send server's date and time to the client.

4. Read the client's IP address sent by the client.

5. Display the client details.

6. Repeat steps 2-5 until the server is terminated.

7. Close all streams.

8. Close the server socket.

9. Stop.

Client

1.Create a client socket and connect it to the server's port number.

2. Retrieve its own IP address using built-in function.

3.Send its address to the server.

4. Display the date & time sent by the server.

5. Close the input and output streams.

6. Close the client socket.

7. Stop.

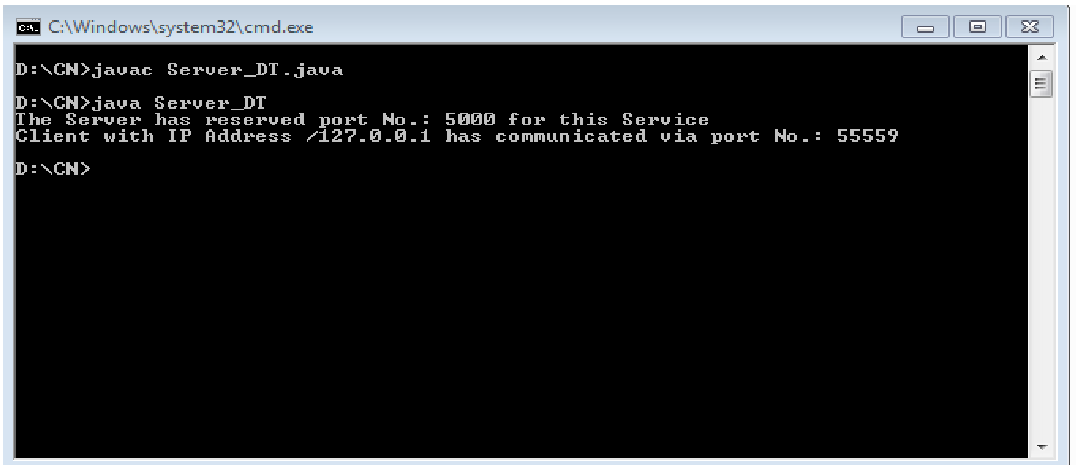
Program

Server_DT

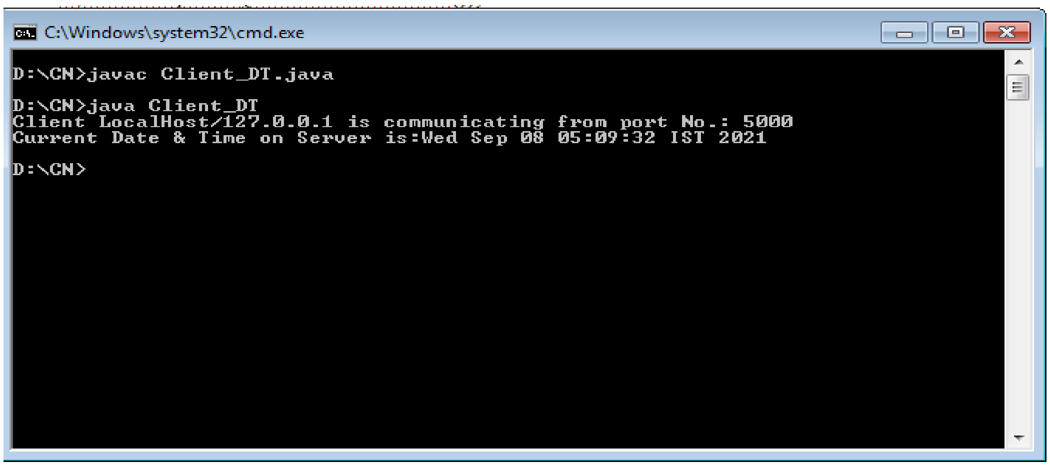
```
import java.net.*;
import java.io.*;
import java.util.Date;

public class Server_DT {


    public static void main(String[] args)throws IOException {
        // TODO code application logic here

        //Step 1. Reserve a port number on the Server to offer this service
        ServerSocket ss= new ServerSocket(5000);
        //(Optional)To confirm Server Reserved specified port or not
        System.out.println("The Server has reserved port No.:
"+ss.getLocalPort()+" for this Service");

        //Step 2. Now create a Client Socket on Server for Bidirectonal
Communication.
        //Socket is created only when client communicates with the server
        Socket cs=ss.accept();

        //To confirm Server communicated through the socket or not

        System.out.println("Client with IP Address "+cs.getInetAddress()+" has
communicated via port No.: "+cs.getPort());

        Date d=new Date();
        String s="Current Date & Time on Server is:"+d;

        //Send String s to client via client socket
        PrintWriter toclient=new PrintWriter(cs.getOutputStream(),true);
        toclient.print(s);

        toclient.close();
        cs.close();
        ss.close();
    }
}
```

Client_DT

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

public class Client_DT {

    public static void main(String[] args) throws
UnknownHostException,IOException {
        // TODO code application logic here
        //Step 1. Create a client socket to connect to Server
        Socket cs= new Socket("LocalHost",5000);

        //To confirm Client is communicating through the port
        System.out.println("Client "+cs.getInetAddress()+" is communicating
from port No.: "+cs.getPort());

        //Receive Date Sent by Server
        BufferedReader fromserver=new BufferedReader(new
InputStreamReader(cs.getInputStream()));

        System.out.println(fromserver.readLine());
        fromserver.close();
        cs.close();

    }
}
```
Output



```
D:\CN>javac Server_DT.java

D:\CN>java Server_DT
The Server has reserved port No.: 5000 for this Service
Client with IP Address /127.0.0.1 has communicated via port No.: 55559

D:\CN>
```


C:\Windows\system32\cmd.exe

```
D:\CN>javac Client_DT.java

D:\CN>java Client_DT
Client LocalHost/127.0.0.1 is communicating from port No.: 5000
Current Date & Time on Server is:Wed Sep 08 05:09:32 IST 2021

D:\CN>
```

Result: Thus the concurrent TCP/IP Day-Time Server has been implemented

Ex.No: 6	**Half Duplex Chat Using TCP/IP**
Date:	

AIM: To implement a chat server and client in java using TCP sockets in half duplex mode.

DESCRIPTION:

TCP Clients send requests to the server and the server will receive the request and response with acknowledgement. Every time either a client or a server can send and receive the messages

ALGORITHM:

Server

1. Create a server socket and bind it to the port.
2. Listen for new connections and when a connection arrives, accept it.
3. Read Client's message and display it
4. Get a message from user and send it to client
5. Repeat steps 3-4 until the client terminates
6. Close all streams
7. Close the server and client socket
8. Stop

Client

1. Create a client socket and connect it to the server's port number
2. Get a message from user and send it to server
3. Read server's response and display it
4. Repeat steps 2-3 until chat is terminated with "exit" message
5. Close all input/output streams

6. Close the client socket

7. Stop

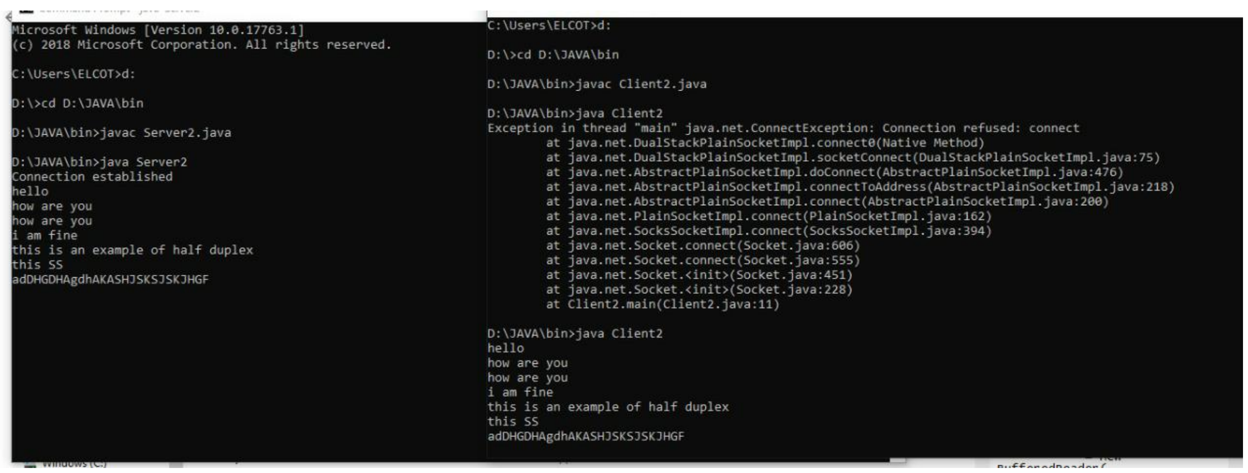
Server

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

class Server2 {

    public static void main(String args[])
        throws Exception
    {

        // Create server Socket
        ServerSocket ss = new ServerSocket(888);

        // connect it to client socket
        Socket s = ss.accept();
        System.out.println("Connection established");

        // to send data to the client
        PrintStream ps
            = new PrintStream(s.getOutputStream());

        // to read data coming from the client
        BufferedReader br
            = new BufferedReader(
                new InputStreamReader(
                    s.getInputStream()));

        // to read data from the keyboard
        BufferedReader kb
            = new BufferedReader(
                new InputStreamReader(System.in));

        // server executes continuously
        while (true) {

            String str, str1;

            // repeat as long as the client
            // does not send a null string

            // read from client
            while ((str = br.readLine()) != null) {
                System.out.println(str);
                str1 = kb.readLine();
                // send to client
                ps.println(str1);
            }

            // close connection
            ps.close();
            br.close();
            kb.close();
            ss.close();
            s.close();

            // terminate application
            System.exit(0);

        } // end of while
    }
}
```

Client

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

class Client2 {

    public static void main(String args[])
        throws Exception
    {

        // Create client socket
        Socket s = new Socket("localhost", 888);

        // to send data to the server
        DataOutputStream dos
            = new DataOutputStream(
                s.getOutputStream());

        // to read data coming from the server
        BufferedReader br
            = new BufferedReader(
                new InputStreamReader(
                    s.getInputStream()));

        // to read data from the keyboard
        BufferedReader kb
            = new BufferedReader(
                new InputStreamReader(System.in));
        String str, str1;
        // repeat as long as exit
        // is not typed at client
        while (!(str = kb.readLine()).equals("exit")) {

            // send to the server
            dos.writeBytes(str + "\n");

            // receive from the server
            str1 = br.readLine();

            System.out.println(str1);
        }

        // close connection.
        dos.close();
        br.close();
        kb.close();
        s.close();
    }
}
```

Output



```
Microsoft Windows [Version 10.0.17763.1]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\ELCOT>d:

D:\>cd D:\JAVA\bin

D:\JAVA\bin>javac Server2.java

D:\JAVA\bin>java Server2
Connection established
hello
how are you
how are you
i am fine
this is an example of half duplex
this SS
adDHGDHAgdhAKASHJSKSJSKJHGF
```

```
C:\Users\ELCOT>d:

D:\>cd D:\JAVA\bin

D:\JAVA\bin>javac Client2.java

D:\JAVA\bin>java Client2
Exception in thread "main" java.net.ConnectException: Connection refused: connect
        at java.net.DualStackPlainSocketImpl.connect0(Native Method)
        at java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.java:75)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:476)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:218)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:200)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:162)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:394)
        at java.net.Socket.connect(Socket.java:606)
        at java.net.Socket.connect(Socket.java:555)
        at java.net.Socket.<init>(Socket.java:451)
        at java.net.Socket.<init>(Socket.java:228)
        at Client2.main(Client2.java:11)

D:\JAVA\bin>java Client2
hello
how are you
how are you
i am fine
this is an example of half duplex
this SS
adDHGDHAgdhAKASHJSKSJSKJHGF
```

Result: Thus Half Duplex Chat Using TCP/IP has been executed using Java programming

Ex.No: 7	**Full Duplex Chat Using TCP/IP**
Date:	

AIM:

To implement a chat server and client in java using TCP sockets.

DESCRIPTION:

TCP Clients send requests to the server and the server will receive the request and response with acknowledgement. Every time the client communicates with the server and receives a response from it.

ALGORITHM:

Server

1. Create a server socket and bind it to the port.
2. Listen for new connections and when a connection arrives, accept it.
3. Read Client's message and display it
4. Get a message from user and send it to client
5. Repeat steps 3-4 until the client sends "exit"
6. Close all streams
7. Close the server and client socket
8. Stop

Client

1. Create a client socket and connect it to the server's port number

3. Get a message from user and send it to server

5.Read server's response and display it

6.Repeat steps 2-3 until chat is terminated with "exit" message

5. Close all input/output streams

6. Close the client socket

7. Stop

Server

```
package com;

import java.io.IOException;
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class Server{
	static ServerSocket serverSocket;
	public static void main(String[] args) {
		try {
			serverSocket = new ServerSocket(1515);
			while(true) {
				Socket accept_client = serverSocket.accept();
				new Thread(new ServerIn(accept_client)).start();
				new Thread(new ServerOut(accept_client)).start();
			}
		} catch (IOException e) {
			e.printStackTrace();
			try {
				serverSocket.close();
			} catch (IOException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
		}
	}
}
//Accepted threads
class ServerIn implements Runnable{
	Socket socket;

	ServerIn(Socket socket){
		this.socket = socket;
	}
	@Override
	public void run() {
		try {
			InputStream in = socket.getInputStream();
			while(true) {
				byte infile[] = new byte[1024];
```

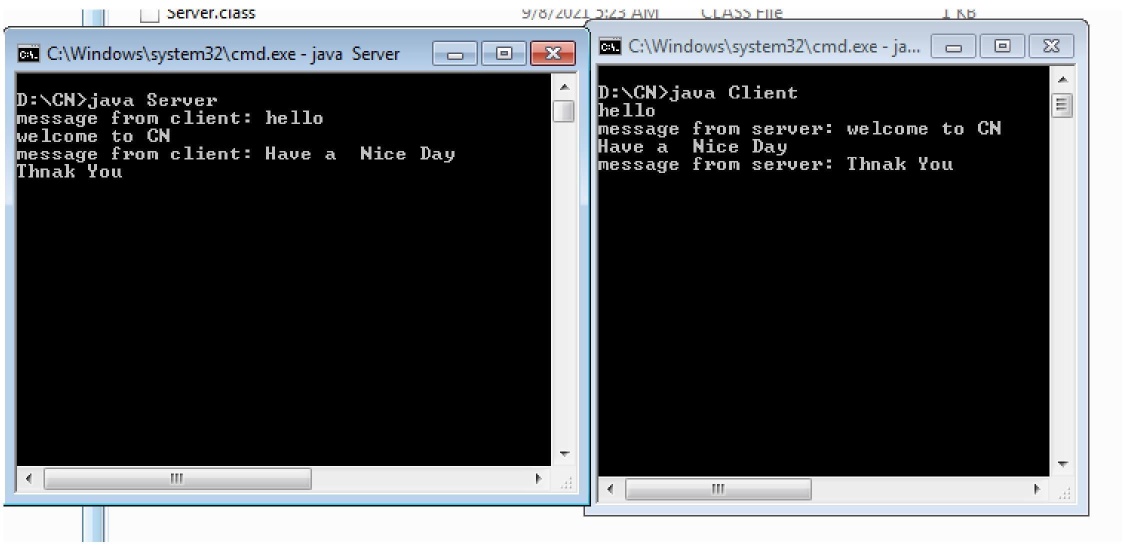
```
                    int size  = in.read(infile);
                    String string = new String(infile,0,size);
                    if(!string.equals("") && !string.equals("\n"))
System.out.println("message from client: "+ string);
                }
            } catch (IOException e) {
                e.printStackTrace();
                try {
                    socket.close();
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        }
}
//Thread to send
class ServerOut implements Runnable{
        Socket socket;
        Scanner reader = new Scanner(System.in);
        ServerOut(Socket socket){
            this.socket = socket;
        }
        public void run() {
            try {
                OutputStreamWriter out = new
OutputStreamWriter(socket.getOutputStream());
                while(true) {
                    String string  = reader.nextLine();
                    out.write(string);
                    out.flush();
                }
            } catch (IOException e) {
                e.printStackTrace();
                try {
                    socket.close();
                    reader.close();
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        }
}
```

Client

```
package com;
```

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

public class Client{
        public static void main(String[] args) {
                try {
                        Socket client = new Socket("127.0.0.1",1515);
                        new Thread(new ClientIn(client)).start();
                        new Thread(new ClientOut(client)).start();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
//Accepted threads
class ClientIn implements Runnable{
        Socket socket;
        ClientIn(Socket socket){
                this.socket = socket;
        }
        @Override
        public void run() {
                try {
                        InputStream in = socket.getInputStream();
                        while(true) {
                                byte infile[] = new byte[1024];
                                int size = in.read(infile);
                                String string = new String(infile,0,size);
                                if(!string.equals("") && !string.equals("\n"))
System.out.println("message from server: "+ string);
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                        try {
                                socket.close();
                        } catch (IOException e1) {
                                e1.printStackTrace();
                        }
                }
        }

}
//Thread to send
class ClientOut implements Runnable{
        Socket socket;
        Scanner reader = new Scanner(System.in);
        ClientOut(Socket socket){
                this.socket = socket;
        }
        public void run() {
                try {
                        OutputStreamWriter out = new
OutputStreamWriter(socket.getOutputStream());
                        while(true) {
                                String string  = reader.nextLine();
                                out.write(string);
                                out.flush();
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                        try {
                                socket.close();
                                reader.close();
                        } catch (IOException e1) {
                                e1.printStackTrace();
                        }
                }
        }
}
```

Output



C:\Windows\system32\cmd.exe - java Server
C:\Windows\system32\cmd.exe - ja...

```
D:\CN>java Server
message from client: hello
welcome to CN
message from client: Have a  Nice Day
Thnak You
```

```
D:\CN>java Client
hello
message from server: welcome to CN
Have a  Nice Day
message from server: Thnak You
```

Result: Thus Half Duplex Chat Using TCP/IP has been executed using Java programming

Ex.No: 8	**Implementation of File Transfer Protocol**
Date:	

Aim: To write a program in java to implement file transfer between client and file server.

Algorithm:
1. Create a socket connection from client to the server with corresponding IP address and port number
2. Create necessary file streams
3. Get the option from the user to SEND/ DISCONNECT and the file name to be Transferred
4. If SEND, read each byte from the file in the client side and write it into the output stream of the socket
5. In the server side, read the bytes from the input stream and write it into the file

Server

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

public class FTPServer
{
   public static void main(String args[]) throws Exception
   {
      ServerSocket soc=new ServerSocket(5217);
      System.out.println("FTP Server Started on Port Number 5217");
      while(true)
      {
         System.out.println("Waiting for Connection ...");
         transferfile t=new transferfile(soc.accept());

      }
   }
}

class transferfile extends Thread
{
   Socket ClientSoc;

   DataInputStream din;
   DataOutputStream dout;

   transferfile(Socket soc)
   {
      try
```

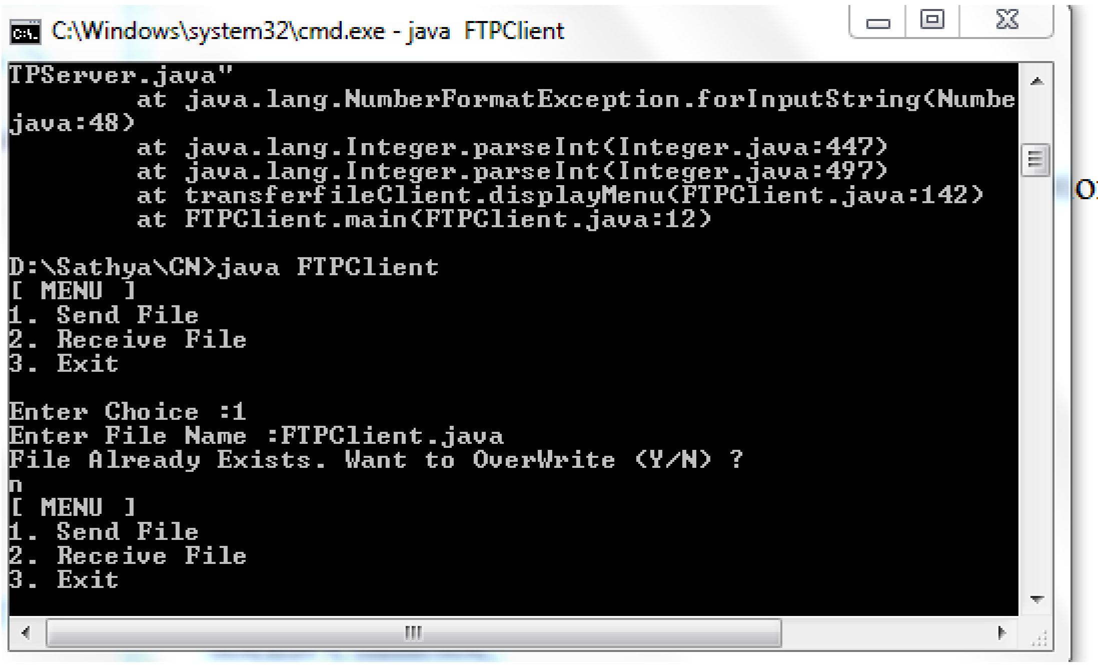
```
    {
        ClientSoc=soc;
        din=new DataInputStream(ClientSoc.getInputStream());
        dout=new DataOutputStream(ClientSoc.getOutputStream());
        System.out.println("FTP Client Connected ...");
        start();

    }
    catch(Exception ex)
    {
    }
}
void SendFile() throws Exception
{
    String filename=din.readUTF();
    File f=new File(filename);
    if(!f.exists())
    {
        dout.writeUTF("File Not Found");
        return;
    }
    else
    {
        dout.writeUTF("READY");
        FileInputStream fin=new FileInputStream(f);
        int ch;
        do
        {
            ch=fin.read();
            dout.writeUTF(String.valueOf(ch));
        }
        while(ch!=-1);
        fin.close();
        dout.writeUTF("File Receive Successfully");
    }
}

void ReceiveFile() throws Exception
{
    String filename=din.readUTF();
    if(filename.compareTo("File not found")==0)
    {
        return;
    }
    File f=new File(filename);
    String option;

    if(f.exists())
    {
        dout.writeUTF("File Already Exists");
        option=din.readUTF();
    }
    else
    {
        dout.writeUTF("SendFile");
        option="Y";
    }

        if(option.compareTo("Y")==0)
        {
            FileOutputStream fout=new FileOutputStream(f);
            int ch;
            String temp;
            do
            {
                temp=din.readUTF();
                ch=Integer.parseInt(temp);
                if(ch!=-1)
                {
                    fout.write(ch);
                }
            }while(ch!=-1);
            fout.close();
            dout.writeUTF("File Send Successfully");
        }
        else
        {
            return;
        }

}


public void run()
{
    while(true)
    {
        try
        {
        System.out.println("Waiting for Command ...");
        String Command=din.readUTF();
        if(Command.compareTo("GET")==0)
        {
            System.out.println("\tGET Command Received ...");
            SendFile();
            continue;
        }
        else if(Command.compareTo("SEND")==0)
```

```
            {
               System.out.println("\tSEND Command Received ...");
               ReceiveFile();
               continue;
            }
            else if(Command.compareTo("DISCONNECT")==0)
            {
               System.out.println("\tDisconnect Command Received ...");
               System.exit(1);
            }
            }
            catch(Exception ex)
            {
            }
         }
      }
   }


import java.net.*;
import java.io.*;
import java.util.*;


class FTPClient
{
   public static void main(String args[]) throws Exception
   {
      Socket soc=new Socket("127.0.0.1",5217);
      transferfileClient t=new transferfileClient(soc);
      t.displayMenu();

   }
}
class transferfileClient
{
   Socket ClientSoc;

   DataInputStream din;
   DataOutputStream dout;
   BufferedReader br;
   transferfileClient(Socket soc)
   {
      try
      {
         ClientSoc=soc;
         din=new DataInputStream(ClientSoc.getInputStream());
         dout=new DataOutputStream(ClientSoc.getOutputStream());
         br=new BufferedReader(new InputStreamReader(System.in));
```

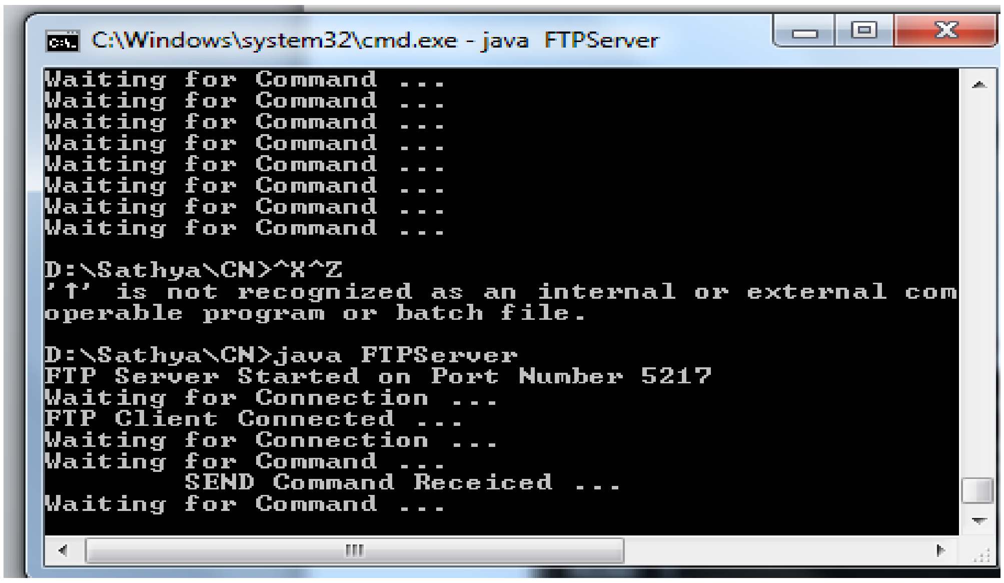
```
    }
    catch(Exception ex)
    {
    }
}
void SendFile() throws Exception
{

    String filename;
    System.out.print("Enter File Name :");
    filename=br.readLine();

    File f=new File(filename);
    if(!f.exists())
    {
        System.out.println("File not Exists...");
        dout.writeUTF("File not found");
        return;
    }

    dout.writeUTF(filename);

    String msgFromServer=din.readUTF();
    if(msgFromServer.compareTo("File Already Exists")==0)
    {
        String Option;
        System.out.println("File Already Exists. Want to OverWrite (Y/N) ?");
        Option=br.readLine();
        if(Option=="Y")
        {
            dout.writeUTF("Y");
        }
        else
        {
            dout.writeUTF("N");
            return;
        }
    }

    System.out.println("Sending File ...");
    FileInputStream fin=new FileInputStream(f);
    int ch;
    do
    {
        ch=fin.read();
        dout.writeUTF(String.valueOf(ch));
    }
    while(ch!=-1);
    fin.close();
```

```
        System.out.println(din.readUTF());

    }

    void ReceiveFile() throws Exception
    {
        String fileName;
        System.out.print("Enter File Name :");
        fileName=br.readLine();
        dout.writeUTF(fileName);
        String msgFromServer=din.readUTF();

        if(msgFromServer.compareTo("File Not Found")==0)
        {
            System.out.println("File not found on Server ...");
            return;
        }
        else if(msgFromServer.compareTo("READY")==0)
        {
            System.out.println("Receiving File ...");
            File f=new File(fileName);
            if(f.exists())
            {
                String Option;
                System.out.println("File Already Exists. Want to OverWrite (Y/N)
?");
                Option=br.readLine();
                if(Option=="N")
                {
                    dout.flush();
                    return;
                }
            }
            FileOutputStream fout=new FileOutputStream(f);
            int ch;
            String temp;
            do
            {
                temp=din.readUTF();
                ch=Integer.parseInt(temp);
                if(ch!=-1)
                {
                    fout.write(ch);
                }
            }while(ch!=-1);
            fout.close();
            System.out.println(din.readUTF());

        }
    }

    public void displayMenu() throws Exception
    {
        while(true)
        {
            System.out.println("[ MENU ]");
            System.out.println("1. Send File");
            System.out.println("2. Receive File");
            System.out.println("3. Exit");
            System.out.print("\nEnter Choice :");
            int choice;
            choice=Integer.parseInt(br.readLine());
            if(choice==1)
            {
                dout.writeUTF("SEND");
                SendFile();
            }
            else if(choice==2)
            {
                dout.writeUTF("GET");
                ReceiveFile();
            }
            else
            {
                dout.writeUTF("DISCONNECT");
                System.exit(1);
            }
        }
    }
}
```

OUTPUT



```
TPServer.java"
        at java.lang.NumberFormatException.forInputString(Numbe
java:48)
        at java.lang.Integer.parseInt(Integer.java:447)
        at java.lang.Integer.parseInt(Integer.java:497)
        at transferfileClient.displayMenu(FTPClient.java:142)
        at FTPClient.main(FTPClient.java:12)

D:\Sathya\CN>java FTPClient
[ MENU ]
1. Send File
2. Receive File
3. Exit

Enter Choice :1
Enter File Name :FTPClient.java
File Already Exists. Want to OverWrite (Y/N) ?
n
[ MENU ]
1. Send File
2. Receive File
3. Exit
```



```
Waiting for Command ...
Waiting for Command ...
Waiting for Command ...
Waiting for Command ...
Waiting for Command ...
Waiting for Command ...
Waiting for Command ...
Waiting for Command ...

D:\Sathya\CN>^X^Z
'↑' is not recognized as an internal or external com
operable program or batch file.

D:\Sathya\CN>java FTPServer
FTP Server Started on Port Number 5217
Waiting for Connection ...
FTP Client Connected ...
Waiting for Connection ...
Waiting for Command ...
        SEND Command Receiced ...
Waiting for Command ...
```

RESULT:

Thus FTP using TCP has been implemented using JAVA Socket Programming

Ex.No: 9	**Remote Command Execution Using UDP**
Date:	

Aim: To execute commands remotely using UDP

Algorithm

Client-Side Programming

Open the socket connection
Communication: In the communication part, there is a slight change. The difference with the previous article lies in the usage of both the input and output streams to send commands and receive the results to and from the server respectively. DataInputStream and DataOutputStream are used instead of basic InputStream and OutputStream to make it machine independent.

Server-Side Programming

Steps involved on the server side are as follows-

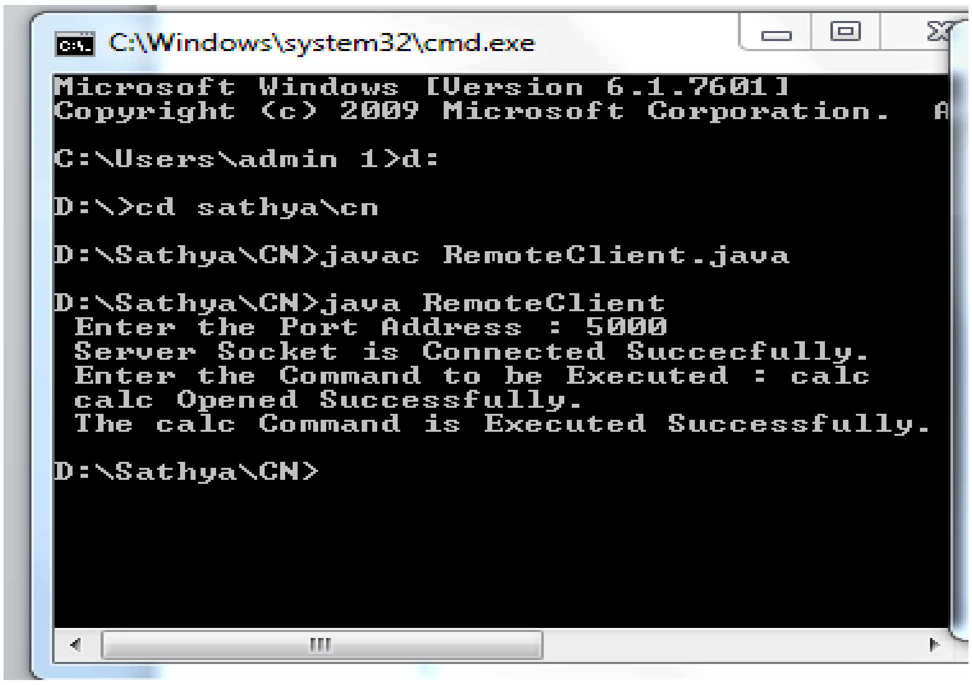
Establish a socket connection.
Process the equations coming from client: In server side also we open both the inputStream and outputStream. After receiving the equation, we process it and return the result back to the client by writing on the outputStream of the socket.
Close the connection.

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

class RemoteClient
{
public static void main(String args[])
{
try
{
int Port;
BufferedReader Buf =new BufferedReader(new
InputStreamReader(System.in));
System.out.print(" Enter the Port Address : " );
Port=Integer.parseInt(Buf.readLine());
Socket s=new Socket("localhost",Port);
if(s.isConnected()==true)
        System.out.println(" Server Socket is Connected Successfully. ");
InputStream in=s.getInputStream();
OutputStream ou=s.getOutputStream();
BufferedReader buf=new BufferedReader(new
InputStreamReader(System.in));
BufferedReader buf1=new BufferedReader(new
InputStreamReader(in));
PrintWriter pr=new PrintWriter(ou);
System.out.print(" Enter the Command to be Executed : " );
pr.println(buf.readLine());
pr.flush();
String str=buf1.readLine();
System.out.println(" " + str + " Opened Successfully. ");
System.out.println(" The " + str + " Command is Executed Successfully. ");
pr.close();
ou.close();
in.close();
}
catch(Exception e)
{
System.out.println(" Error : " + e.getMessage());
}
}
}
```

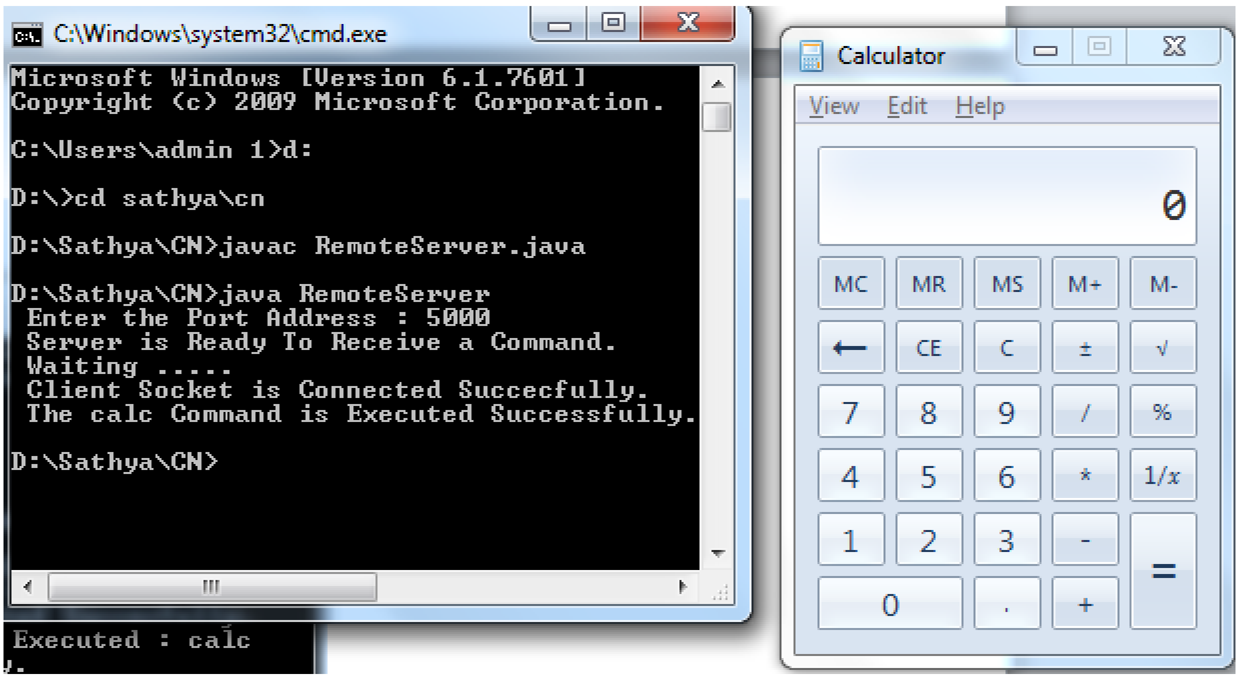
Server

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

class RemoteServer
{
public static void main(String args[])
{
try
{
int Port;
BufferedReader Buf =new BufferedReader(new
InputStreamReader(System.in));
System.out.print(" Enter the Port Address : " );
Port=Integer.parseInt(Buf.readLine());
ServerSocket ss=new ServerSocket(Port);
System.out.println(" Server is Ready To Receive a Command. ");
System.out.println(" Waiting ..... ");
Socket s=ss.accept();
if(s.isConnected()==true)
        System.out.println(" Client Socket is Connected Successfully. ");
InputStream in=s.getInputStream();
OutputStream ou=s.getOutputStream();
BufferedReader buf=new BufferedReader(new
InputStreamReader(in));
String command=buf.readLine();
PrintWriter pr=new PrintWriter(ou);
pr.println(command);
Runtime H=Runtime.getRuntime();
Process P=H.exec(command);
System.out.println(" The " + command + " Command is Executed
Successfully. ");
pr.flush();
pr.close();
ou.close();
in.close();
}
catch(Exception e)
{
System.out.println(" Error : " + e.getMessage());
}
}
}
```

OUTPUT



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  A

C:\Users\admin 1>d:

D:\>cd sathya\cn

D:\Sathya\CN>javac RemoteClient.java

D:\Sathya\CN>java RemoteClient
 Enter the Port Address : 5000
 Server Socket is Connected Succecfully.
 Enter the Command to be Executed : calc
 calc Opened Successfully.
 The calc Command is Executed Successfully.

D:\Sathya\CN>
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\admin 1>d:

D:\>cd sathya\cn

D:\Sathya\CN>javac RemoteServer.java

D:\Sathya\CN>java RemoteServer
 Enter the Port Address : 5000
 Server is Ready To Receive a Command.
 Waiting ......
 Client Socket is Connected Succecfully.
 The calc Command is Executed Successfully.

D:\Sathya\CN>
```

```
Executed : calc
```

RESULT:

Thus RPC using UDP has been implemented using JAVA Socket Programming

Ex.No: 10	**ARP protocols using UDP**
Date:	

Aim:
To write a java program for simulating ARP protocols using TCP

ALGORITHM:
Client
1. Start the program
2. Using socket connection is established between client and server.
3. Get the IP address to be converted into a MAC address.
4. Send this IP address to the server.
5. Server returns the MAC address to the client.

Server
1. Start the program
2. Accept the socket which is created by the client.
3. Server maintains the table in which IP and corresponding MAC addresses are stored.
4. Read the IP address which is sent by the client.
5. Map the IP address with its MAC address and return the MAC address to the client.

Program
Server Program

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverarp12
{
        public static void main(String args[])
        {
        try
        {
                DatagramSocket server=new DatagramSocket(1309);
                while(true)
                {
                        byte[] sendbyte=new byte[1024];
                        byte[] receivebyte=new byte[1024];
                        DatagramPacket receiver=new
DatagramPacket(receivebyte,receivebyte.length);
                        server.receive(receiver);
                        String str=new String(receiver.getData());
                        String s=str.trim();
                        //System.out.println(s);
                        InetAddress addr=receiver.getAddress();
```

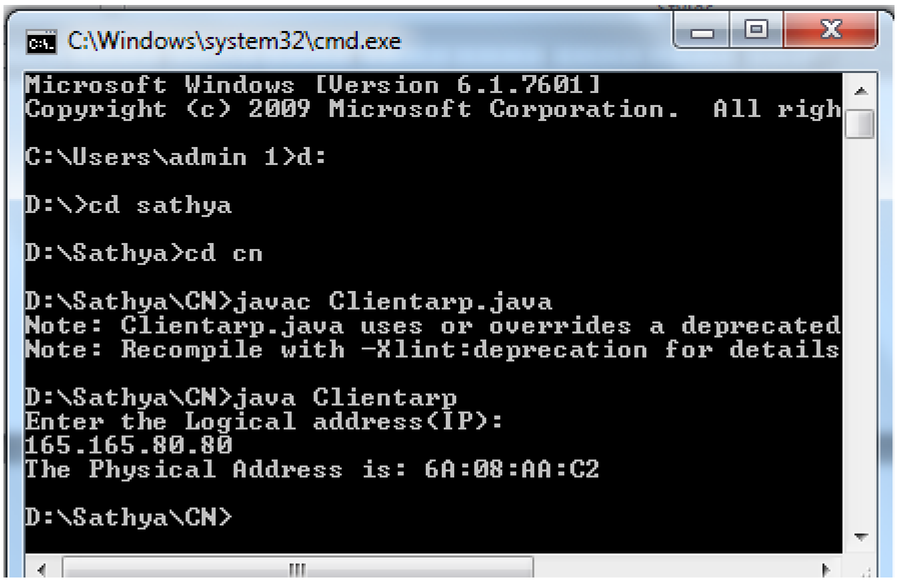
```
                              int port=receiver.getPort();
                              String ip[]={"165.165.80.80","165.165.79.1"};
                              String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
                              for(int i=0;i<ip.length;i++)
                              {
                                        if(s.equals(ip[i]))
                                        {
                                                  sendbyte=mac[i].getBytes();
                                                  DatagramPacket sender=new
DatagramPacket(sendbyte,sendbyte.length,addr,port);
                                                  server.send(sender);
                                                  break;
                                        }
                              }
                              break;


                    }
          }
          catch(Exception e)
          {
                    System.out.println(e);
          }
          }
}
```

Client Program

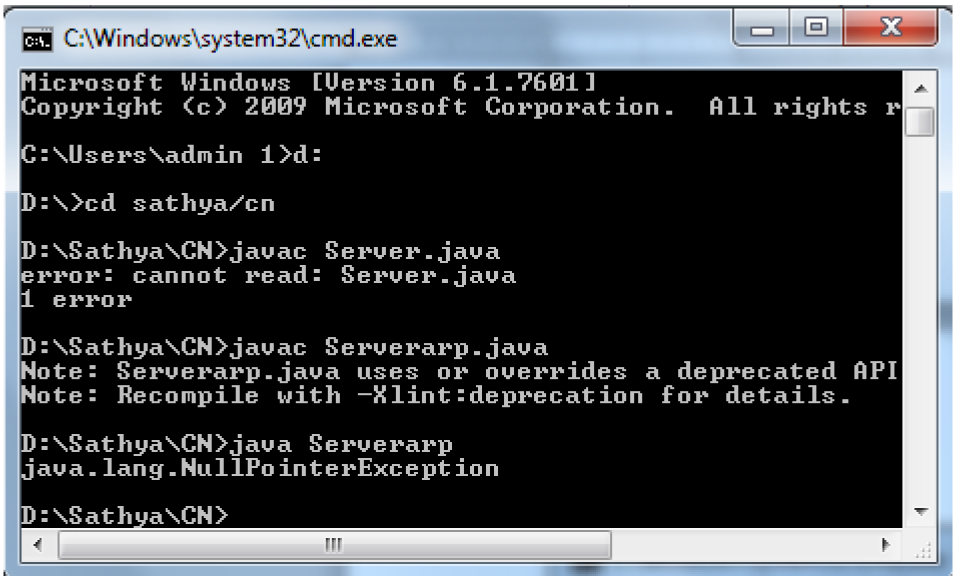
```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp12
{
          public static void main(String args[])
          {
          try
          {
                    DatagramSocket client=new DatagramSocket();
                    InetAddress addr=InetAddress.getByName("127.0.0.1");

                    byte[] sendbyte=new byte[1024];
                    byte[] receivebyte=new byte[1024];
                    BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
                    System.out.println("Enter the logical address (IP):");
                    String str=in.readLine();
                    sendbyte=str.getBytes();
```

```
                DatagramPacket sender=new
DatagramPacket(sendbyte,sendbyte.length,addr,1309);
                client.send(sender);
                DatagramPacket receiver=new
DatagramPacket(receivebyte,receivebyte.length);
                client.receive(receiver);
                String s=new String(receiver.getData());
                System.out.println("The Physical Address is: "+s.trim());
                client.close();
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
        }
}
```

Output:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All righ

C:\Users\admin 1>d:

D:\>cd sathya

D:\Sathya>cd cn

D:\Sathya\CN>javac Clientarp.java
Note: Clientarp.java uses or overrides a deprecated
Note: Recompile with -Xlint:deprecation for details

D:\Sathya\CN>java Clientarp
Enter the Logical address(IP):
165.165.80.80
The Physical Address is: 6A:08:AA:C2

D:\Sathya\CN>
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights r

C:\Users\admin 1>d:

D:\>cd sathya/cn

D:\Sathya\CN>javac Server.java
error: cannot read: Server.java
1 error

D:\Sathya\CN>javac Serverarp.java
Note: Serverarp.java uses or overrides a deprecated API
Note: Recompile with -Xlint:deprecation for details.

D:\Sathya\CN>java Serverarp
java.lang.NullPointerException

D:\Sathya\CN>
```

RESULT: Thus ARP using UDP has been implemented using JAVA Socket Programming

Ex No: 11	**STUDY OF IPV6 ADDRESSING AND SUBNETTING**
Date:	

AIM:
To study IPV6 address terminology, IPV6 address format, Types of Addresses, assigning IPaddresses to devices and subnetting.

PROCEDURE:

IPv6 Address Terminology

Node
Any device that runs an implementation of IPv6. This includes routers and hosts.

Router
A node that can forward IPv6 packets not explicitly addressed to itself. On an IPv6 network, a router also typically advertises its presence and host configuration information.

Host
A node that cannot forward IPv6 packets not explicitly addressed to itself (a non-router). A host is typically the source and a destination of IPv6 traffic, and it silently discards traffic received that is not explicitly addressed to itself.

Upper-layer protocol
A protocol above IPv6 that uses IPv6 as its transport. Examples include Internet layer protocols such as ICMPv6 and Transport layer protocols such as TCP and UDP (but not Application layer protocols such as FTP and DNS, which use TCP and UDP as their transport).

Link
The set of network interfaces that are bounded by routers and that use the same 64-bit IPv6unicast address prefix. Other terms for “link” are subnet and network segment.

Network
Two or more subnets connected by routers. Another term for networks is internetworks.

Neighbors
Nodes connected to the same link. Neighbors in IPv6 have special significance because of IPv6Neighbor Discovery, which has facilities to resolve neighbor link layer addresses and detect and monitor neighbor reach ability

Interface
The representation of a physical or logical attachment of a node to a link. An example of a physical interface is a network adapter. An example of a logical interface is a "tunnel" interface that is used to send IPv6 packets across an IPv4 network by encapsulating the IPv6 packet inside an IPv4header.

Address
An identifier that can be used as the source or destination of IPv6 packets that is assigned at the IPv6layer to an interface or set of interfaces.

Packet
The protocol data unit (PDU) that exists at the IPv6 layer and is composed of an IPv6 header and payload.

Link
MTU The maximum transmission unit (MTU)—the number of bytes in the largest IPv6packet—that can be sent on a link. Because the maximum frame size includes the link-layer medium headers and trailers, the link MTU is not the same as the maximum frame size of the link. The linkMTU is the same as the maximum payload size of the link-layer technology. For example, for Ethernet Using Ethernet II encapsulation, the maximum Ethernet frame payload size is 1500 bytes. Therefore,the link MTU is 1500. For a link with multiple link-layer technologies (for example, a bridged link),the link MTU is the smallest link MTU of all the link-layer technologies present on the link

Path

MTU The maximum-sized IPv6 packet that can be sent without performing host fragmentation between a source and destination over a path in an IPv6 network. The path MTU is typically the smallest link MTU of all the links in the path.

IPv6 Address Format

Whereas IPv4 addresses use a dotted-decimal format, where each byte ranges from 0 to255.IPv6 addresses use eight sets of four hexadecimal addresses (16 bits in each set), separated by a colon (:),like this: xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx (x would be a hexadecimal value). This Notation is commonly called string notation.

Hexadecimal values can be displayed in either lower- or upper-case for the numbers A–F..A leading zero in a set of numbers can be omitted; for example, you could either enter 0012 or 12 in one of the eight fields—both are correct. If you have successive fields of zeroes in an IPv6 address, you can represent them as two colons (::). For example, 0:0:0:0:0:0:0:5 could be represented as ::5; andABC:567:0:0:8888:9999:1111:0 could be represented as ABC:567::8888:9999:1111:0. However, you can only do this once in the address: ABC::567::891::00 would be invalid since :: appears more than once in the address. The reason for this limitation is that if you had two or more repetitions, you wouldn't know how many sets of zeros were being omitted from each part. An unspecified address is represented as ::, since it contains all zeros.

Types of IPv6 Addresses

Anycast

An anycast address identifies one or more interfaces. Notice that the term device isn’t used since a device can have more than one interface. Sometimes people use the term node to designate an interface on a device. Basically, an anycast is a hybrid of a unicast and multicast address.

- With a unicast, one packet is sent to one destination;
- With a multicast, one packet is sent to all members of the multicast group;
- With an anycast, a packet is sent to any one member of a group of devices that are configured

with the anycast address. By default, packets sent to an anycast address are forwarded to the closest interface (node), which is based on the routing process employed to get the packet to the destination. Given this process, anycast addresses are commonly referred to as one-to-the-nearest addresses.

Multicast

Represent a group of interfaces interested in seeing the same traffic.

• The first 8 bits are set to FF.

• The next 4 bits are the lifetime of the address: 0 is permanent and 1 is temporary.

• The next 4 bits indicate the scope of the multicast address (how far the packet can travel):

1 is for a node, 2 is for a link, 5 is for the site, 8 is for the organization, and E is global(the Internet).

Unicast

The following types of addresses are unicast IPv6 addresses:

• Global unicast addresses

• Link-local addresses

• Site-local addresses

• Unique local addresses

• Special addresses

• Transition addresses

Global Unicast Addresses

IPv6 global addresses are equivalent to public IPv4 addresses. They are globally routable and reachable on the IPv6 Internet. Global unicast addresses are designed to be aggregated or summarized for an efficient routing infrastructure. Unlike the current IPv4-based Internet, which is a mixture of both flat and hierarchical routing, the IPv6-based Internet has been designed from its foundation to support efficient, hierarchical addressing and routing. The scope of a global address is the entire IPv6Internet. RFC 4291 defines global addresses as all addresses that are not the unspecified, loopback,link-local unicast, or multicast addresses. However, Figure shows the structure of global unicast addresses defined in RFC 3587 that are currently being used on the IPv6 Internet. The structure of global unicast addresses defined in RFC 3587. The fields in the global unicast address are described in the following list:

Global Routing Prefix Indicates the global routing prefix for a specific organization’s site. The Combination of the three fixed bits and the 45-bit Global Routing Prefix is used to create a 48-bit site prefix, which is assigned to an individual site of an organization. A site is an autonomously operatingIP-based network that is connected to the IPv6 Internet. Network architects and administrators within the site determine the addressing plan and routing policy for the organization network. Once assigned,routers on the IPv6 Internet forward IPv6 traffic matching the 48-bit prefix to the routers of the organization's site.

Subnet ID The Subnet ID is used within an organization’s site to identify subnets within its site. The Size of this field is 16 bits. The organization’s site can use these 16 bits within its site to create 65,536subnets or multiple levels of addressing hierarchy and an efficient routing infrastructure. With 16 bits in subnetting flexibility, a global unicast prefix assigned to an organization site is equivalent to a publicIPv4 Class A address prefix (assuming that the last octet is used for identifying nodes on subnets). Therouting structure of the organization’s network is not visible to the ISP.Interface ID Indicates the interface on a specific subnet within the site. The size of this field is 64 bits. The interface ID in IPv6is equivalent to the node ID or host ID in IPv4.

Local-Use Unicast Addresses

Local-use unicast addresses do not have a global scope and can be reused. There are two types of local-use unicast addresses: Link-local addresses are used between on-link neighbors and forNeighbor Discovery processes. Site-local addresses are used between nodes communicating with other nodes in the same organization.

Link-Local Addresses FE8:: through FEB::

Link-local addresses are a new concept in IPv6. These kinds of addresses have a smaller scope as to how far they can travel: just the local link (the data link layer link). Routers will process packets destined to a link-local address, but they will not forward them to other links. Their most common uses for a device to acquire unicast site-local or global unicast addressing information, discovering the default gateway, and discovering other layer 2 neighbors on the segment. IPv6 link-local addresses identified by the initial 10 bits being set to 1111 1110 10 and the next 54 bits set to 0, are used by nodes when communicating with neighboring nodes on the same link. For example, on a single-link IPv6network with no router, link-local addresses are used to communicate between hosts on the link. IPv6link-local addresses are similar to IPv4 link-local addresses defined in RFC 3927 that use the169.254.0.0/16 prefix. The use of IPv4 link-local addresses is known as Automatic Private IPAddressing (APIPA) in Windows Vista, Windows Server 2008, Windows Server 2003, and WindowsXP. The scope of a link local address is the local link. A link-local address is required for someNeighbor Discovery processes and is always automatically configured, even in the absence of all other unicast addresses. Link-local addresses always begin with FE80. With the 64-bit interface identifier,the prefix for link-local addresses is always FE80::/64.

Site-Local Addresses FEC:: through FFF::

It represents a particular site or company. These addresses can be used within a company without having to waste any public IP addresses—not that this is a concern, given the large number of addresses available in IPv6. However, by using private addresses, you can easily control who is allowed to leave your network and get returning traffic back by setting up address translation policies for IPv6.

Site-local addresses, identified by setting the first 10 bits to 1111 1110 11, are equivalent to the IPv4private address space (10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16). For example, private intranets that do not have a direct, routed connection to the IPv6 Internet can use site local addresses conflicting with global addresses. Site-local addresses are not reachable from other sites, and routers must not forward site-local traffic outside the site. Site-local addresses can be used in addition to globaladdresses. The scope of a site-local address is the site. Unlike link-local addresses, site-local addresses are not automatically configured and must be assigned either through stateless or stateful address autoconfiguration. The first 10 bits are always fixed for site-local addresses, beginning with FEC0::/10.After the 10 fixed bits is a 54-bit Subnet ID field that provides 54 bits with which you can create subnets within your organization. You can have a flat subnet structure, or you can divide the high order bits of the Subnet ID field to create a hierarchical and summarized routing infrastructure. After the SubnetID field is a 64-bit Interface ID field that identifies a specific interface on a subnet. Site-local addresses have been formally deprecated in RFC 3879 for future IPv6 implementations. However, existing implementations of IPv6 can continue to use site-local addresses.

Zone IDs for Local-Use Addresses

Unlike global addresses, local-use addresses (link-local and site-local addresses) can be reused.Link-local addresses are reused on each link. Site-local addresses can be reused within each site of an organization. Because of this address reuse capability, link-local and site-local addresses are ambiguous. To specify the link on which the destination is located or the site within which the destination is located, an additional identifier is needed. This additional identifier is a zone identifier(ID), also known as a scope ID, which identifies a connected portion of a network that has a specified scope. The syntax specified in RFC 4007 for identifying the zone associated with a local-use addressis Address%zone ID, in which Address is a local-use unicast IPv6 address and zone ID is an integer value representing the zone. The values of the zone ID are defined relative to the sending host.

Therefore, different hosts might determine different zone ID values for the same physical zone. For Example, Host A might choose 3 to represent the zone of an attached link and Host B might choose 4to represent the same link.

Unique Local Addresses

Site-local addresses provide a private addressing alternative to global addresses for internet traffic. However, because the site-local address prefix can be reused to address multiple sites within an organization, a site-local address prefix can be duplicated. The ambiguity of site local addresses in an organization adds complexity and difficulty for applications, routers, and network managers

To replace site-local addresses with a new type of address that is private to an organization yet unique across all the sites of the organization, RFC 4193 defines unique local IPv6 unicast addresses.

The first 7 bits have the fixed binary value of 1111110. All local addresses have the address prefixFC00::/7. The Local (L) flag is set 1 to indicate that the prefix is locally assigned. The L flag value set to 0 is not defined in RFC 3879. Therefore, unique local addresses within an organization with the Lflag set to

1 have the address prefix of FD00::/8. The Global ID identifies a specific site within an organization and is set to a randomly derived 40-bit value. By deriving a random value for the GlobalID, an organization can have statistically unique 48-bit prefixes assigned to their sites. Additionally,two organizations that use unique local addresses that merge have a low probability of duplicating a48-bit unique local address prefix, minimizing site renumbering. Unlike the Global Routing Prefix in global addresses, the Global IDs in unique local address prefixes are not designed to be summarized.

The following are the special IPv6 addresses:

Unspecified address

The unspecified address (0:0:0:0:0:0:0:0 or ::) is used only to indicate the absence of an address. It is equivalent to the IPv4 unspecified address of 0.0.0.0. The unspecified address is typically used as a source address when a unique address has not yet been determined. The unspecified address is neverassigned to an interface or used as a destination address.

Loopback address

The loopback address (0:0:0:0:0:0:0:1 or ::1) is assigned to a loopback interface, enabling a node to send packets to itself. It is equivalent to the IPv4 loopback address of 127.0.0.1. Packets addressed to the loopback address must never be sent on a link or forwarded by an IPv6 router.

Transition Addresses

To aid in the transition from IPv4 to IPv6 and the coexistence of both types of hosts, the following addresses are defined:

IPv4-compatible address

The IPv4-compatible address, 0:0:0:0:0:0:w.x.y.z or ::w.x.y.z (where w.x.y.z is the dotted decimal representation of a public IPv4 address), is used by IPv6/IPv4 nodes that are communicating with IPv6over an IPv4 infrastructure that uses public IPv4 addresses, such as the Internet. IPv4-compatible addresses are deprecated in RFC 4291 and are not supported in IPv6 for Windows Vista and WindowsServer 2008.

IPv4-mapped address

The IPv4-mapped address, 0:0:0:0:0:FFFF:w.x.y.z or ::FFFF: w.x.y.z, is used to represent an IPv4address as a 128-bit IPv6 address.

ISATAP address

An address of the type 64-bit prefix:0:5EFE:w.x.y.z, where w.x.y.z is a private IPv4 address, is assignedto a node for the Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) IPv6 transition technology.

Teredo address

A global address that uses the prefix 2001::/32 and is assigned to a node for the Teredo IPv6 transition technology. Beyond the first 32 bits, Teredo addresses are used to encode the IPv4 address of a Teredo Server, flags, and an obscured version of a Teredo client's external address and UDP port number.

Assigning IPv6 address to Devices

IPv6 Addresses for a Host

An IPv4 host with a single network adapter typically has a single IPv4 address assigned to that adapter.An IPv6 host, however, usually has multiple IPv6 addresses assigned to each adapter. The interfaces on a typical IPv6 host are assigned the following unicast addresses:

A link-local address for each interface

Additional unicast addresses for each interface (which could be one or multiple unique local or global addresses)

• The loopback address (::1) for the loopback interface: Typical IPv6 hosts are always logically multi homed because they always have at least two addresses with which they can receive packets—a link-local address for local link traffic and a routable unique local or global address.Additionally, each interface on an IPv6 host is listening for traffic on the following multicast addresses:

• The interface-local scope all-nodes multicast address (FF01::1)

• The link-local scope all-nodes multicast address (FF02::1)

• The solicited-node address for each unicast address assigned

• The multicast addresses of joined groups

SUBNETTING

A subnetwork or subnet is a logical subdivision of an IP network. The practice of dividing a network into two or more networks is called subnetting. Computers that belong to a subnet are addressed with an identical most-significant bit-group in their IP addresses.

Advantage of Subnetting

• Subnetting allows us to break a single large network in smaller networks. Small Networks are easy to manage.

• Subnetting reduces network traffic by allowing only the broadcast traffic which is relevant to the subnet.

• By reducing unnecessary traffic, Subnetting improves overall performance of the network.

• By blocking a subnet' traffic in the subnet, Subnetting increases security of the network.

• Subnetting reduces the requirement of IP range.

Disadvantage of Subnetting

• Different subnets need an intermediate device known as router to communicate with each otherSince each subnet uses its own network address and broadcast address, more subnetsmean more wastage of IP addresses.

• Subnetting adds complexity in the network. An experienced network administrator is required to manage the subnetted network.

Class A Subnets

In Class A, only the first octet is used as Network identifier and the rest of three octets are used to be assigned to Hosts (i.e. 16777214 Hosts per Network). To make more subnet in Class A, bits fromHost part are borrowed and the subnet mask is changed accordingly.For example, if one MSB (Most Significant Bit) is borrowed from host bits of second octet and added to Network address, it creates two Subnets (21=2) with (223-2) 8388606 Hosts per Subnet.

Class B Subnets

By default, using Classful Networking, 14 bits are used as Network bits providing (214) 16384Networks and (216-2) 65534 Hosts. Class B IP Addresses can be subnetted the same way as Class Aaddresses, by borrowing bits from Host bits.

Class C Subnets

Class C IP addresses are normally assigned to a very small size network because it can only have 254 hosts in a network.

RESULT:

Thus the IPV6 address terminology, IPV6 address format, Types of Addresses, assigning IPaddresses to devices and subnetting were studied successfully

Ex No: 12	**Implementation of NAT**
Date:	

AIM:

To implement Network Address Translation (NAT) Protocol using Java program

ALGORITHM

• A new datagram packet is made consisting of the data from the client and is sent to the server with the device's assigned addresses - hence performing NAT.

• The server on receiving the packet will display the message and then ask for a response. This response is then sent to the device i.e. the address from where the packet came.

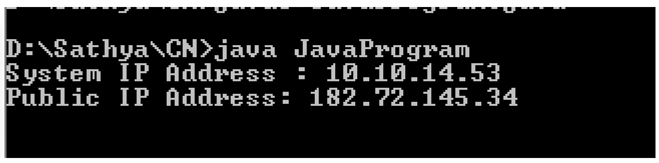
• The device will then print the System IP Address

• If An error "Cannot Execute Properly"message is sent either by client or server, the entire established connection will be terminated.

PROGRAM:

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

public class JavaProgram
{
    public static void main(String args[]) throws Exception
    {
        // Returns the instance of InetAddress containing
        // local host name and address
        InetAddress localhost = InetAddress.getLocalHost();
        System.out.println("System IP Address : " +
                    (localhost.getHostAddress()).trim());

        // Find public IP address
        String systemipaddress = "";
        try
        {
            URL url_name = new URL("http://bot.whatismyipaddress.com");

            BufferedReader sc =
            new                                        BufferedReader(new
InputStreamReader(url_name.openStream()));

            // reads system IPAddress
            systemipaddress = sc.readLine().trim();
        }
        catch (Exception e)
        {
            systemipaddress = "Cannot Execute Properly";
        }
        System.out.println("Public IP Address: " + systemipaddress +"\n");
    }
}
```

Output:



```
D:\Sathya\CN>java JavaProgram
System IP Address : 10.10.14.53
Public IP Address: 182.72.145.34
```

RESULT:

Thus NAT has been implemented

Ex No: 13	**Implementation of VPN**
Date:	

AIM:

To implement Virtual Private Network (VPN) protocol using Java program

PROGRAM (VPN):

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

```
public class UDP_VPN{
static String encode(String[] array){
        StringBuilder output=new StringBuilder();
        for (String i:array ) {
                output.append((i.length()+1)+"-"+i);
        }
        return output.toString();
}

        static String[] decode(String encoded){
                StringBuilder en=new StringBuilder(encoded);
                ArrayList<String> output = new ArrayList<String>();
                int len,i;
                for (i=0,len=0;en.length()!=0;++i,len=0) {
                        while(Character.isDigit(en.charAt(0))){
                                len=len*10+Integer.parseInt(en.charAt(0)+"");
                                en.deleteCharAt(0);
                        }
                        output.add(en.substring(1,len));//consider region after -
                        en.delete(0,len);
                }
                return output.toArray(new String[0]);
        }

        public static void main(String[] args) {
                System.out.println();
                Scanner sc=new Scanner(System.in);
                String inMsg="",outMsg="";
                //for time being they will be same
                String        secretAuthenticationKey        =        "AbraKaDabra",
validAuthenticationRespnse="readyToUse";
                byte[] inData,outData;
                InetAddress clientAddress,serverAddress;
                int
clientPort,serverPort,vpnPublicPort=3333,vpnPrivatePort=5555;
                String[] clientRequest,responseToClient= new String[3];
                boolean validUser=false;
```

```
            String vpnPublicIP="",vpnPrivateIP="";
            try{
                    vpnPublicIP=InetAddress.getLocalHost().getHostAddress();
                    vpnPrivateIP=InetAddress.getLocalHost().getHostAddress();
            }catch(Exception e){
                    e.printStackTrace();
            }

            try{
                    DatagramSocket ds=new DatagramSocket(vpnPublicPort);
                    DatagramPacket dsp,drp;
                    inData=new byte[1024];
                    drp=new DatagramPacket(inData,inData.length);
                    ds.receive(drp);
                    clientAddress=drp.getAddress();
                    clientPort=drp.getPort();
                                            /*String clientaddr;
                                            clientaddr=clientAddress.toString();

        Class.forName("com.mysql.jdbc.Driver");
                                            Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vpn","root"
,"");
                                            PreparedStatement
stmt=con.prepareStatement("insert into ipad values(?,?,?,?)");
                                            stmt.setString(1,clientaddr);
                                            stmt.setInt(2,clientPort);
                                            stmt.setString(3,vpnPrivateIP);
                                            stmt.setInt(4,vpnPrivatePort);
                                            int i=stmt.executeUpdate();
                                            con.close();*/
                    inMsg=new String(drp.getData(),0,drp.getLength());
                    clientRequest = decode(inMsg);
                    System.out.println("Client              msg              :
"+Arrays.toString(clientRequest));
                    if (clientRequest[2].equals(secretAuthenticationKey)) {
                        responseToClient[0]=vpnPrivateIP;//private          ip
address of vpn
                        responseToClient[1]=vpnPrivatePort+"";
                        responseToClient[2]=validAuthenticationRespnse;
                        validUser=true;
                        System.out.println("Client Authenticated\n");
                  }
                  else{
                        responseToClient[0]=vpnPrivateIP;
                        responseToClient[1]=vpnPublicPort+"";
                        responseToClient[2]="requestDenied";
                        validUser=false;
                        System.out.println("Unauthorized client");
                  }
                  outMsg=encode(responseToClient);
                  outData=new byte[1024];
                  outData=outMsg.getBytes();
                  dsp=new
DatagramPacket(outData,outData.length,clientAddress,clientPort);
                  ds.send(dsp);
                  ds.close();
            }
            catch (Exception e) {
                  System.out.println(e.toString());
            }


            if (validUser) {
                  try{
                        DatagramSocket                          ds=new
DatagramSocket(vpnPrivatePort);
                        try{
                              DatagramPacket dsp,drp;
                              do{
                                    //take message from client
                                    inData=new byte[1024];
                                    drp=new
DatagramPacket(inData,inData.length);
                                    ds.receive(drp);
                                    clientAddress=drp.getAddress();
                                    clientPort=drp.getPort();

                                    inMsg=new
String(drp.getData(),0,drp.getLength());
                                    clientRequest=decode(inMsg);
                                    System.out.println("Client    msg    :
"+Arrays.toString(clientRequest));

                                    //forwarding client's message to server

                serverAddress=InetAddress.getByName(clientRequest[0]);

                serverPort=Integer.parseInt(clientRequest[1]);
                                    outMsg=clientRequest[2];
                                    outData=new byte[1024];
                                    outData=outMsg.getBytes();
                                    dsp=new
DatagramPacket(outData,outData.length,serverAddress,serverPort);
                                    ds.send(dsp);
                                    if (outMsg.equalsIgnoreCase("bye")) {
                                        break;
                                    }

                                    //take response msg from server
                                    inData=new byte[1024];
                                    drp=new
DatagramPacket(inData,inData.length);
                                    ds.receive(drp);
                                    serverAddress=drp.getAddress();
                                    serverPort=drp.getPort();
                                    inMsg=new
String(drp.getData(),0,drp.getLength());
```

```
                                        //forwarding server response to client

     responseToClient[0]=serverAddress.getHostAddress();
                                        responseToClient[1]=serverPort+"";
                                        responseToClient[2]=inMsg;
                                        System.out.println("Server      msg      :
"+Arrays.toString(responseToClient));
                                        outMsg=encode(responseToClient);
                                        outData=new byte[1024];
                                        outData=outMsg.getBytes();
                                        dsp=new
DatagramPacket(outData,outData.length,clientAddress,clientPort);
                                        ds.send(dsp);

                                   }while
(!inMsg.equalsIgnoreCase("bye")&&validUser);
                                   ds.close();
                              }
                              catch (Exception e) {
                                   System.out.println(e.toString());
                              }
                              finally{
                                   ds.close();
                                   System.out.println();
                              }
                         }catch(Exception e){}
                    }
          }
}
```

PROGRAM (Client Side):

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

public class UDPClient{
	static String encode(String[] array){
		StringBuilder output=new StringBuilder();
		for (String i:array ) {
			output.append((i.length()+1)+"-"+i);
		}
		return output.toString();
	}

	static String[] decode(String encoded){
		StringBuilder en=new StringBuilder(encoded);
		ArrayList<String> output = new ArrayList<String>();
		int len,i;
		for (i=0,len=0;en.length()!=0;++i,len=0) {
			while(Character.isDigit(en.charAt(0))){
				len=len*10+Integer.parseInt(en.charAt(0)+"");
				en.deleteCharAt(0);
			}
			output.add(en.substring(1,len));//consider region after -
			en.delete(0,len);
		}
		return output.toArray(new String[0]);
	}

	public static void main(String[] args) {
		System.out.println();
		String inMsg = "", outMsg = "";
		int format = 3, serverPort, vpnPublicPort = 3333,vpnPrivatePort;
		boolean vpnConnect=false;
		String vpnPublicIP="",serverIP,vpnPrivateIP;
		String                    secretAuthenticationKey                    =
"AbraKaDabra",validAuthenticationRespnse="readyToUse";
		String[] authentication = new String[format], vpnDetails = new
String[format], vpnResponse, server;
		Scanner sc = new Scanner(System.in);
		byte[] inData, outData;
		try{
        vpnPublicIP=InetAddress.getLocalHost().getHostAddress();//you can
also make it as user input
                        DatagramSocket ds = new DatagramSocket();
                        DatagramPacket dsp, drp;
                        authentication[0] = vpnPublicIP;//public ip address of vpn
-- dummy
                        authentication[1] = ""+vpnPublicPort;//public port of vpn
                        authentication[2] = secretAuthenticationKey;//key is
required to establish a connection
                        outMsg = encode(authentication);
                        outData = outMsg.getBytes();
                        dsp = new DatagramPacket(outData, outData.length,
InetAddress.getByName(vpnPublicIP), vpnPublicPort);
                        ds.send(dsp);
                        System.out.println("asking for authentication from vpn
"+secretAuthenticationKey);

                        inData = new byte[1024];
                        drp = new DatagramPacket(inData, inData.length);
                        ds.receive(drp);
                        inMsg = new String(drp.getData(), 0, drp.getLength());
                        vpnDetails = decode(inMsg);
                        if(vpnDetails[2].equals(validAuthenticationRespnse)){
                                vpnConnect = true;

        vpnPrivateIP=InetAddress.getLocalHost().getHostAddress();//this can
be different
                                vpnPrivatePort=Integer.parseInt(vpnDetails[1]);
                                System.out.println("Connection is established with
vpn");
                                System.out.println("VPN Details : " +
Arrays.toString(vpnDetails));
                        }
                        else{
                                System.out.println("Connection refused by vpn
"+Arrays.toString(vpnDetails));
```

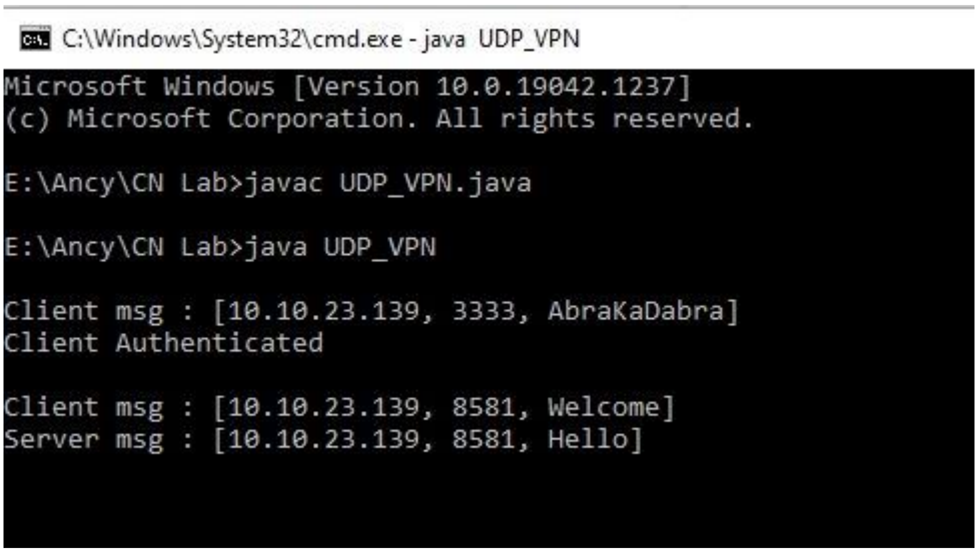
```
                        System.out.println();
                    }
                }
                catch(Exception e){
                    e.printStackTrace();
                }
                if (vpnConnect==true) {
                    //below part will run after connection is established with
the vpn
                    System.out.print("Enter Initial Server IP : ");
                    serverIP = sc.next();
                    System.out.print("Enter Initial server Port : ");
                    serverPort = sc.nextInt();
                    sc.nextLine();
                    System.out.println();
                    server = new String[]{serverIP, "" + serverPort,""};
                    try{
                        DatagramSocket ds=new DatagramSocket();
                        DatagramPacket dsp,drp;
                        do{
                            System.out.print("Enter something : ");
                            server[2] = sc.nextLine();
                            outMsg = encode(server);
                            outData = outMsg.getBytes();
                            dsp     =     new     DatagramPacket(outData,
outData.length,                    InetAddress.getByName(vpnDetails[0]),
Integer.parseInt(vpnDetails[1]));
                            ds.send(dsp);
                            if (server[2].equalsIgnoreCase("bye")) {
                                break;
                            }
                            inData = new byte[1024];
                            drp                =                new
DatagramPacket(inData,inData.length);
                            ds.receive(drp);
                            inMsg    =    new    String(drp.getData(),    0,
drp.getLength());
```

```
                                        vpnResponse = decode(inMsg);
                                        System.out.println("Server     msg     :     "     +
vpnResponse[2]);

                                }while
(!vpnResponse[2].equalsIgnoreCase("bye")&&vpnConnect);
                                ds.close();
                        }
                        catch (Exception e) {
                                System.out.println(e.toString());
                        }
                }
        }
}
```

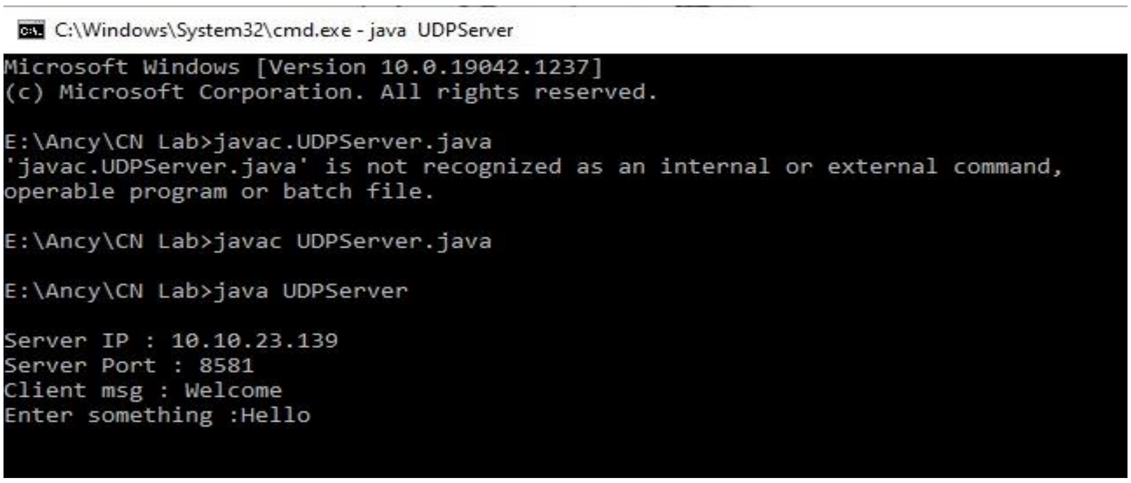
PROGRAM (Server Side):

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

public class UDPServer{
        public static void main(String[] args) {
                System.out.println();
                Scanner sc=new Scanner(System.in);
                String inMsg="",outMsg="";
                byte[] inData,outData;
                InetAddress clientAddress;
                int clientPort;
                try{
                        DatagramSocket ds=new DatagramSocket(8581);
                        try{
                                //ds=new DatagramSocket(8515);
                                System.out.println("Server                  IP                :
"+InetAddress.getLocalHost().getHostAddress());
                                System.out.println("Server                Port                :
"+ds.getLocalPort());
                                DatagramPacket dsp,drp;
                                do{
                                        inData=new byte[1024];
                                        drp=new
DatagramPacket(inData,inData.length);
                                        ds.receive(drp);
                                        clientAddress=drp.getAddress();
                                        clientPort=drp.getPort();
                                        inMsg=new
String(drp.getData(),0,drp.getLength());
                                        System.out.println("Client msg : "+inMsg);
                                        if (inMsg.equalsIgnoreCase("bye")) {
                                                break;
                                        }

                                        System.out.print("Enter something :");
                                        outMsg=sc.nextLine();
                                        outData=new byte[1024];
                                        outData=outMsg.getBytes();
                                        dsp=new
DatagramPacket(outData,outData.length,clientAddress,clientPort);
                                        ds.send(dsp);
                                }while (!outMsg.equalsIgnoreCase("bye"));
                                ds.close();
                        }
                        catch (Exception e) {
                                System.out.println(e.toString());
                                System.out.println();
                        }
                        finally{
                                ds.close();
                        }
                }
                catch(Exception e){}
        }
}
```

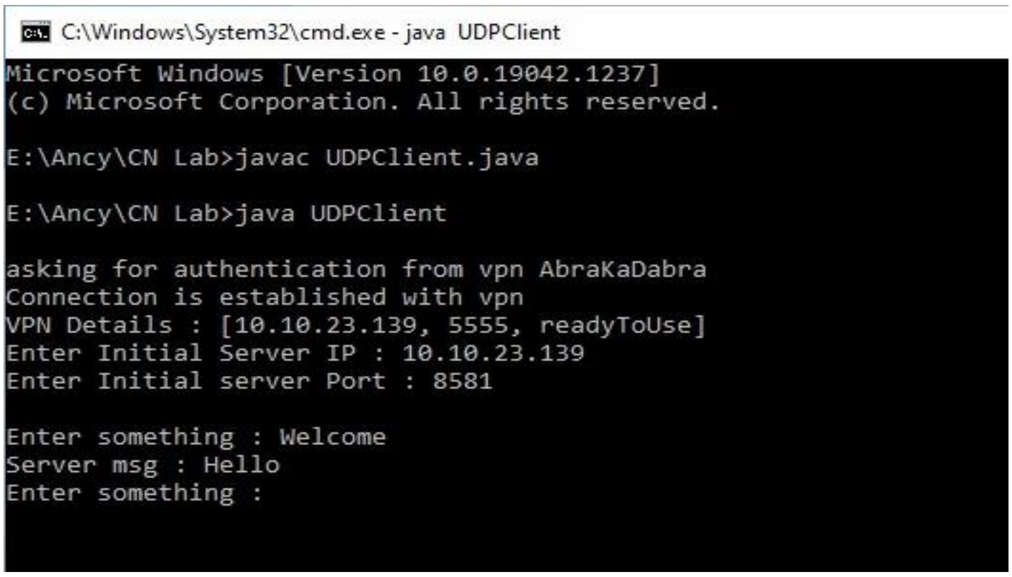
OUTPUT


C:\Windows\System32\cmd.exe - java UDP_VPN

```
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

E:\Ancy\CN Lab>javac UDP_VPN.java

E:\Ancy\CN Lab>java UDP_VPN

Client msg : [10.10.23.139, 3333, AbraKaDabra]
Client Authenticated

Client msg : [10.10.23.139, 8581, Welcome]
Server msg : [10.10.23.139, 8581, Hello]
```


C:\Windows\System32\cmd.exe - java UDPServer

```
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

E:\Ancy\CN Lab>javac.UDPServer.java
'javac.UDPServer.java' is not recognized as an internal or external command,
operable program or batch file.

E:\Ancy\CN Lab>javac UDPServer.java

E:\Ancy\CN Lab>java UDPServer

Server IP : 10.10.23.139
Server Port : 8581
Client msg : Welcome
Enter something :Hello
```


C:\Windows\System32\cmd.exe - java UDPClient

```
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

E:\Ancy\CN Lab>javac UDPClient.java

E:\Ancy\CN Lab>java UDPClient

asking for authentication from vpn AbraKaDabra
Connection is established with vpn
VPN Details : [10.10.23.139, 5555, readyToUse]
Enter Initial Server IP : 10.10.23.139
Enter Initial server Port : 8581

Enter something : Welcome
Server msg : Hello
Enter something :
```

RESULT:

Thus the implementation of Virtual Private Network (VPN) protocol has been done

Ex No: 14(a)	Communication Using HDLC
Date:	

AIM:

To study the concept and different frames of HDLC protocol.

PROCEDURE:

High-Level Data Link Control (HDLC) is a bit-oriented code-transparent synchronous data link layer protocol developed by the International Organization for Standardization (ISO) .

The original ISO standards for HDLC are:

1. ISO 3309 – Frame Structure
2. ISO 4335 – Elements of Procedure
3. ISO 6159 – Unbalanced Classes of Procedure
4. ISO 6256 – Balanced Classes of Procedure

The current standard for HDLC is ISO 13239, which replaces all of those standards.HDLC provides both connection-oriented and connectionless service. HDLC can be used for point to multipoint connections, but is now used almost exclusively to connect one device to another, using what is known asAsynchronous Balanced Mode (ABM). The original master-slave modes Normal Response Mode (NRM) and Asynchronous Response Mode (ARM) are rarely used.HDLC is based on IBM's SDLC protocol, which is the layer 2 protocol for IBM's Systems Network Architecture (SNA). It was extended and standardized by the ITU as LAP, while ANSI named their essentially identical version ADCCP.

Derivatives have since appeared in innumerable standards. It was adopted into the X.25 protocol stack as LAPB, into the V42 protocol as LAMP, into the Frame Relay protocol stack as LAPF and into the ISDN protocol stack as LAPD.HDLC was the inspiration for the IEEE 802.2 LLC protocol, and it is the basis for the framing mechanism used with the PPP on synchronous lines, as used by many servers to connect to a WAN, most commonly the Internet. A mildly different version is also used as the control channel for E- carrier (E1) and SONET multichannel telephone lines. Some vendors, such as Cisco, implemented protocols such as Cisco HDLC that used the low-level HDLC framing techniques but added a protocol field to the standard HDLC header. More importantly, HDLC is the default encapsulation for serial interfaces on Cisco routers. It has also been used on Tellabs DXX for the destination of Trunk.

FRAMING

HDLC frames can be transmitted over synchronous or asynchronous serial communication links. Those links have no mechanism to mark the beginning or end of a frame, so the beginning and end of each frame has to be identified. This is done by using a frame delimiter, or *flag*, which is a unique sequence of bits that is guaranteed not to be seen inside a frame. This sequence is '01111110', or, in hexadecimal notation, 0x7E. Each frame begins and ends with a frame delimiter. A frame delimiter at the end of a frame may also mark the start of the next frame. A sequence of 7 or more consecutive 1-bits within a frame will cause the frame to be aborted.

When no frames are being transmitted on a simplex or full-duplex synchronous link, a frame delimiter is continuously transmitted on the link. Using the standard NRZI encoding from bits to line levels (0 bit = transition, 1 bit = no transition), this generates one of two continuous waveforms, depending on the initial state:

This is used by modems to train and synchronize their clocks via phase-locked loops. Some protocols allow the 0-bit at the end of a frame delimiter to be shared with the start of the next frame delimiter, i.e. '011111101111110'.

For half-duplex or multi-drop communication, where several transmitters share a line, a receiver on the line will see continuous idling 1-bits in the inter-frame period when no transmitter is active.

Since the flag sequence could appear in user data, such sequences must be modified during transmission to keep the receiver from detecting a false frame delimiter. The receiver must also detect when this has occurred so that the original data stream can be restored before it is passed to higher layer protocols. This can be done using bit stuffing, in which a "0" is added after the occurrence of every "11111" in the data. When the receiver detects these "11111" in the data, it removes the "0" added by the transmitter.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different data flows

3. Open a nam trace file and define the finish procedure then close the trace file, and execute nam on trace file.

4. Create six nodes that forms a network numbered from 0 to 5

5. Create duplex links between the nodes

6. Setup UDP Connection between n(0) and n(2)

7. Apply CBR Traffic over UDP

8. Choose distance vector routing protocol as a high level data link control.

9. Make any one of the links to go down to check the working nature of HDLC

10. Schedule events and run the program.

PROGRAM

```
set ns [new Simulator]

#Tell the simulator to use dynamic routing

$ns rtproto DV

$ns macType Mac/Sat/UnslottedAloha #Open the nam trace file

set nf [open aloha.nam w]

$ns namtrace-all $nf #Open the output files set f0 [open aloha.tr w]

$ns trace-all $f0

#Define a finish procedure proc finish {} {

global ns f0 nf

$ns flush-trace #Close the trace file close $f0

close $nf

exec nam aloha.nam & exit 0

}
```

# Create six nodes set n0 [$ns node] set n1 [$ns node] set n2 [$ns node] set n3 [$ns node] set n4 [$ns node] set n5 [$ns node]

# Create duplex links between nodes with bandwidth and distance

$ns duplex-link $n0 $n4 1Mb 50ms DropTail

$ns duplex-link $n1 $n4 1Mb 50ms DropTail

$ns duplex-link $n2 $n5 1Mb 1ms DropTail

$ns duplex-link $n3 $n5 1Mb 1ms DropTail

$ns duplex-link $n4 $n5 1Mb 50ms DropTail

$ns duplex-link $n2 $n3 1Mb 50ms DropTail

# Create a duplex link between nodes 4 and 5 as queue position

$ns duplex-link-op $n4 $n5 queuePos 0.5 #Create a UDP agent and attach it to node n(0) set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0 set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_ 500

$cbr0 set interval_ 0.005

$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node n(2) set null0 [new Agent/Null]

$ns attach-agent $n2 $null0

#Connect the traffic source with the traffic sink

$ns connect $udp0 $null0

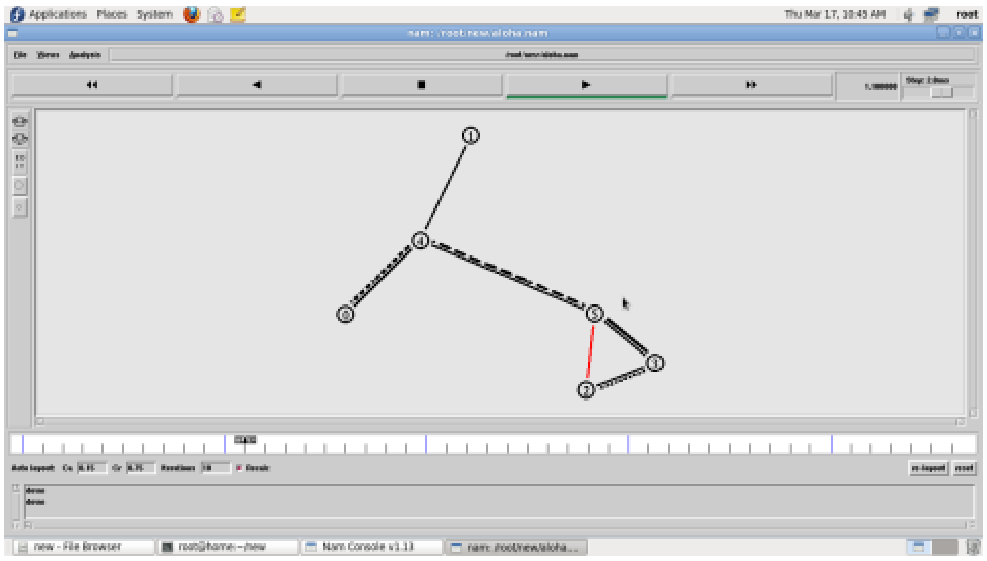
#Schedule events for the CBR agent and the network dynamics

$ns at 0.5 "$cbr0 start"

$ns rtmodel-at 1.0 down $n5 $n2

```
$ns rtmodel-at 2.0 up $n5 $n2

$ns at 4.5 "$cbr0 stop"

#Call the finish procedure after 5 seconds of simulation time

$ns at 5.0 "finish" #Run the simulation

$ns run
```

OUTPUT:



RESULT:

Thus the HDLC is studied and simulated.

Ex No: 14(B)	**Bit Stuffing**

Date:	

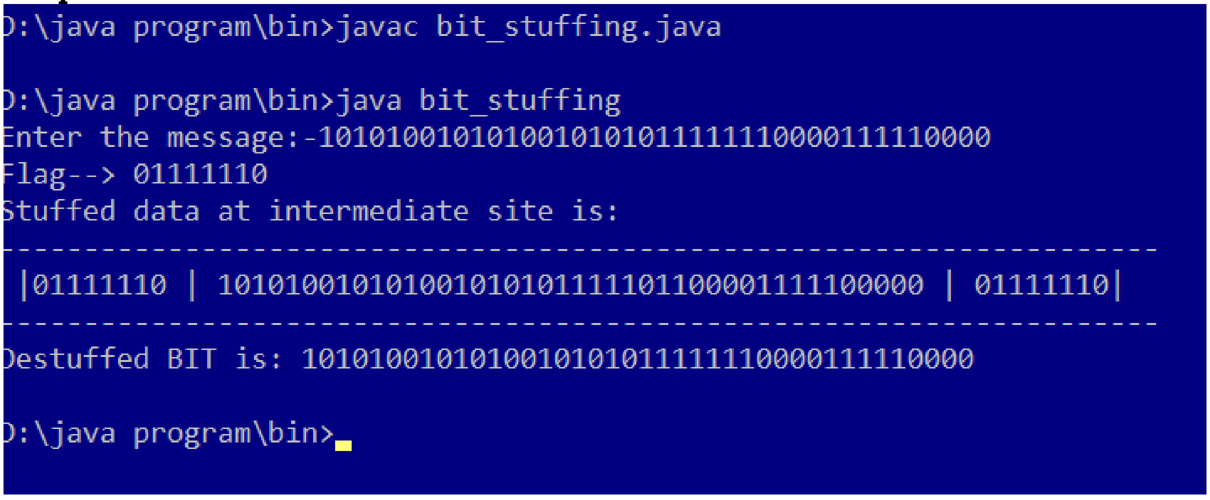
Aim : To write a java program for simulating Bit Stuffing

Program:

```
import java.util.*;
public class bit_stuffing
   {
   public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the message:-");
        String d1 = sc.nextLine();
        String remaining = new String();
        String output=new String();
        int counter = 0;
        for(int i=0;i<d1.length();i++)
         {

           if (d1.charAt(i)!='1' && d1.charAt(i)!='0')
              {
                 System.out.println("Enter valid Binary values");
                 return;
              }
           if(d1.charAt(i) == '1')
              {
                 counter++;
                 remaining = remaining + d1.charAt(i);
              }
           else
              {
                 remaining = remaining + d1.charAt(i);
                 counter = 0;
              }
           if(counter == 5)
              {
                 remaining = remaining + '0';
                 counter = 0;
              }
         }
       System.out.println("Flag--> 01111110");
       String new1="|01111110 | "+remaining+" | 01111110|";
       System.out.println("Stuffed data at intermediate site is:");
       for(int k=0;k<=(28+d1.length());k++)
       {
         System.out.print("-");
       }
       System.out.println();
       System.out.println(" "+new1);
       for(int k=0;k<=(28+d1.length());k++)
       {
         System.out.print("-");
```

```
            }
            System.out.println();
            counter=0;
            for(int i=0;i<remaining.length();i++)
                {

                    if(remaining.charAt(i) == '1')
                        {

                            counter++;
                            output = output + remaining.charAt(i);

                        }
                    else
                        {
                             output = output + remaining.charAt(i);
                             counter = 0;
                        }
                   if(counter == 5)
                        {
                              if((i+2)!=remaining.length())
                              {
                                output = output + remaining.charAt(i+2);
                              }
                              else
                              {
                                output=output + '1';
                              }
                              i=i+2;
                              counter = 1;
                        }
               }
             System.out.println("Destuffed BIT is: "+output);
        }
}
```

Output



```
D:\java program\bin>javac bit_stuffing.java

D:\java program\bin>java bit_stuffing
Enter the message:-1010100101010010101011111110000111110000
Flag--> 01111110
Stuffed data at intermediate site is:
------------------------------------------------------------------
 |01111110 | 101010010101001010101111101100001111100000 | 01111110|
------------------------------------------------------------------
Destuffed BIT is: 1010100101010010101011111110000111110000

D:\java program\bin>
```

Result: Thus the Program for Bit Stuffing Has been executed successfully

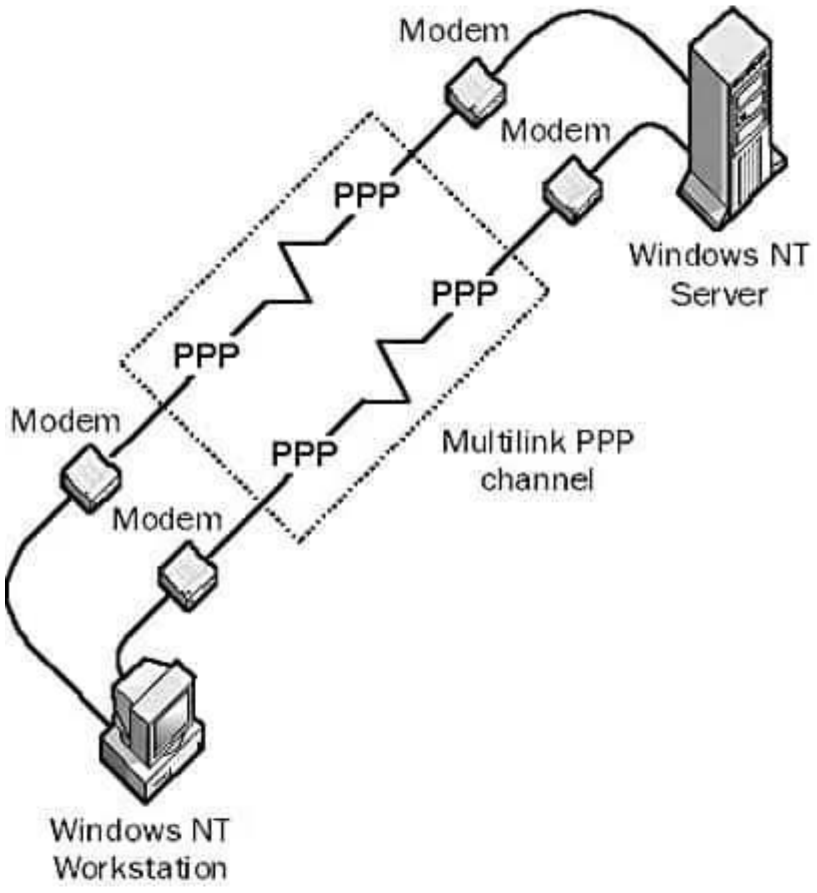
Ex No: 15	Communication Using PPP
Date:	

AIM:

To study Peer to Peer communication protocol

DESCRIPTION:

Point-to-Point Protocol is an industry-standard data-link layer protocol for wide area network (WAN) transmission that was developed in the early 1990s. Point-to-Point Protocol (PPP) allows Remote Access Service (RAS) products and devices from different vendors to interoperate for WAN communication.



Point to Point Protocol (PPP)

How It Works?

PPP supports the transmission of network packets over a serial point-to-point link by specifying framing mechanisms for encapsulating network protocols such as Internet Protocol (IP), Internetwork Packet Exchange (IPX), or NetBEUI into PPP frames.

PPP encapsulation is based on the High-level Data Link Control (HDLC) derived from the mainframe environment. These PPP frames can be transmitted over serial transmission lines such as Plain Old Telephone Service (POTS), Integrated Services Digital Network (ISDN), and packet-switched networks such as X.25.

PPP includes an extensible Link Control Protocol (LCP) for establishing, tearing down, and testing data-link WAN connections, as well as a number of Network Control Protocols (NCPs) for establishing and configuring network communication using each network protocol. PPP also supports a number of authentication schemes, such as Password Authentication Protocol (PAP) and Challenge Handshake Authentication Protocol (CHAP).

A typical dial-up session using PPP is completely automated and requires no real-time user input. It has four stages:

- Link establishment: PPP uses LCP to establish and maintain a PPP link over a serial transmission line. LCP frames are sent over the data link to test its integrity and establish the link.
- User authentication: PPP uses one of several authentication protocols, including PAP, CHAP, and Microsoft Challenge Handshake Authentication Protocol (MS-CHAP).
- Callback: PPP Callback Control (Microsoft's implementation of PPP) uses Callback Control Protocol (CBCP) if it is configured.
- Configuration: NCPs are used to establish network connections, perform compression and encryption, lease IP addresses using Dynamic Host Configuration Protocol (DHCP), and so on. NCP frames are sent over the link to establish a network connection between the PPP server and the remote PPP client.

RESULT:

Thus study of Peer to Peer communication protocol mechanism is done