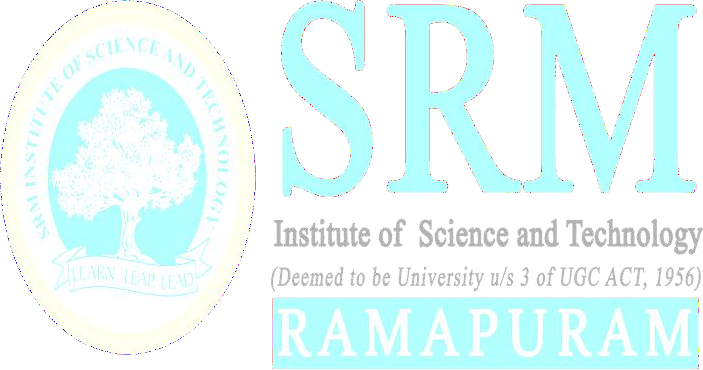
`

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**Ramapuram,Chennai-600089**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

# Department of Computer Science & Engineering Academic Year (Odd 2020 – 2021)



**COMPUTERNETWORKS**

**(18CSC302J)**

**SEMESTER- V**

Name

:

Reg. No

RA1811003020152

Class

: B.Tech (CSE) C - sec

Year

: III year

Shivendra Kumar Jha



`



REG NO:

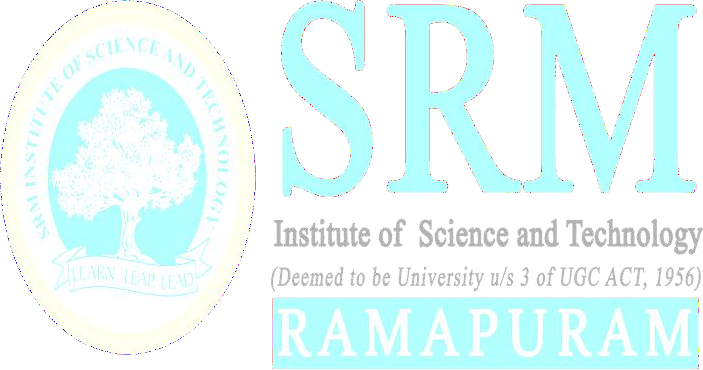
RA1811003020152

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**Ramapuram,Chennai-600089**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

# Department of Computer Science & Engineering



**BONAFIDE CERTIFICATE**

Certified that this is the bonafide record of work done by Shivendra Kumar Jha

of V semester B.Tech Computer Science and Engineering during theacademic year 2020 – 2021, Odd Semester in 18CSC302J - COMPUTERNETWORKS

---------------------------

***Staff-In charge***

--------------------------------------

***Head of the Department***

Submitted for the End Semester Practical Examination heldon 21/11/2020

at SRM Institute of Science & Technology, Ramapuram, Chennai-89.

----------------------------

***Examiner - 1***

-------------------------------

***Examiner - 2***

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Table of Contents** | **Page No.** |
| 1 | Study of header files with respect to socket programming | **4** |
| 2 | Study of Basic Functions of Socket Programming | **6** |
| 3 | Simple Tcp/Ip Client Server Communication | **12** |
| 4 | UDP Echo Client Server Communication | **16** |
| 5 | Concurrent TCP/IP Day-Time Server | **20** |
| 6 | Half Duplex Chat Using TCP/IP | **23** |
| 7 | Full Duplex Chat Using TCP/IP | **27** |
| 8 | Implementation Of File Transfer Protocol | **32** |
| 9 | Remote Command Execution Using UDP | **35** |
| 10 | ARP Implementation Using UDP | **39** |
| 11 | Study of IPV6 addressing | **40** |
| 12 | Network address translation | **46** |
| 13 | Implementation of VPN | **49** |
| 14 | Communication using HDLC | **52** |
| 15 | Communication using PPP | **55** |

## Ex.No:1

**Date:**

**AIM:**

# STUDY OF HEADER FILES WITH RESPECT TO SOCKET PROGRAMMING

TO STUDY HAEDER FILES WITH RESPECT TO SOCKET PROGRAMMING

**HEADER FILES:**

1. **stdio.h:**

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

1. **unistd.h:**

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

1. **string.h:**

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

1. **stdlib.h:**

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

1. **sys/types.h:**

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

1. **sys/socket.h:**

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

1. **netinet/in.h:**

Defines the IPv4 socket address structure commonly called Internet socket address structure calledsockaddr\_in.

1. **netdb.h:**

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

1. **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.

1. **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.

1. **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.

1. **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 besniffed.

1. **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.

1. **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.

1. **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\_ntoafunctions.

**RESULT:**

Thus, the header file with respect to socket programming is studied.

# Ex No: 2

**STUDY OF BASIC FUNCTIONS OF SOCKET PROGRAMMING**

**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,int type,int protocol);

**DESCRIPTION:**

* + Socket creates an endpoint for communication and returns adescriptor.
  + The domain parameter specifies a common domain this selects the protocol family which will be used forcommunication.
  + 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-NovellProtocols. |
| PF\_APPLETALK | Apple Talk. |

* + The socket has the indicated type, which specifies the communicationsemantics.

**TYPES:**

1. **SOCK\_STREAM:**
   * Provides sequenced , reliable, two-way , connection based bytestreams.
   * An out-of-band data transmission mechanism, may besupported.
2. **SOCK\_DGRAM:**
   * Supports datagram (connectionless, unreliable messages of a fixed maximumlength).
3. **SOCK\_SEQPACKET:**
   * Provides a sequenced , reliable, two-way connection based data transmission path for datagrams of fixed maximumlength.
4. **SOCK\_RAW:**
   * Provides raw network protocolaccess.
5. **SOCK\_RDM:**
   * Provides a reliable datagram layer that doesn’t guaranteeordering.
6. **SOCK\_PACKET:**
   * Obsolete and shouldn’t be used in newprograms.
7. **manconnect:**

**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\_t addrlen);

**DESCRIPTION:**

* + The file descriptor sockfd must refer to asocket.
  + 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 anothersocket.

**RETURN VALUE**:

* + If the connection or binding succeeds, zero isreturned.
  + On error , -1 is returned , and error number is setappropriately.

**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. |

1. **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 specifieddestination.
  + The –r option sets the reason for rejecting printjobs.
  + The –e option forces encryption when connecting to theserver.

1. **man send NAME:**

send, sendto, sendmsg - send a message from a socket.

**SYNOPSIS:**

#include<sys/types.h> #include<sys/socket.h>

ssize\_t send(int s, const void \*buf, size\_t len, int flags);

ssize\_t sendto(int s, const void \*buf, size\_t len, int flags, const struct sock\_addr\*to, socklen\_t tolen); ssize\_t sendmsg(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 connectedstate.
  + The only difference between send and write is the presence offlags.
  + The parameter is the file descriptor of the sendingsocket.

1. **man recv NAME:**

recv, recvfrom, recvmsg – receive a message from a socket.

**SYNOPSIS:**

#include<sys/types.h> #include<sys/socket.h>

ssize\_t recv(int s, void \*buf, size\_t len, int flags);

ssize\_t recvfrom(int s, void \*buf, size\_t len, int flags, struct sockaddr \*from, socklen\_t\* from len); ssize\_t recvmsg(int s, struct msghdr \*msg, int flags);

**DESCRIPTION:**

1. **man read NAME:**

* 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 connectionoriented.
* If from is not NULL, and the underlying protocol provides the src addr , this src addr is filledin.
* The recv call is normally used only on a connection socket and is identical to recvfrom with a NULL fromparameter.

read, readonly, return

1. **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 theform: Message from yourname @yourhost on yourtty athh:mm:…
  + Any further lines you enter will be copied to the specified user’sterminal.
  + If the other user wants to reply they must run write aswell.

1. **ifconfig NAME:**

ifconfig- configure a network interface.

**SYNOPSIS:**

ifconfig[interface]

ifconfig interface[aftype] options | address……

**DESCRIPTION:**

* + ifconfig is used to configure the kernel resident networkinterfaces.
  + It is used at boot time to setup interfaces asnecessary.
  + After that, it is usually only needed when debugging or when system tuning isneeded.
  + If no arguments are given, ifconfig displays the status of the currently activeinterfaces.

1. **man bind SYNOPSIS:**

bind[-m keymap] [-lp sv psv]

1. **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\_tnetshort);

**DESCRIPTION:**

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

1. **man gethostname NAME:**

gethostname, sethostname- get/set host name.

**SYNOPSIS:**

#include<unistd.h>

int gethostname(char \*name,size\_t len);

int sethostname(const char \*name,size\_t len);

**DESCRIPTION:**

* + These functions are used to access or to change the host name of the currentprocessor.
  + The gethostname() returns a NULL terminated hostname(set earlier by sethostname()) in the array name that has a length of lenbytes.
  + In case the NULL terminated then hostname does not fit ,no error is returned, but the hostname istruncated.
  + It is unspecified whether the truncated hostname will be NULLterminated.

1. **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,int af);

**DESCRIPTION:**

* + The gethostbyname() returns a structure of type hostent for the givenhostname.
  + Name->hostname or IPV4/IPV6 with dotnotation.
  + gethostbyaddr()- struct of type hostent / host addresslength
  + Address types- AF\_INET,AF\_INET6.
  + sethostent() – stay open istrue(1).
  + TCP socket connection should be open duringqueries.
  + Server queries for UDPdatagrams.
  + endhostent()- ends the use of TCPconnection.
  + Members of hostentstructure:

1. h\_name
2. h\_aliases
3. h\_addrtype
4. h\_length
5. h\_addr-list
6. h\_addr.

**RESULT**:

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

# EX.NO.3 SIMPLE TCP/IPCLIENTSERVER COMMUNICATION

**GIVEN REQUIREMENTS:**

There are two hosts, Client and Server. Both the Client and the Server exchange message

i.e. they send messages or receive message from the other. There is only a single way communication between them.

**TECHNICAL OBJECTIVE:**

To implement a half duplex application, where the Client establishes a connection with the Server. The Client can send and the server well receive messages at the same time.

**METHODOLOGY:**

**Server:**

**Client:**

* + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family to AF\_INET, sin\_addr to INADDR\_ANY, sin\_port to dynamically assigned port number.
  + Bind the local host address to socket using the bindfunction.
  + Listen on the socket for connection request from theclient.
  + Accept connection request from the Client using acceptfunction.
  + Fork the process to receive message from the client and print it on theconsole.
  + Read message from the console and send it to theclient.
  + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family toAF\_INET.
  + Get the server IP address and the Port number from theconsole.
  + Using gethostbyname function assign it to a hostent structure, and assign it to sin\_addr of the server addressstructure.
  + Request a connection from the server using the connectfunction.
  + Fork the process to receive message from the server and print it on theconsole.
  + Read message from the console and send it to theserver.

**CODING:**

**Server: tcpserver.c** #include<stdio.h> #include<stdlib.h> #include<sys/types.h> #include<sys/socket.h> #include<netinet/in.h> #include <unistd.h>

int main(){

char serverMessage[256] = "You have a missed call from server\n";

//create the server socket

int socketDescriptor = socket(AF\_INET,SOCK\_STREAM,0);

//define the server address

//creating the address as same way we have created for TCPclient struct sockaddr\_in serverAddress;

serverAddress.sin\_family = AF\_INET; serverAddress.sin\_port = htons(9002); serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

//calling bind function to oir specified IP and port

bind(socketDescriptor,(struct sockaddr\*) &serverAddress,sizeof(serverAddress)); listen(socketDescriptor,5);

//starting the accepting

//accept(socketWeAreAccepting,structuresClientIsConnectingFrom,) int client\_socket = accept(socketDescriptor, NULL, NULL);

//sending data

//send(toWhom,Message,SizeOfMessage,FLAG); send(client\_socket,serverMessage,sizeof(serverMessage),0);

//close the socket close(socketDescriptor); return 0;

}

**Client: tcpclient.c** #include <stdio.h> #include <stdlib.h>

//for socket and related functions #include <sys/types.h>

#include <sys/socket.h>

//for including structures which will store information needed #include <netinet/in.h>

#include <unistd.h> #define SIZE 1000

//main functions int main()

{

int socketDescriptor = socket(AF\_INET, SOCK\_STREAM, 0);

// server address

struct sockaddr\_in serverAddress; serverAddress.sin\_family = AF\_INET; serverAddress.sin\_port = htons(9002); serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

// communicates with listen

connect(socketDescriptor, (struct sockaddr \*)&serverAddress, sizeof(serverAddress));

char serverResponse[SIZE];

recv(socketDescriptor, &serverResponse, sizeof(serverResponse), 0); printf("Ther server sent the data : %s", serverResponse);

//closing the socket close(socketDescriptor); return 0;

}

**SAMPLE OUTPUT:**

**Server**:

**Server**:

**(Host Name:Root1)**

[root@localhost 4ita33]# vi tcpserver.c [root@localhost 4ita33]# cc tcpserver.c [root@localhost 4ita33]# ./a.out

Server is running…. Message received ishi

Message received ishi

**Client:**

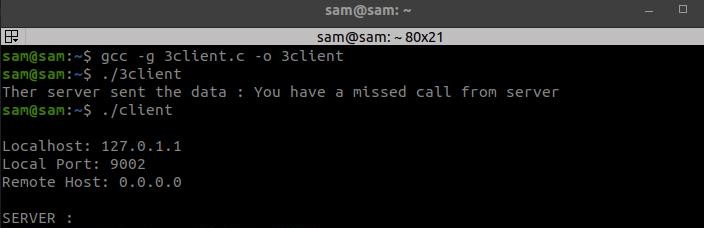
**(Host Name:Root2)**

[root@localhost 4ita33]# vi tcpclient.c [root@localhost 4ita33]# cc tcpclient.c [root@localhost 4ita33]# ./a.out 127.0.0.1 Enter the message:

hi

Data Sent hi Enter the message: how ru

Data Sent how r u

**Enter the message:**

**RESULT:**

Thus, a half-duplex application, where the Client establishes a connection with the Server is implemented successfully.

# ExNo:4 UDP ECHO CLIENT SERVERCOMMUNICATION

**GIVEN REQUIREMENTS:**

There are two hosts, Client and Server. Both the Client and the Server exchange message

i.e. they send messages or receive message from the other. There is only a single way communication between them.

**TECHNICAL OBJECTIVE:**

To implement a half duplex application, where the Client establishes a connection with the Server. The Client can send and the server well receive messages at the same time.

**METHODOLOGY:**

**Server:**

**Client:**

* + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family to AF\_INET, sin\_addr to INADDR\_ANY, sin\_port to dynamically assigned port number.
  + Bind the local host address to socket using the bindfunction.
  + Listen on the socket for connection request from theclient.
  + Accept connection request from the Client using acceptfunction.
  + Fork the process to receive message from the client and print it on theconsole.
  + Read message from the console and send it to theclient.
  + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family toAF\_INET.
  + Get the server IP address and the Port number from theconsole.
  + Using gethostbyname function assign it to a hostent structure, and assign it to sin\_addr of the server addressstructure.
  + Request a connection from the server using the connectfunction.
  + Fork the process to receive message from the server and print it on theconsole.
  + Read message from the console and send it to theserver.

**CODING:**

**Server: udpserver.c**

#include<sys/socket.h> #include<stdio.h> #include<unistd.h> #include<string.h> #include<netinet/in.h> #include<netdb.h> #include<arpa/inet.h> #include<sys/types.h>

int main(int argc,char \*argv[])

{

int sd;

char buff[1024];

struct sockaddr\_in cliaddr,servaddr; socklen\_t clilen; clilen=sizeof(cliaddr);

**/\*UDP socket is created, an Internet socket address structure isfilledwith wildcard address & server’s well knownport\*/**

sd=socket(AF\_INET,SOCK\_DGRAM,0); if (sd<0)

{

perror ("Cannot open Socket"); exit(1);

}

bzero(&servaddr,sizeof(servaddr));

**/\*Socket address structure\*/** servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); servaddr.sin\_port=htons(5669);

**/\*Bind function assigns a local protocol address to the socket\*/**

if(bind(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr))<0)

{

perror("error in binding the port"); exit(1);

}

printf("%s","Server is Running…\n"); while(1)

{

bzero(&buff,sizeof(buff));

**/\*Read the message from the client\*/**

if(recvfrom(sd,buff,sizeof(buff),0,(struct sockaddr\*)&cliaddr,&clilen)<0)

{

perror("Cannot rec data"); exit(1);

}

printf("Message is received \n",buff);

**/\*Sendto function is used to echo the message from server to client side\*/**

if(sendto(sd,buff,sizeof(buff),0,(struct sockadddr\*)&cliaddr,clilen)<0)

{

perror("Cannot send data to client"); exit(1);

}

printf("Send data to UDP Client: %s",buff);

}

cloSe(sd); return 0;

}

**Client: udpclient.c**

#include<sys/types.h> #include<sys/socket.h> #include<stdio.h> #include<unistd.h> #include<string.h> #include<netinet/in.h> #include<netdb.h>

int main(int argc,char\*argv[])

{

int sd;

char buff[1024];

struct sockaddr\_in servaddr; socklen\_t len; len=sizeof(servaddr);

**/\*UDP socket is created, an Internet socket address structure is filled with wildcard address & server’s well known port\*/**

sd = socket(AF\_INET,SOCK\_DGRAM,0); if(sd<0)

{

perror("Cannot open socket"); exit(1);

}

bzero(&servaddr,len);

**/\*Socket address structure\*/** servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); servaddr.sin\_port=htons(5669);

while(1)

{

printf("Enter Input data : \n"); bzero(buff,sizeof(buff));

**/\*Reads the message from standard input\*/**

fgets(buff,sizeof (buff),stdin);

**/\*sendto is used to transmit the request message to the server\*/**

if(sendto (sd,buff,sizeof (buff),0,(struct sockaddr\*)&servaddr,len)<0)

{

perror("Cannot send data"); exit(1);

}

printf("Data sent to UDP Server:%s",buff); bzero(buff,sizeof(buff));

**/\*Receiving the echoed message from server\*/**

if(recvfrom (sd,buff,sizeof(buff),0,(struct sockaddr\*)&servaddr,&len)<0)

{

perror("Cannot receive data");

exit(1);

}

printf("Received Data from server: %s",buff);

}

close(sd); return 0;

}

**SAMPLE OUTPUT:**

**Server**:

**(Host Name:Root1)**

[root@localhost 4ita33]# vi udpserver.c [root@localhost 4ita33]# cc udpserver.c [root@localhost 4ita33]# ./a.out

Server isRunning…

Message isreceived

Send data to UDP Client: hi

Message is received

Send data to UDP Client: how are u

**Client:**

**(Host Name:Root2)**

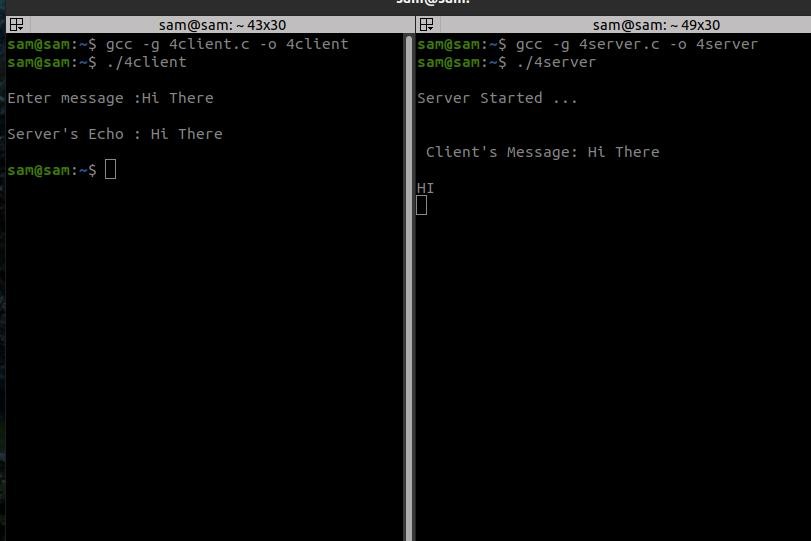
[root@localhost 4ita33]# vi udpclient.c [root@localhost 4ita33]# cc udpclient.c [root@localhost 4ita33]# ./a.out 127.0.0.1 Enter input data :

hi

Data sent to UDP Server:hi Received Data from server: hi

Enter input data :

how are u

Data sent to UDP Server:how are u Received Data from server: how are u

# ExNo:5 CONCURRENT TCP/IP DAY-TIMESERVER

**GIVEN REQUIREMENTS:**

There are two hosts, Client and Server. Both the Client and the Server exchange message

i.e. they send messages or receive message from the other. There is only a single way communication between them.

**TECHNICAL OBJECTIVE:**

To implement a half duplex application, where the Client establishes a connection with the Server. The Client can send and the server well receive messages at the same time.

**METHODOLOGY:**

**Server:**

**Client:**

* + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family to AF\_INET, sin\_addr to INADDR\_ANY, sin\_port to dynamically assigned port number.
  + Bind the local host address to socket using the bindfunction.
  + Listen on the socket for connection request from theclient.
  + Accept connection request from the Client using acceptfunction.
  + Fork the process to receive message from the client and print it on theconsole.
  + Read message from the console and send it to theclient.
  + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family toAF\_INET.
  + Get the server IP address and the Port number from theconsole.
  + Using gethostbyname function assign it to a hostent structure, and assign it to sin\_addr of the server addressstructure.
  + Request a connection from the server using the connectfunction.
  + Fork the process to receive message from the server and print it on theconsole.
  + Read message from the console and send it to theserver.

**CODING:**

**Server:dtserver.c**

#include<time.h> #include<sys/types.h>#include<sys/socket.h>#include<unistd.h> #include<stdio.h> #include<string.h> #include<netinet/in.h> #include<netdb.h>

0int main(int argc,char \*argv[])

{

int sd,ad;

char buff[1024];

struct sockaddr\_in servaddr,cliaddr;

//socklen\_t clilen=sizeof(cliaddr); time\_t t1; bzero(&servaddr,sizeof(servaddr));

**/\*Socket address structure\*/** servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); servaddr.sin\_port=htons(1507);

**/\*TCP socket is created, an Internet socket address structure is filled with wildcard address & server’s well known port\*/** sd=socket(AF\_INET,SOCK\_STREAM,0);

**/\*Bind function assigns a local protocol address to the socket\*/**

bind(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

**/\*Listen function specifies the maximum number of connections that kernel should queue for this socket\*/**

listen(sd,5);

printf("Server is running…\n");

**/\*The server to return the next completed connection from the front of the completed connection Queue calls it\*/**

ad=accept(sd,(struct sockaddr \*)NULL,NULL); while(1)

{

bzero(&buff,sizeof(buff));

**/\*Library function time returns the Coordinated Universal Time\*/**

t1=time(NULL);

**/\*Prints the converted string format\*/**

snprintf(buff,sizeof(buff),"%24s\r\n",ctime(&t1)); send(ad,buff,sizeof(buff),0);

}

**Client:dtclient.c**

#include<stdio.h> #include<sys/types.h>#include<sys/socket.h>#include<netdb.h> #include<netinet/in.h> #include<unistd.h> #include<time.h>

int main(int argc,char \*argv[])

{

int sd,ad;

char buff[1024];

struct sockaddr\_in cliaddr,servaddr; struct hostent \*h; h=gethostbyname(argv[1]); bzero(&servaddr,sizeof(servaddr));

**/\*Socket address structure\*/**

servaddr.sin\_family=AF\_INET; memcpy((char\*)&servaddr.sin\_addr.s\_addr,h->h\_addr\_list[0],h->h\_length); servaddr.sin\_port=htons(1507);

**/\*TCP socket is created, an Internet socket address structure is filled with wildcard address & server’s well known port\*/** sd=socket(AF\_INET,SOCK\_STREAM,0);

**/\*Connect establishes connection with the server using server IP address\*/** connect(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr)); recv(sd,buff,sizeof(buff),0);

printf("Day time of server is: %s\n",buff);

}

**SAMPLE OUTPUT:**

**Server**:

**(Host Name:Root1)**

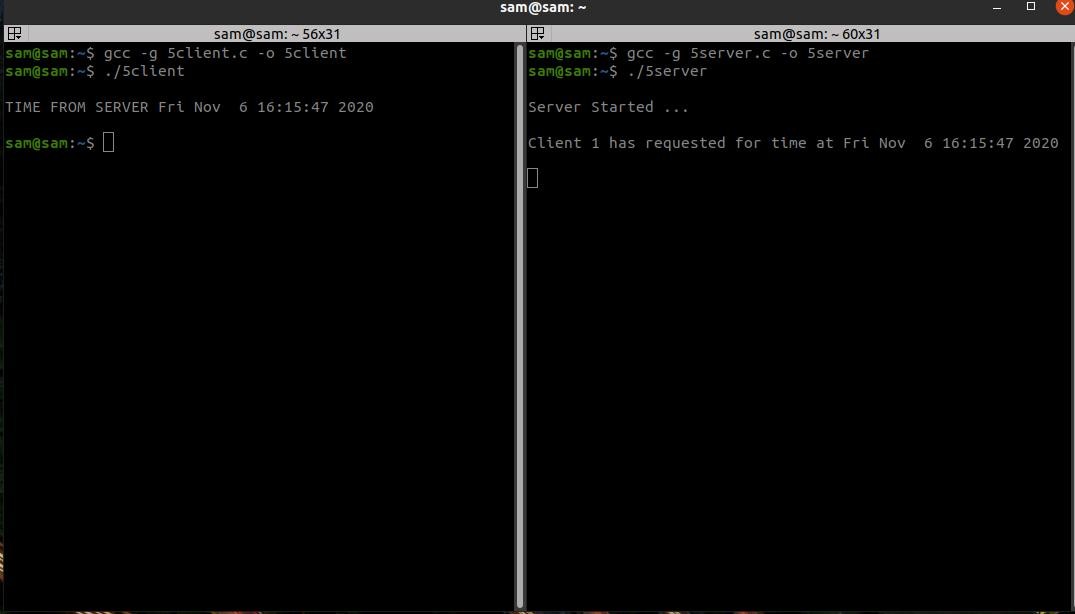
[root@localhost 4ita33]# vi dtserver.c [root@localhost 4ita33]# cc dtserver.c [root@localhost 4ita33]# ./a.out

Server is running…

**Client:**

**(Host Name:Root2)**

[root@localhost 4ita33]# vi dtclient.c [root@localhost 4ita33]# cc dtclient.c [root@localhost 4ita33]# ./a.out 127.0.0.1

Day time of server is: Sat Oct 27 18:02:21 2007

# ExNo:6 HALF DUPLEX CHAT USINGTCP/IP

**GIVEN REQUIREMENTS:**

There are two hosts, Client and Server. Both the Client and the Server exchange message

i.e. they send messages or receive message from the other. There is only a single way communication between them.

**TECHNICAL OBJECTIVE:**

To implement a half duplex application, where the Client establishes a connection with the Server. The Client can send and the server well receive messages at the same time.

**METHODOLOGY:**

**Server:**

**Client:**

* + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family to AF\_INET, sin\_addr to INADDR\_ANY, sin\_port to dynamically assigned port number.
  + Bind the local host address to socket using the bindfunction.
  + Listen on the socket for connection request from theclient.
  + Accept connection request from the Client using acceptfunction.
  + Fork the process to receive message from the client and print it on theconsole.
  + Read message from the console and send it to theclient.
  + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family toAF\_INET.
  + Get the server IP address and the Port number from theconsole.
  + Using gethostbyname function assign it to a hostent structure, and assign it to sin\_addr of the server addressstructure.
  + Request a connection from the server using the connectfunction.
  + Fork the process to receive message from the server and print it on theconsole.
  + Read message from the console and send it to theserver.

**CODING:**

**Server: hserver.c**

#include<sys/types.h> #include<stdio.h> #include<netdb.h> #include<sys/socket.h> #include<arpa/inet.h> #include<unistd.h> #include<netinet/in.h>

int main(int argc,char \*argv[])

{

int n,sd,ad;

struct sockaddr\_in servaddr,cliaddr; socklen\_t clilen,servlen;

char buff[10000],buff1[10000]; bzero(&servaddr,sizeof(servaddr));

**/\*Socket address structure\*/** servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); servaddr.sin\_port=htons(5000);

**/\*TCP socket is created, an Internet socket address structure is filled with wildcard address & server’s well known port\*/** sd=socket(AF\_INET,SOCK\_STREAM,0);

**/\*Bind function assigns a local protocol address to the socket\*/**

bind(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

**/\*Listen function specifies the maximum number of connections that kernel should queue for this socket\*/**

listen(sd,5);

printf("%s\n","server is running…");

**/\*The server to return the next completed connection from the front of the completed connection Queue calls it\*/**

ad=accept(sd,(struct sockaddr\*)&cliaddr,&clilen); while(1)

{

bzero(&buff,sizeof(buff));

**/\*Receiving the request from client\*/**

recv(ad,buff,sizeof(buff),0);

printf("Receive from the client:%s\n",buff); n=1;

while(n==1)

{

bzero(&buff1,sizeof(buff1)); printf("%s\n","Enter the input data:");

**/\*Read the message from client\*/**

fgets(buff1,10000,stdin);

**/\*Sends the message to client\*/** send(ad,buff1,strlen(buff1)+1,0); printf("%s\n","Data sent"); n=n+1;

}

}

return 0;

}

**Client: hclient.c** #include<sys/types.h> #include<sys/socket.h> #include<arpa/inet.h> #include<netinet/in.h> #include<unistd.h> #include<stdio.h> #include<netdb.h>

int main(int argc,char \*argv[])

{

int n,sd,cd;

struct sockaddr\_in servaddr,cliaddr; socklen\_t servlen,clilen;

char buff[10000],buff1[10000]; bzero(&servaddr,sizeof(servaddr));

**/\*Socket address structure\*/** servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=inet\_addr(argv[1]); servaddr.sin\_port=htons(5000);

**/\*Creating a socket, assigning IP address and port number for that socket\*/**

sd=socket(AF\_INET,SOCK\_STREAM,0);

**/\*Connect establishes connection with the server using server IP address\*/**

cd=connect(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr)); while(1)

{

bzero(&buff,sizeof(buff)); printf("%s\n","Enter the input data:");

**/\*This function is used to read from server\*/**

fgets(buff,10000,stdin);

**/\*Send the message to server\*/** send(sd,buff,strlen(buff)+1,0); printf("%s\n","Data sent");

n=1;

while(n==1)

{

bzero(&buff1,sizeof(buff1));

**/\*Receive the message from server\*/** recv(sd,buff1,sizeof(buff1),0); printf("Received from the server:%s\n",buff1); n=n+1;

}

}

return 0;

}

**SAMPLE OUTPUT:**

**Server**:

**(Host Name:Root1)**

[root@localhost 4ita33]# vi hserver.c [root@localhost 4ita33]# cc hserver.c [root@localhost 4ita33]# ./a.out

Server is running… Receive from the client:hi

Enter the input data:

how are u da .. Data sent

Receive from the client:me fine da ... Enter the input data:

**Client:**

**(Host Name:Root2)**

[root@localhost 4ita33]# vi hclient.c [root@localhost 4ita33]# cc hclient.c [root@localhost 4ita33]# ./a.out 127.0.0.1 Enter the input data:

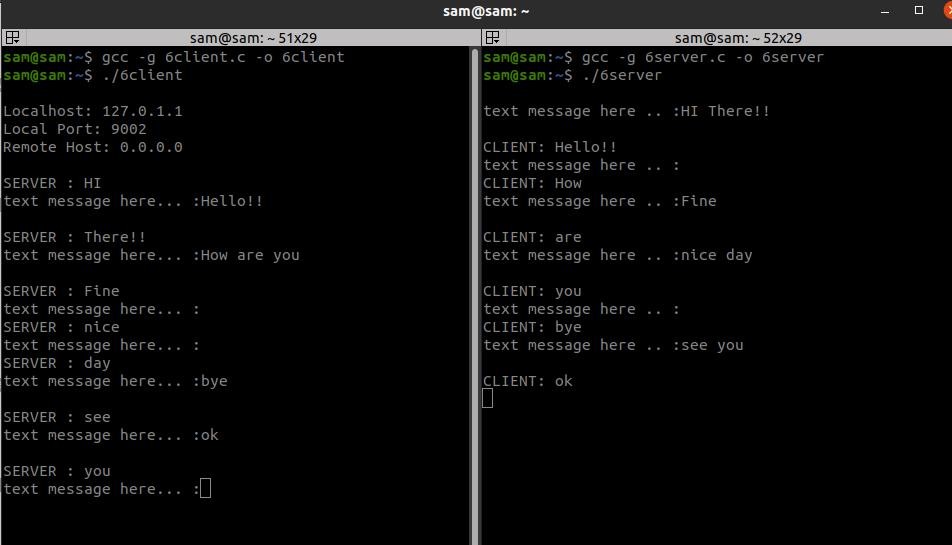
hi

Data sent:

Received from the server:how are u da ..

Enter the input data: me fine da ...

Data sent



**INFERENCE:**

Thus the chat application full duplex communication is established by sending the request from the client to the server, server gets the message and gives response to the client and prints it.

# ExNo:7 FULL DUPLEX CHAT USINGTCP/IP

**GIVEN REQUIREMENTS:**

There are two hosts, Client and Server. Both the Client and the Server exchange message

i.e. they send messages to and receive message from the other. There is a two way communication between them.

**TECHNICAL OBJECTIVE:**

To implement a full duplex application, where the Client establishes a connection with the Server. The Client and Server can send as well as receive messages at the same time. Both the Client and Server exchange messages.

**METHODOLOGY:**

**Server:**

* + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family to AF\_INET, sin\_addr to INADDR\_ANY, sin\_port to dynamically assigned port number.
  + Bind the local host address to socket using the bindfunction.
  + Listen on the socket for connection request from theclient.
  + Accept connection request from the Client using acceptfunction.
  + Fork the process to receive message from the client and print it on theconsole.
  + Read message from the console and send it to theclient.

**Client:**

* + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family toAF\_INET.
  + Get the server IP address and the Port number from theconsole.
  + Using gethostbyname function assign it to a hostent structure, and assign it to sin\_addr of the server addressstructure.
  + Request a connection from the server using the connectfunction.
  + Fork the process to receive message from the server and print it on theconsole.
  + Read message from the console and send it to theserver.

**CODING:**

**Server: fserver.c**

#include<sys/types.h> #include<sys/socket.h> #include<stdio.h> #include<unistd.h> #include<netdb.h> #include<arpa/inet.h> #include<netinet/in.h>

int main(int argc,char \*argv[])

{

int ad,sd;

struct sockaddr\_in servaddr,cliaddr; socklen\_t servlen,clilen;

char buff[1000],buff1[1000]; pid\_t cpid;

bzero(&servaddr,sizeof(servaddr));

**/\*Socket address structure\*/** servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); servaddr.sin\_port=htons(5500);

**/\*TCP socket is created, an Internet socket address structure is filled with wildcard address & server’s well known port\*/**sd=socket(AF\_INET,SOCK\_STREAM,0);

**/\*Bind function assigns a local protocol address to the socket\*/**

bind(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

**/\*Listen function specifies the maximum number of connections that kernel should queue for this socket\*/**

listen(sd,5);

printf("%s\n","Serverisrunning ");

**/\*The server to return the next completed connection from the front of the completed connection Queue calls it\*/**

ad=accept(sd,(struct sockaddr\*)&cliaddr,&clilen);

**/\*Fork system call is used to create a new process\*/**

cpid=fork();

if(cpid==0)

{

while(1)

{

bzero(&buff,sizeof(buff));

}

}

else

{

**/\*Receiving the request from client\*/**

recv(ad,buff,sizeof(buff),0);

printf("Received message from the client:%s\n",buff);

while(1)

{

bzero(&buff1,sizeof(buff1)); printf("%s\n","Enter the input data:");

**/\*Read the message from client\*/**

fgets(buff1,10000,stdin);

}

}

return 0;

}

**/\*Sends the message to client\*/** send(ad,buff1,strlen(buff1)+1,0); printf("%s\n","Data sent…");

**Client: fclient.c**

#include<sys/socket.h> #include<sys/types.h> #include<stdio.h> #include<arpa/inet.h>#include<unistd.h> #include<netdb.h> #include<netinet/in.h>

int main(int argc,char \*argv[])

{

int sd,cd;

struct sockaddr\_in servaddr,cliaddr; socklen\_t servlen,clilen;

char buff[1000],buff1[1000]; pid\_t cpid;

bzero(&servaddr,sizeof(servaddr)); servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=inet\_addr(argv[1]); servaddr.sin\_port=htons(5500);

**/\*Creating a socket, assigning IP address and port number for that socket\*/**

sd=socket(AF\_INET,SOCK\_STREAM,0);

**/\*Connect establishes connection with the server using server IP address\*/**

cd=connect(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

**/\*Fork is used to create a new process\*/**

cpid=fork(); if(cpid==0)

{

while(1)

{

bzero(&buff,sizeof(buff)); printf("%s\n","Enter the input data:");

**/\*This function is used to read from server\*/**

fgets(buff,10000,stdin);

**/\*Send the message to server\*/** send(sd,buff,strlen(buff)+1,0); printf("%s\n","Data sent…");

}

}

else

{

while(1)

{

bzero(&buff1,sizeof(buff1));

**/\*Receive the message from server\*/**

recv(sd,buff1,sizeof(buff1),0);

printf("Received message from the server:%s\n",buff1);

}

}

return 0;

}

**SAMPLE OUTPUT:**

**Server**:

**(Host Name:Root1)**

[root@localhost 4ita33]# vi fserver.c [root@localhost 4ita33]# cc fserver.c [root@localhost 4ita33]# ./a.out

Server is running.......

Enter the input data:

Received message from the client:hi how are u

Data sent…

Enter the input data:

Received message from the client:i am fine

**Client:**

**(Host Name:Root2)**

[root@localhost 4ita33]# vi fclient.c [root@localhost 4ita33]# cc fclient.c [root@localhost 4ita33]# ./a.out 127.0.0.1 Enter the input data:

hi

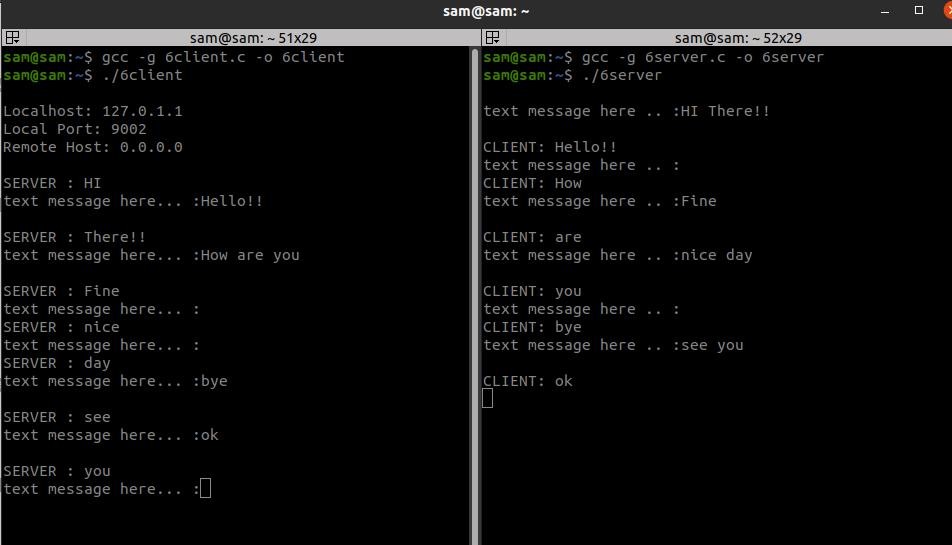
Data sent…

Enter the input data:

Received message from the server:how are u i am fine

Data sent…

Enter the input data:



**INFERENCE:**

Thus the chat application full duplex communication is established by sending the request from the client to the server, server gets the message and gives response to the client and prints it.

# ExNo:8 IMPLEMENTATION OF FILE TRANSFERPROTOCOL

**GIVEN REQUIREMENTS:**

There are two hosts, Client and Server. The Client sends the name of the file it needs from the Server and the Server sends the contents of the file to the Client, where it is stored in a file.

**TECHNICAL OBJECTIVE:**

To implement FTP application, where the Client on establishing a connection with the Server sends the name of the file it wishes to access remotely. The Server then sends the contents of the file to the Client, where it is stored.

**METHODOLOGY:**

**Server:**

* Include the necessary headerfiles.
* Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
* Initialize server address to 0 using the bzerofunction.
* Assign the sin\_family to AF\_INET, sin\_addr to INADDR\_ANY, sin\_port to dynamically assigned port number.
* Bind the local host address to socket using the bindfunction.
* Listen on the socket for connection request from theclient.
* Accept connection request from the Client using acceptfunction.
* Within an infinite loop, receive the file name from theClient.
* Open the file, read the file contents to a buffer and send the buffer to theClient.

**Client**:

* + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_STREAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family toAF\_INET.
  + Get the server IP address and the Port number from theconsole.
  + Using gethostbyname function assign it to a hostent structure, and assign it to sin\_addr of the server addressstructure.
  + Within an infinite loop, send the name of the file to be viewed to theServer.
  + Receive the file contents, store it in a file and print it on theconsole.

**CODING:**

**Server: ftps.c** #include<sys/types.h> #include<sys/socket.h> #include<sys/stat.h> #include<arpa/inet.h> #include<netinet/in.h> #include<netdb.h> #include<unistd.h> #include<stdio.h> #include<string.h>

int main(int argc,char \*argv[])

{

int sd,ad,size;

struct sockaddr\_in servaddr,cliaddr; socklen\_t clilen; clilen=sizeof(cliaddr);

struct stat x;

char buff[100],file[10000]; FILE \*fp;

bzero(&servaddr,sizeof(servaddr)); servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); servaddr.sin\_port=htons(1500);

sd=socket(AF\_INET,SOCK\_STREAM,0); bind(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr)); listen(sd,5);

printf("%s\n","ServerIsRunning ");

ad=accept(sd,(struct sockaddr\*)&cliaddr,&clilen); while(1)

{

bzero(buff,sizeof(buff)); bzero(file,sizeof(file)); recv(ad,buff,sizeof(buff),0); fp=fopen(buff,"r"); stat(buff,&x); size=x.st\_size; fread(file,sizeof(file),1,fp); send(ad,file,sizeof(file),0);

}

}

**Client: ftpc.c** #include<sys/types.h> #include<sys/socket.h> #include<netinet/in.h> #include<arpa/inet.h> #include<netdb.h> #include<stdio.h> #include<unistd.h>

int main(int argc,char \*argv[])

{

int sd,cd;

struct sockaddr\_in servaddr,cliaddr; socklen\_t clilen;

char buff[100],file[10000]; struct hostent \*h;

h=gethostbyname(argv[1]); bzero(&servaddr,sizeof(servaddr)); servaddr.sin\_family=h->h\_addrtype;

memcpy((char \*)&servaddr.sin\_addr.s\_addr,h->h\_addr\_list[0],h->h\_length); servaddr.sin\_port=htons(1500);

sd=socket(AF\_INET,SOCK\_STREAM,0); cd=connect(sd,(struct sockaddr\*)&servaddr,sizeof(servaddr));

while(1)

{

printf("%s\n","Enter the File Name :"); scanf("%s",buff); send(sd,buff,strlen(buff)+1,0); printf("%s\n","File Output :"); recv(sd,file,sizeof(file),0); printf("%s",file);

}

return 0;

}

**SAMPLE OUTPUT:**

**Server**:

**(Host Name:Root1)**

[root@localhost 4ita33]# vi ftps.c [root@localhost 4ita33]# cc ftps.c [root@localhost 4ita33]# ./a.out

Server is Running… FILE REACHED

File output : this is my network lab

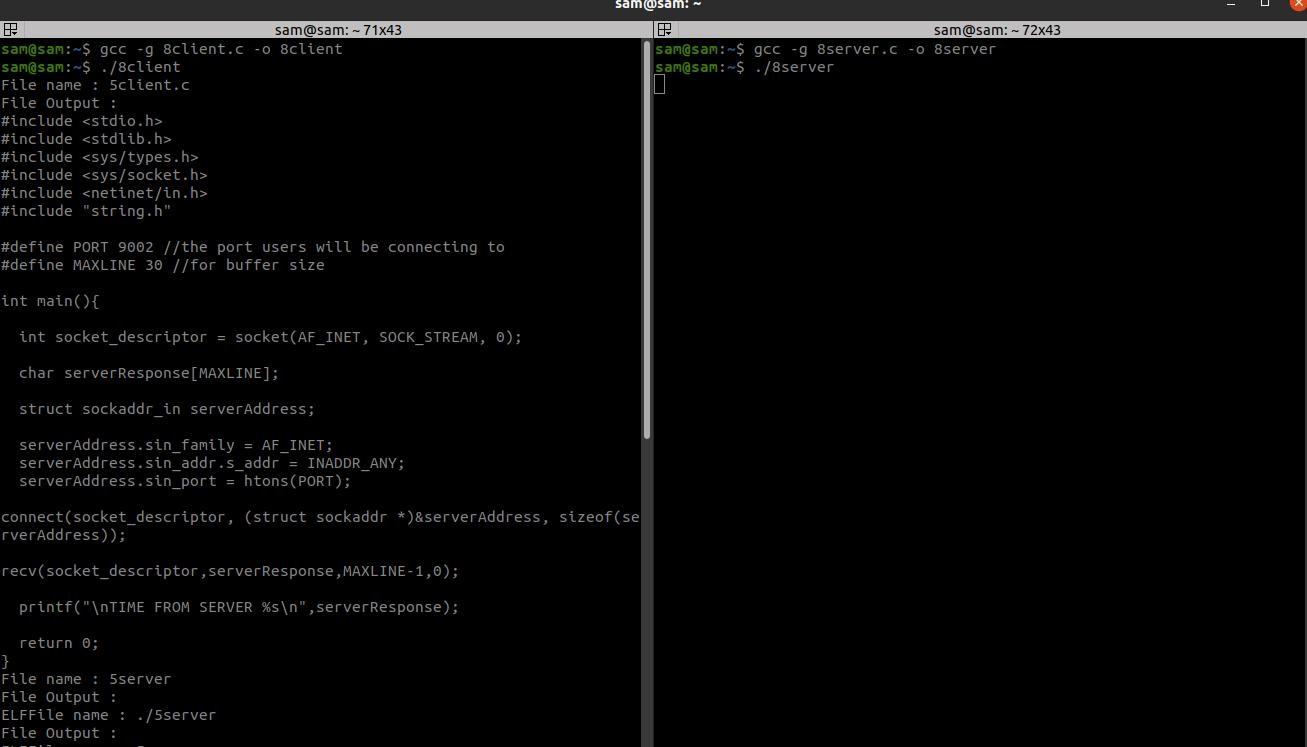
**Client:**

**(Host Name:Root2)**

[root@localhost 4ita33]# vi ftpc.c [root@localhost 4ita33]# cc ftpc.c [root@localhost 4ita33]# ./a.out Enter the filename:

ita.txt

Sending the file content Data sent..…



**INFERENCE:**

Thus the FTP client-server communication is established and data is transferred between the client and server machines.

# ExNo:9 REMOTE COMMAND EXECUTION USINGUDP

**GIVEN REQUIREMENTS:**

There are two hosts, Client and Server. The Client sends a command to the Server, which executes the command and sends the result back to the Client.

**TECHNICAL OBJECTIVE:**

Remote Command execution is implemented through this program using which Client is able to execute commands at the Server. Here, the Client sends the command to the Server for remote execution. The Server executes the command and the send result of the execution back to the Client.

**METHODOLOGY:**

**Server**:

* Include the necessary headerfiles.
* Create a socket using socket function with family AF\_INET, type asSOCK\_DGRAM.
* Initialize server address to 0 using the bzerofunction.
* Assign the sin\_family to AF\_INET, sin\_addr to INADDR\_ANY, sin\_port to dynamically assigned port number.
* Bind the local host using the bind() systemcall.
* Within an infinite loop, receive the command to be executed from theclient.
* Append text “> temp.txt” to thecommand.
* Execute the command using the “system()” systemcall.
* Send the result of execution to the Client using a filebuffer.

**Client:**

* + Include the necessary headerfiles.
  + Create a socket using socket function with family AF\_INET, type asSOCK\_DGRAM.
  + Initialize server address to 0 using the bzerofunction.
  + Assign the sin\_family toAF\_INET.
  + Get the server IP address and the Port number from theconsole.
  + Using gethostbyname() function assign it to a hostent structure, and assign it to sin\_addr of the server addressstructure.
  + Obtain the command to be executed in the server from theuser.
  + Send the command to theserver.
  + Receive the output from the server and print it on theconsole.

**CODING:**

**Server: udpremoteserver.c** #include<sys/types.h> #include<sys/socket.h> #include<stdio.h> #include<netdb.h> #include<netinet/in.h> #include<string.h>

#include<sys/stat.h> #include<arpa/inet.h> #include<unistd.h>

int main(int argc,char\* argv[])

{

int sd,size;

char buff[1024],file[10000];

struct sockaddr\_in cliaddr,servaddr; FILE \*fp;

struct stat x; socklen\_t clilen; clilen=sizeof(cliaddr);

bzero(&servaddr,sizeof(servaddr)); servaddr.sin\_family=AF\_INET; servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY); servaddr.sin\_port=htons(9976); sd=socket(AF\_INET,SOCK\_DGRAM,0); if(sd<0)

{

printf("Socket CReation Error");

}

bind(sd,(struct sockaddr \*)&servaddr,sizeof(servaddr)); while(1)

{

bzero(buff,sizeof(buff));

recvfrom(sd,buff,sizeof(buff),0,(struct sockaddr \*)&cliaddr,&clilen); strcat(buff,">file1");

system(buff); fp=fopen("file1","r");

stat("file1",&x); size=x.st\_size; fread(file,size,1,fp);

sendto(sd,file,sizeof(file),0,(struct sockaddr \*)&cliaddr,sizeof(cliaddr)); printf("Data Sent to UDPCLIENT %s",buff);

}

close(sd); return 0; }

**Client: udpremoteclient.c**

#include<sys/types.h> #include<sys/socket.h> #include<stdio.h> #include<unistd.h> #include<netdb.h> #include<netinet/in.h> #include<string.h> #include<arpa/inet.h> #include<sys/stat.h>

int main(int argc,char\* argv[])

{

int sd;

char buff[1024],file[10000];

struct sockaddr\_in cliaddr,servaddr; struct hostent \*h;

socklen\_t servlen; servlen=sizeof(servaddr);

h=gethostbyname(argv[1]); bzero(&servaddr,sizeof(servaddr)); servaddr.sin\_family=h->h\_addrtype;

memcpy((char \*)&servaddr.sin\_addr,h->h\_addr\_list[0],h->h\_length); servaddr.sin\_port=htons(9976); sd=socket(AF\_INET,SOCK\_DGRAM,0);

if(sd<0)

{

printf("Socket CReation Error");

}

bind(sd,(struct sockaddr \*)&servaddr,sizeof(servaddr)); while(1)

{

printf("\nEnter the command to be executed"); fgets(buff,1024,stdin);

sendto(sd,buff,strlen(buff)+1,0,(struct sockaddr \*)&servaddr,sizeof(servaddr)); printf("\nData Sent");

recvfrom(sd,file,strlen(file)+1,0,(struct sockaddr \*)&servaddr,&servlen); printf("Recieved From UDPSERVER %s",file);

}

return 0;

}

**SAMPLE OUTPUT:**

**Server**:

**(Host Name:Root1)**

[root@localhost 4ita33]# vi udpremoteserver.c [root@localhost 4ita33]# cc udpremoteserver.c [root@localhost 4ita33]# ./a.out

Server is running............

VIM(1) VIM(1)

NAME

vim - Vi IMproved, a programmers text editor SYNOPSIS

vim [options] [file ..] vim [options] -

vim [options] -t tag

vim [options] -q [errorfile} ex

review rview renbjc

**Client:**

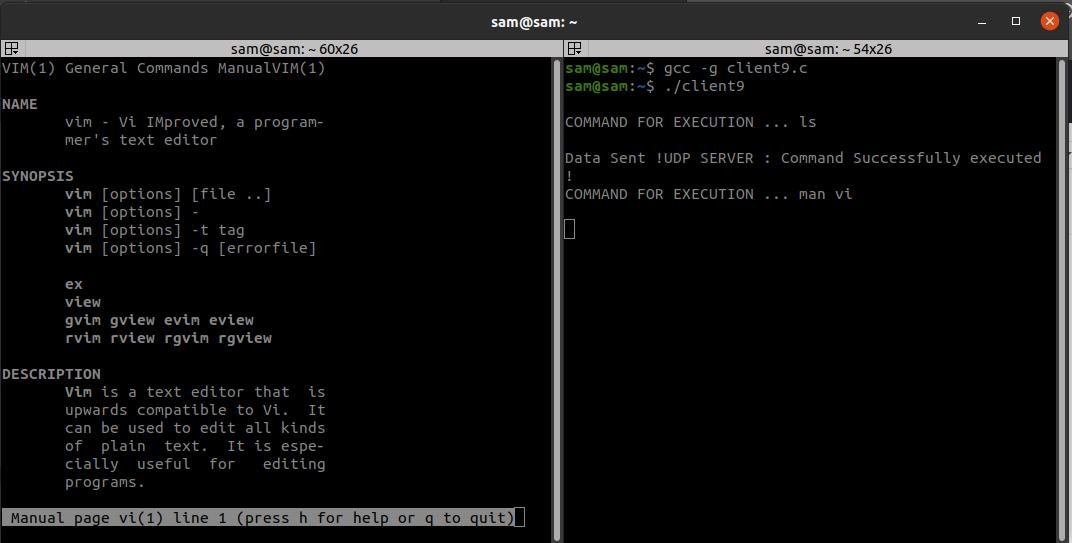
**(Host Name:Root2)**

[root@localhost 4ita33]# vi udpremoteclient.c [root@localhost 4ita33]# cc udpremoteclient.c

[root@localhost 4ita33]# ./a.out 127.0.0.1 Enter command:

man vi

Command sent to server Enter command:



**INFERENCE:**

Thus the Remote Command Execution between the client and server is implemented.

# ExNo: 10 ARP IMPLEMENTATION USINGUDP

**GIVEN REQUIREMENTS:**

There is a single host. The IP address of any Client in the network is given as input and the corresponding hardware address is got as the output.

**TECHNICAL OBJECTIVE**:

Address Resolution Protocol (ARP) is implemented through this program. The IP address of any Client is given as the input. The ARP cache is looked up for the corresponding hardware address. This is returned as the output. Before compiling that Client ispinged.

**METHODOLOGY:**

* + - Include the necessary headerfiles.
    - Create a socket using socket function with family AF\_INET, type asSOCK\_DGRAM.
    - Declare structures arpreq ( as NULL structure, if required) andsockaddr\_in.
    - Initialize server address to 0 using the bzerofunction.
    - Assign the sin\_family to AF\_INET and sin\_addr usinginet\_aton().
    - Using the object of arpreq structure assign the name of the Network Device to the data member arp\_dev like,arp\_dev=”eth0”.
    - Ping the requiredClient.
    - Using the ioctl() we get the ARP cache entry for the given IPaddress.
    - The output of the ioctl() function is stored in the sa\_data[0] datamember of the arp\_ha structure which is in turn a data member of structurearpreq.
    - Print the hardware address of the given IP address on the outputconsole.

**CODING:**

ARP:arp.c

#include<sys/types.h> #include<sys/socket.h> #include<net/if\_arp.h> #include<sys/ioctl.h> #include<stdio.h> #include<unistd.h> #include<netinet/in.h> #include<arpa/inet.h>

int main(int argc,char \*argv[])

{

struct sockaddr\_in sin={0}; struct arpreq myarp={{0}}; unsigned char \*ptr;

int sd; sin.sin\_family=AF\_INET;

if(inet\_aton(argv[1],&sin.sin\_addr)==0)

{

printf("Ip address Entered '%s' is not valid \n",argv[1]); exit(0);

}

memcpy(&myarp.arp\_pa,&sin,sizeof(myarp.arp\_pa)); strcpy(myarp.arp\_dev,"eth0"); sd=socket(AF\_INET,SOCK\_DGRAM,0); if(ioctl(sd,SIOCGARP,&myarp)==1)

{

printf("No Entry in ARP Cache for '%s'",argv[1]); exit(0);

}

ptr=&myarp.arp\_ha.sa\_data[0]; printf("MAC Address For '%s' : ",argv[1]);

printf("%X:%X:%X:%X:%X:%X\n",\*ptr,\*(ptr+1),\*(ptr+2),\*(ptr+3),\*(ptr+4),\*(ptr+5),\*(ptr+5)); return 0;

}

**SAMPLE OUTPUT:**

**Host: arp.c**

**(HostName:Root1)**

[root@localhost 4ita33]# vi arp.c [root@localhost 4ita33]# ping172.16.29.51

PING 172.16.29.51 (172.16.29.51) 56(84) bytes of data.

64 bytes from172.16.29.51: icmp\_seq=1 ttl=64 time=1.19 ms 64 bytes from 172.16.29.51: icmp\_seq=2 ttl=64 time=0.817 ms

-- 172.16.29.51ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 999ms rtt min/avg/max/mdev = 0.817/1.005/1.193/0.188 ms [root@localhost 4ita33]# cc arp.c

[root@localhost 4ita33]# ./a.out 172.16.29.51

Hardware Address is: 172.16.29.51:

The MAC address is:0:8:5C:5D:47:50:

**INFERENCE**:

Thus the ARP implementation is developed to gets the MAC address of the remote machine’s IP address from ARP cache and prints it.

## Ex No:11

**Aim:**

**STUDY OF IPV6 ADDRESSING AND SUBNETTING**

To study IPV6 address terminology, IPV6 addreess format, Types of Addresses, assigning IP addresses to devices andsubnetting.

## 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 toitself.

## 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 IPv6 unicast address prefix. Other terms for “link” are subnet and network segment.

## Network

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

## Neighbors

Nodes connected to the same link. Neighbors in IPv6 have special significance because of IPv6 Neighbor 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 IPv6 layer 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

MTUThemaximumtransmissionunit(MTU)—thenumberofbytesinthelargestIPv6

packet—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 link MTU 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 thepath.

## IPv6 Address Format

Whereas IPv4 addresses use a dotted-decimal format, where each byte ranges from 0 to 255.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 stringnotation.

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; and ABC: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 zeroes were being omitted from each part. An unspecified address is represented as ::, since it contains allzeroes.

## 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 multicastaddress.

* With a unicast, one packet is sent to onedestination;
* With a multicast, one packet is sent to all members of the multicastgroup;
* 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 closet 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-nearestaddress.

## Multicast

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

* The first 8 bits are set toFF.
* The next 4 bits are the lifetime of the address: 0 is permanent and 1 istemporary.
* The next 4 bits indicate the scope of the multicast address (how far the packet can travel): 1isforanode,2isforalink,5isforthesite,8isfortheorganization,andEisglobal

(the Internet).

## Unicast

The following types of addresses are unicast IPv6 addresses:

* Global unicastaddresses
* Link-localaddresses
* Site-localaddresses
* Unique localaddresses
* Specialaddresses
* Transitionaddresses

## 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 IPv6 Internet. 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 operating IP-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 65,536 subnets or multiple levels of addressing hierarchy and an efficient routing infrastructure. With 16 bits of subnetting flexibility, a global unicast prefix assigned to an organization site is equivalent to a public IPv4 Class A address prefix (assuming that the last octet is used for identifying nodes on subnets). The routing 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 IPv6 is 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 for Neighbor Discovery processes. Site-local addresses are used between nodes communicating with other nodes in the sameorganization.

## 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 use is 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 IPv6 network with no router, link-local addresses are used to communicate between hosts on the link. IPv6 link-local addresses are similar to IPv4 link-local addresses defined in RFC 3927 that use the 169.254.0.0/16 prefix. The use of IPv4 link-local addresses is known as Automatic Private IP Addressing (APIPA) in Windows Vista, Windows Server 2008, Windows Server 2003, and Windows XP. The scope of a link local address is the local link. A link-local address is required for some Neighbor 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 IPv4 private 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 without 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 global addresses. 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 summarize able routing infrastructure. After the Subnet ID 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-localaddresses.

## 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 address is 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 4 to represent the samelink.

## Unique Local Addresses

Site-local addresses provide a private addressing alternative to global addresses for intranet 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 prefix FC00::/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 L flag 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 Global ID, 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 a 48-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. Unique local addresses have a global scope, but their reach ability is defined by routing topology and filtering policies at Internet boundaries. Organizations will not advertise their unique local address prefixes outside of their organizations or create DNS entries with unique local addresses in the Internet DNS. Organizations can easily create filtering policies at their Internet boundaries to prevent all unique local-addressed traffic from being forwarded. Because they have a global scope, unique local addresses do not need a zone ID. The global address and unique local address share the same structure beyond the first 48 bits of the address. In both addresses, the 16-bit Subnet ID field identifies a subnet within an organization. Because of this, you can create a subnetted routing infrastructure that is used for both local and global addresses. For example, a specific subnet of your organization can be assigned both the global prefix 2001:DB8:4D1C:221A::/64 and the local prefix FD0E:2D:BA9:221A::/64, where the subnet is identified for both types of prefixes by the Subnet ID value of 221A. Although the subnet identifier is the same for both prefixes, routes for both prefixes must still be propagated throughout the routing infrastructure so that addresses based on both prefixes arereachable.

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 never assigned 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 IPv6 over 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 Windows Server2008.

## 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 IPv4 address as a 128-bit IPv6address.

## 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 assigned to 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 multicastaddresses:
* 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 addressassigned
* The multicast addresses of joinedgroups

## 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 IPaddresses.

## Advantage of Subnetting

* + Subnetting allows us to break a single large network in smaller networks. Small networks are easy tomanage.
  + Subnetting reduces network traffic by allowing only the broadcast traffic which is relevant to thesubnet.
  + By reducing unnecessary traffic, Subnetting improves overall performance of the network.
  + By blocking a subnet’ traffic in subnet, Subnetting increases security of the network.
  + Subnetting reduces the requirement of IPrange.

## Disadvantage of Subnetting

* + Different subnets need an intermediate device known as router to communicate with eachother.
  + Since each subnet uses its own network address and broadcast address, more subnets mean more wastage of IPaddresses.
  + Subnetting adds complexity in network. An experienced network administrator is required to manage the subnettednetwork.

## Class A Subnets

In Class A, only the first octet is used as Network identifier and 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 from Host part are borrowed and the subnet mask is changedaccordingly.

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) 16384 Networks and (216-2) 65534 Hosts. Class B IP Addresses can be subnetted the same way as Class A addresses, 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.

**CONCLUSION:** Thus the IPV6 address terminology, IPV6 address format, Types of Addresses, assigning IP addresses to devices and sub netting were studied successfully

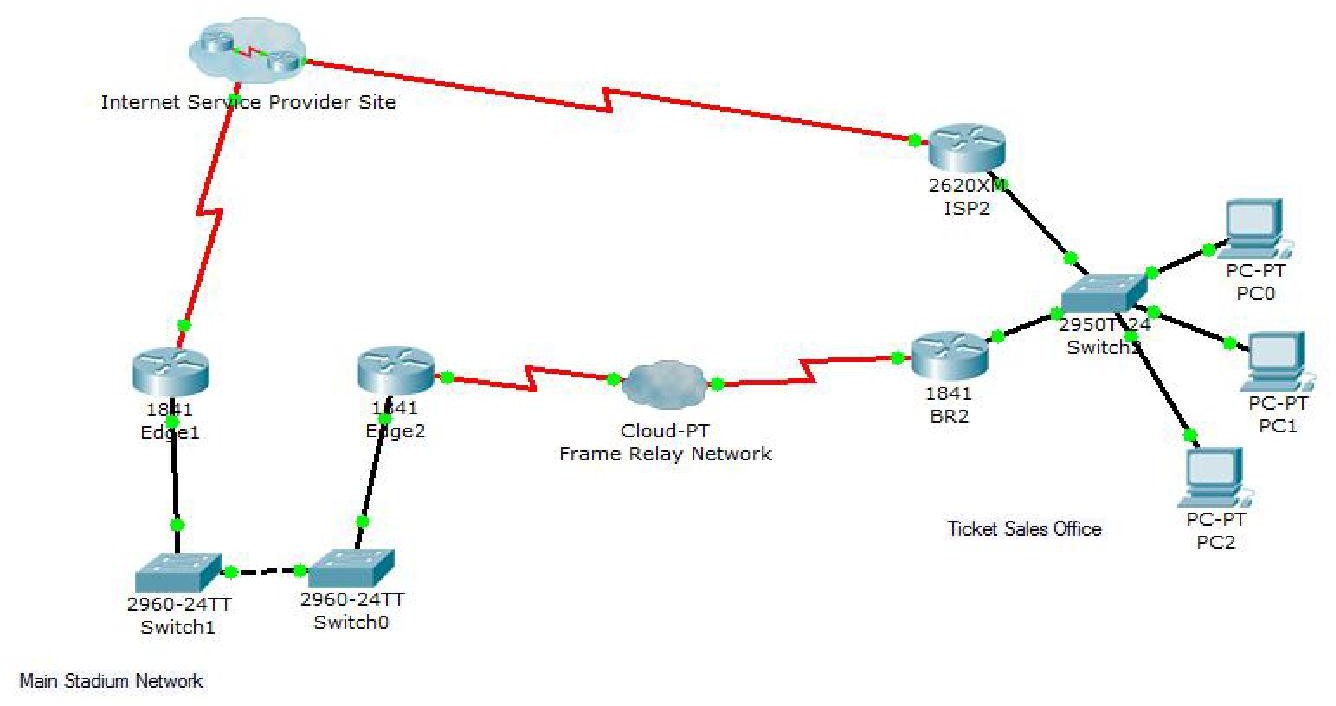
## Ex No:12

**Aim:**

**NETWORK ADDRESS TRANSLATION**

To examine NAT processes as traffic traverses a NAT border router**.**

## Topology Diagram



**Background / Preparation**

In this activity, you will use Packet Tracer Simulation mode to examine the contents of the IP header as traffic crosses the NAT border router.

**Step 1:** Prepare the network for Simulation mode.

Verify that the network is ready to send and receive traffic. All the link lights should be green. If some link lights are still amber, you can switch between Simulation and Realtime mode several times to force the lights to turn green faster. Switch to Simulation mode before going to the next step.

**Step 2**: Send an HTTP request from an inside host to an outside web server.

* Click CustomerPC.
* Click the Desktop tab and then WebBrowser.
* In the URL field, type the web address for the ISP server ([www.ispserver.com](http://www.ispserver.com/)).
* Make sure that you are in Simulation mode, and then clickGo.
* In the event list, notice that Customer PC queues a DNS request and sends out an ARP request. You can view the contents of the ARP request by either clicking on the packet in the topology or clicking on the packet color under Info in the Event Listwindow.
* In the PDU Information at Device: Customer PC window, which IP address is Customer PC attempting to find a MAC address for? And Which MAC address is placed inside the ARPreply?
* Then builds another packet. What is the protocol for this new packet? If you click Outbound PDU Details for this packet, you can see the details of theprotocol.
* In the Event List window, click Capture/Forward twice. Click the packet at the [www.customerserver.com](http://www.customerserver.com/)server.
* Then click the Outbound PDU Details tab. Scroll down to the bottom to see the Application Layer data.
* In the Event List window, click Capture/Forward 10 times until Customer PC formulates an HTTP requestpacket.
* Customer PC finally has enough information to request a web page from the ISPserver.
* In the Event List window, click Capture/Forward three times. Click the packet at Customer Router to examine thecontents.
* Customer Router is a NAT border router. What is the inside local address and the inside global address for CustomerPC?
* In the Event List window, click Capture/Forward seven times until the HTTP reply reaches Customer Router.
* Examine the contents of the HTTP reply and notice that the inside local and global addresses have changed again as the packet is forwarded on to CustomerPC.

**Step 3:** Send an HTTP request from an outside host to an inside web server.

* Customer Server provides web services to the public (outside addresses) through the domain name[www.customerserver.com](http://www.customerserver.com/).
* Follow a process similar to Step 2 to observe an HTTP request on ISPWorkstation.

1. Click ISP Workstation. Click the Desktop tab, and then Web Browser. In the URL field, type the Customer Server web address ([www.customerserver.com](http://www.customerserver.com/)). Make sure that you are in Simulation mode, and then clickGo.
2. You can either click Auto Capture/Play or Capture/Forward to step through each stage of the process. The same ARP and DNS processes occur before the ISP Workstation can formulate an HTTPrequest.
3. When the HTTP request arrives at Customer Router, check the packet contents. What is the inside local address? What is the inside globaladdress?

## CONCLUSION:

Thus the NAT processes as traffic traverses a NAT border router have been examined successfully.

## Ex No:13

**Aim:**

**IMPLEMENTATION OF VPN**

To design and implement SSL VPN in Host to Host Tunnel using OpenSSL package.

## Lab Environment

We need to use OpenSSL package in this lab. The package includes the header files, libraries, and commands. The package was not installed in our pre-built VM image, but it can be easily installed using the

following command.

$ apt-get source openssl

After downloading the source package, unpack the .tar.gz file, and then follow the standard steps ("./config", "make", "make install") to build and install the OpenSSL package. Read the README and INSTALL files in the package for detailed instructions**.**

## Create a Host-to-Host Tunnel using TUN/TAP

The enabling technology for the TLS/SSL VPNs is TUN/TAP, which is now widely implemented in modern operating systems. TUN and TAP are virtual network kernel drivers; they implement network device that are supported entirely in software. TAP (as in network tap) simulates an Ethernet device and it operates with layer-2 packets such as Ethernet frames; TUN (as in network TUNnel) simulates a network layer device and it operates with layer-3 packets such as IP packets. With TUN/TAP, we can create virtual network interfaces. A user-space program is usually attached to the TUN/TAP virtual network interface. Packets sent by an operating system via a TUN/TAP network interface are delivered to the user-space program. On the other hand, packets sent by the program via a TUN/TAP network interface are injected into the operating system netwrok stack; to the operating system, it appears that the packets come from an external source through the virtual network interface.When a program is attached to a TUN/TAP interface, the IP packets that the computer sends to thisinterface will be piped into the program; on the other hand, the IP packets that the program sends to theinterface will be piped into the computer, as if they came from the outside through this virtual networkinterface. The program can use the standard read() and write() system calls to receive packets fromor send packets to the virtualinterface.

When you compile the sample code from the tutorial, if you see error messages regarding linux/if.h, try to change

"<linux/if.h>" to "<net/if.h>" in the include statement.

For the convenience of this lab, we have modified the Brini’s simpletun program, and linked the code in the lab web page. Students can simply download this C program and run the following command to

compile it. We will use simpletun to create tunnels in this lab:

$ gcc -o simpletun simpletun.c

## Creating Host-to-Host Tunnel:

The following procedure shows how to create a host-to-host tunnel using the simpletun program. The simpletun program can run as both a client and a server. When it is running with the -s flag, it acts as a server; when it is running with the -c flag, it acts as aclient.

* 1. **Launch two virtual machines**. For this task, we will launch these two VMs on the same host machine. The IP addresses for the two machines are 192.168.10.5, and 192.168.20.5, respectively (you can choose any IP addresses you like). See the configuration in Figure1.

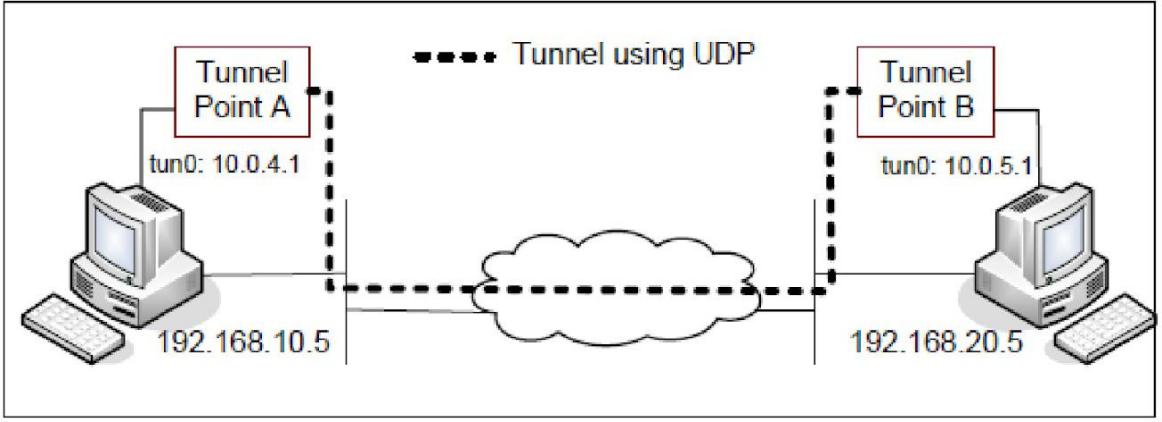


Fig.1.Host To Host Tunnel

* 1. **Tunnel Point** A: we use Tunnel Point A as the server side of the tunnel. Point A is on machine 192.168.10.5 (see Figure 1). It should be noted that the client/server concept is only meaningful when establishing the connection between the two ends. Once the tunnel is established, there is no difference between client and server; they are simply two ends of a tunnel. We run the following command (the -d flag asks the program to print out the debugging information):

On Machine 192.168.10.5:

# ./simpletun -i tun0 -s -d

After the above step, your computer now have multiple network interface, one is its own Ethernet card interface, and the other is the virtual network interface called tun0. This new interface is not yet configured, so we need to configure it by assigning an IP address. We use the IP address from the reserved IP address space(10.0.0.0/8).

It should be noted that the above command will block and wait for connections, so, we need to find another window to configure the tun0 interface. Run the following commands (the first command will assign an IP address to the interface "tun0", and the second command will bring up theinterface):

On Machine 192.168.10.5:

# ip addr add 10.0.4.1/24 dev tun0 # ifconfig tun0 up

* 1. **Tunnel Point B**: we use Tunnel Point B as the client side of the tunnel. Point B is on machine 192.168.20.5 (see Figure 1). We run the following command on this machine (The first command will connect to the server program running on 192.168.10.5, which is the machine that runs the

Tunnel Point A. This command will block as well, so we need to find another window for the second and the third commands):

On Machine 192.168.20.5:

# ./simpletun -i tun0 -c 192.168.10.5 -d # ip addr add 10.0.5.1/24 dev tun0

# ifconfig tun0 up

* 1. **Routing Path**: After the above two steps, the tunnel will be established. Before we can use the tunnel, we need to set up the routing path on both machines to direct the intended outgoing traffic through the tunnel. The following routing table entry directs all the packets to the 10.0.5.0/24 network (10.0.4.0/24 network for the second command) through the interface tun0, from where the packet will be hauled through the tunnel.

On Machine192.168.10.5:

# route add -net 10.0.5.0 netmask 255.255.255.0 dev tun0 On Machine 192.168.20.5:

# route add -net 10.0.4.0 netmask 255.255.255.0 dev tun0

* 1. **Using Tunnel:** Now we can access 10.0.5.1 from 192.168.10.5 (and similarly access 10.0.4.1 from 192.168.20.5). We can test the tunnel using ping and ssh (note: do not forget to start the ssh server first):

On Machine 192.168.10.5:

$ ping 10.0.5.1

$ ssh 10.0.5.1

On Machine 192.168.20.5:

$ ping 10.0.4.1

$ ssh 10.0.4.1

## RESULT:

Thus SSL VPN in Host to Host Tunnel using OpenSSL package has been implemented successfully.

## Ex No:14

**Aim:**

**COMMUNICATION USING HDLC**

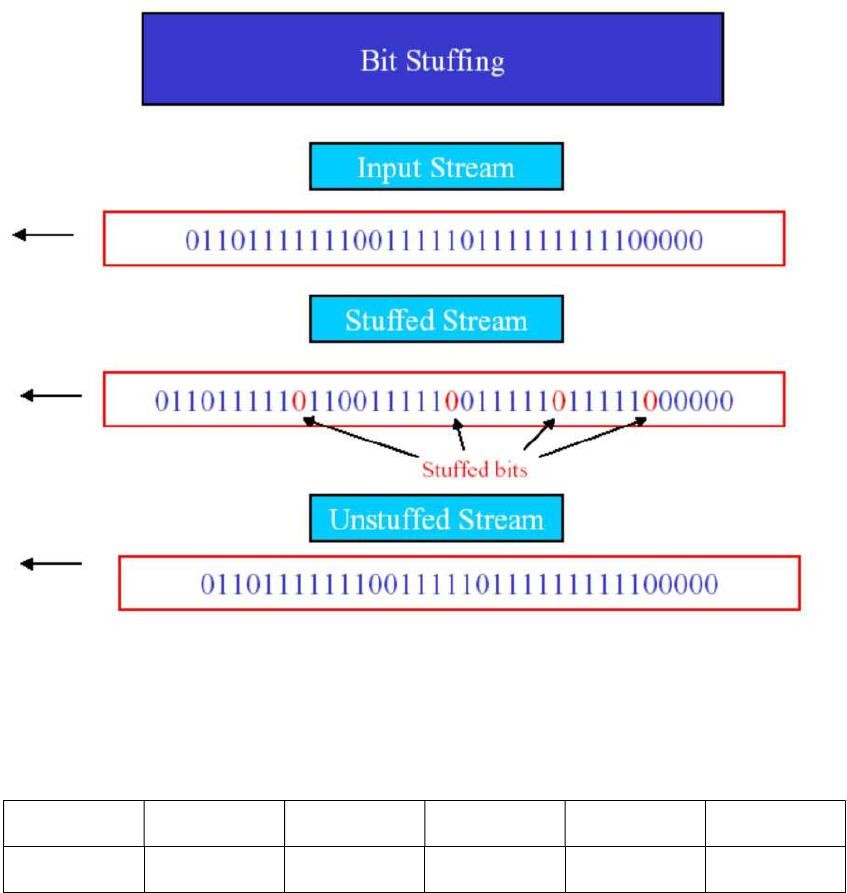
To write a program for a HLDC frame to perform the bit stuffing & de-stuffing for the given data

1. Bit stuffing & de-stuffing using HDLC

Framing involves identifying the beginning and end of a block of information within a

digital

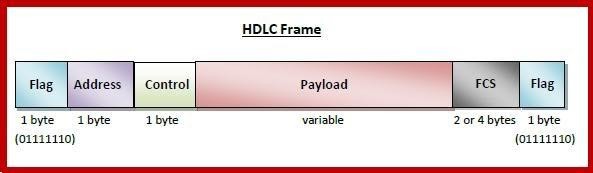
stream. In asynchronous data transmission, since transmissions do not occur at regular intervals, the receiver resynchronizes at the start of each eight bit character by the use of a start bit that precedes and a stop bit that ends each character .In synchronous data transmission bits are transmitted at regular intervals and the receiver has the circuitry that recovers and tracks the frequency and bit transitions of the received data Framing may involve delineating the boundaries between frames that are of fixed length or it may involve delineating between frames that are of variable length. Variable length frames need more information to delineate. The methods available include: Special characters to identify beginning and end of frame Special bit patterns-“flags” to identify the beginning and end of frames and character counts Bit Stuffing: Flag based synchronization was developed to transfer an arbitrary number of bits within a frame. The figure below shows the structure of an HDLC frame. The beginning and end of an HDLC frame is indicated by the presence of an eight bit flag. The flag in HDLC consists of the byte 01111110 that is HEX 7E . Bit stuffing prevents the occurrence of the flag inside the frame. The transmitter examines the contents of the frame and inserts an extra 0 after each instance of five consecutive 1s. The transmitter then attaches the flag at the beginning and end of the resulting bit stuffed frame. The receiver looks for five consecutive 1s in the received sequence. Five 1s followed by a 0 indicate that the 0 is a stuffing bit and so the bit is removed. Five consecutive 1s followed by 10 indicate a flag. Five 1 s followed by 11 indicate an error. The example below show bit stuffing inHDLC



Bit stuffing in HDLC

## HDLC frame

HDLC frame is given below



## HDLC Frame

The high level data link control protocol is a bit oriented protocol which uses bit stuffing for data transparency. HDLC frame is as shown.

8 8 8 >0 168

0111 1110 Address Control Data Checksum 0111 1110

The Control field is used for sequence numbers and acknowledgements and other purposes. The Data field may contain arbitrary information. The Checksum field is a minor variation of the CRC code using CRC-CCITT as the generator. Frames are delimited by 0111 1110 and this sequence is not allowed in the data field.

HDLC is a bit - oriented protocol where each frame contains up to six fields. The structure varies according to the type of frame. The fields of a HDLC frame are −

Flag − It is an 8-bit sequence that marks the beginning and the end of the frame. The bit pattern of the flag is01111110.

Address − It contains the address of the receiver. If the frame is sent by the primary station, it contains the address(es) of the secondary station(s). If it is sent by the secondary station, it contains the address of the primary station. The address field may be from 1 byte to several bytes. Control − It is 1 or 2 bytes containing flow and error control information.

Payload − This carries the data from the network layer. Its length may vary from one network to another.

FCS − It is a 2 byte or 4 bytes frame check sequence for error detection. The standard code used is CRC (cyclic redundancy code)

## Algorithm for Bit Stuffing:

1. Input datasequence
2. Add start of frame to outputsequence
3. for every bit ininput
   1. Append bit to outputsequence
   2. Is abit1? Yes:

Incrementcount

If count is 5, append 0 to output sequence and resetcount No: Set count to0

1. Add stop of frame bits to outputsequence

## Algorithm for Bit Destuffing:

1. Input datasequence
2. Remove start of frame to outputsequence
3. for every bit ininput
   1. Append bit to outputsequence
   2. Is abit1? Yes:

Increment count

If count is 5, remove 0 from input sequence and reset count No:Set count to 0 4. Remove stop of frame bits to output sequence.

/\* Program to simulate bit stuffing & destuffing where the flag byte is 01111110 \*/ # include <stdio.h>

# include <conio.h> #include<string.h> void main() {

char ch, array[50]={"01111110"},recd\_array[50]; int counter=0,i=8,j,k;

clrscr();

printf("Enter the original data stream for bit stuffing : \n");while((ch=getche())!='\r')

{ if(ch=='1')

++counter; else counter=0; array[i++]=ch;

if(counter==5) /\* If 5 ones are encountered append azero\*/ { array[i++]='0';

counter=0;

}}

array[i]=’\0’; strcat(array,"01111110"); array[i+8]=’\0’;

printf("\nThe stuffed data stream is : %s \n",array);

/\* Destuffing \*/ counter=0;

printf("\nThe destuffed data stream is : \n"); for (j=8,k=0;j<(strlen(array)-8);++j)

{ if(array[j]=='1')

++counter; else

counter=0; recd\_array[k++]=array[j];

if (counter==5) /\* If five ones appear, delete the following zero \*/

{ ++j; counter=0;

}

}

recd\_array[k]=’\0’;

printf(“ the destuffed data stream is %s”,recd\_array); getch();

}

## OUTPUT:

Enter the original data stream for bit stuffing :

011011111111110010

The stuffed data stream is :011111100110111110111110001001111110 The destuffed data stream is :011011111111110010

## RESULT:

Thus a program for a HLDC frame to perform the bit stuffing & de-stuffing for the given data has been implemented successfully.

## Ex No:15

**Aim:**

**COMMUNICATION USING PPP**

To configure the Point To Point Protocol over Ethernet

## Point To Point Protocol over Ethernet

PPPoE is a protocol which encapsulates PPP frames in Ethernet frames. PPPoE is used because PPP has the option to authenticate a user using a username and a password via any of the PAP or CHAP authentication methods. PPoE on Ethernet feature enhances Point-to-Point Protocol over Ethernet (PPPoE) by being able to add direct connection to actual Ethernet interfaces.The fact that Ethernet is a shared medium; it can be used to serve multiple users to open PPPsessions.

## Simulation Files:

**configuring\_pppoe \_init.pkt** – contains the initial topology. All needed physical connections are configured.

**configuring\_pppoe\_final.pkt** – this is the final configuration with what you should have configured.

You can use this file to compare your configuration.

Before you go ahead and start configuring, in case you are required, use this information:

* use as DHCP pool for PPPoE clients the range 10.0.0.10 – 10.0.0.100 name the DHCP pool DHCP\_PPPoE
* use the username and password: pppoe/cisco
* name the VPDN(virtual private dial-up networking) groupGROUP

## Task 1 requirements:

* Enable PPPoE on the LANinterface.
* Create a Virtual-Templateinterface
* Enable IP on the Virtual-Template interface without assigning a specific IP. Use the LAN interface.
* Use DHCP to assign an IP address to any remote peer connecting to thisinterface.
* Use CHAPauthentication.
* EnableVPDN.
* Associates a VPDN group to a customer or VPDNprofile.
* Create an accept dial-in VPDNgroup.
* Specifies the VPDN group that will be used to establish PPPoEsessions.
* Specifies which virtual template will be used to clone virtual accessinterfaces.
* Create the DHCPpool.
* Create the username that will be used to connect viaPPPoE.

## Task 1 verification:

* + Connect to PC1 and from Desktop tab, use PPPoEdialer.
  + Use the username/password: pppoe/cisco and a pop-up window should tell you that you are connected.
  + From Desktop tab, use Command Prompt and issue a ping to towards the server. This should besuccessful.
  + From Desktop tab, use Command Prompt and issue the command ‘ipconfig’. Confirm that you were assigned an IP address from the range 10.0.0.10 –10.0.0.100.
  + Once you’re logged in, check the logs on the router and you should see something similar tothis:

%LINK-5-CHANGED: Interface Virtual-Access1.1, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface Virtual-Access1.1, changed state to

up

## Task 1 hints

* + Use the command ‘pppoe enable’ on WAN interface to enablePPPoE.
  + Use the command ‘interface Virtual-Template 1’ to configure the Virtual-Template interface.
  + Use the command ‘ip unnumbered FastEthernet0/1’ to assign an IP on the Virtual- Templateinterface.
  + Use the command ‘peer default ip address pool DHCP\_PPPoE’ to assign to an IP address to any remote peer connecting to interfaceVirtual-Template1.
  + Use the command ‘ppp authentication chap’ to specify the PPPauthentication.
  + Use the command ‘vpdnenable’.
  + Use the command ‘vpdn-group GROUP’ to associate a VPDN group to acustomer.
  + Use the command‘accept-dialin’.
  + Use the command ‘protocol pppoe’ to specify that the VPDN group will be used to establish PPPoEsessions.
  + Use the command ‘virtual-template 1’ to specify which interface will be used to clone the virtual accessinterface.
  + Use the command ‘ip local pool DHCP\_PPPoE 10.0.0.10 10.0.0.100’ to create the DHCP pool.
  + Use the command ‘username pppoe password 0 cisco’ to create the username used for PPPoE.

## RESULT:

Thus the Point To Point Protocol over Ethernet has been implemented successfully.