EX NO: 1 STUDY OF HEADER FILES WITH RESPECT TO SOCKET PROGRAMMING
Date:

1. **Stdio.h:**
   Standard Input/Output Header, provides I/O functions, like printf and scanf, for reading and writing data in C and C++ programs.
2. **unistd.h:**
   Provides essential functions (e.g., close) for low-level I/O operations and process control in C/C++ programs.
3. **string.h:**
   is a C standard library header file providing functions for string manipulation, including copying, comparing, and searching.
4. **stdlib.h:**
   This header file contains the utility functions such as string conversion routines, memory allocation routines, random number generator, etc. (abort, exit, rand, atoi etc.)
5. **sys/types.h:**
   Defines the data type of socket address structure in unsigned long.(clock_t, size_t, dev_t etc.)
6. **sys/socket.h:**
   The socket functions can be defined as taking pointers to the generic socket address structure called sockaddr. (SO_REUSEADDR, SO_ERROR, SO_ACCEPTCONN etc.)
7. **netinet/in.h:**
   Defines the IPv4 socket address structure commonly called Internet socket address structure called sockaddr_in. (IPPROTO_IP, IPPROTO_ICMP, IPPROTO_TCP etc.)
8. **netdb.h:**
   Defines the structure hostent for using the system call gethostbyname to get the network host entry. (HOST_NOT_FOUND, NO_DATA, NO_RECOVERY etc.)
9. **time.h:**
   Has structures and functions to get the system date and time and to perform time manipulation functions. We use the function ctime(), that is defined in this header file , to calculate the current date and time.
10. **sys/stat.h:**
    Contains the structure stat to test a descriptor to see if it is of a specified type. Also it is used to display file or file system status.stat() updates any time related fields.when copying from 1 file to another

11. **sys/ioctl.h:**
    Macros and defines used in specifying an ioctl request are located in this header file. We use the function ioctl() that is defined in this header file. ioctl() function is used to perform ARP cache operations
12. **pcap.h:**
    Has function definitions that are required for packet capturing. Some of the functions are pcap_lookupdev(),pcap_open_live() and pcap_loop(). pcap_lookupdev() is used to initialize the network device. The device to be sniffed is opened using the pcap_open_live(). Pcap_loop() determines the number of packets to be sniffed.
13. **net/if_arp.h:**
    Contains the definitions for Address Resolution Protocol. We use this to manipulate the ARP request structure and its data members arp_pa,arp_dev and arp_ha. The arp_ha structure's data member sa_data[ ] has the hardware address
14. **errno.h:**
    It sets an error number when an error and that error can be displayed using perror function. It has symbolic error names. The error number is never set to zero by any library function
15. **arpa/inet.h:**
    This is used to convert internet addresses between ASCII strings and network byte ordered binary values (values that are stored in socket address structures). It is used for inet_aton, inet_addr, inet_ntoa functions).

EX NO: 2 **STUDY OF BASIC FUNCTIONS OF SOCKET**
Date: **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);
eg: sd=socket(AF_INET,SOCK_STREAM,0);
```

DESCRIPTION:

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

TYPES:

1. SOCK_STREAM: ¬ Provides sequenced , reliable, two-way , connection based byte streams. ¬ An out-of-band data transmission mechanism, may be supported.

2. SOCK_DGRAM: ¬ Supports datagram (connectionless, unreliable messages of a fixed maximum length)

3. SOCK_SEQPACKET: ¬ Provides a sequenced , reliable, two-way connection based data transmission path for datagrams of fixed maximum length.

4. SOCK_RAW: ¬ Provides raw network protocol access.

5. SOCK_RDM: ¬ Provides a reliable datagram layer that doesn't guarantee ordering.

6. SOCK_PACKET: ¬ Obsolete and shouldn't be used in new programs.

2. Man connect:

NAME:

connect – initiate a connection on a socket.

SYNOPSIS:

```
#include<sys/types.h>
#include<sys/socket.h>
int connect(int sockfd,const (struct sockaddr*)&serv_addr,socklen_t addrlen);
```

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

DESCRIPTION:

- The file descriptor sockfd must refer to a socket.
- If the socket is of type SOCK_DGRAM then the serv_addr address is the address to which datagrams are sent by default and the only addr from which datagrams are received.
- If the socket is of type SOCK_STREAM or SOCK_SEQPACKET , this call attempts to make a connection to another socket.

RETURN VALUE:

- If the connection or binding succeeds, zero is returned.
- On error , -1 is returned , and error number is set appropriately

ERRORS:

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

3.Man accept:

NAME:

accept/reject job is sent to a destination.

SYNOPSIS:

accept destination(s)

reject[-t] [-h server] [-r reason] destination(s)

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

DESCRIPTION:

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

4. Man send:

NAME:

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

SYNOPSIS:

```
#include<sys/types.h>
#include<sys/socket.h>
ssize_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 connected state.
- The only difference between send and write is the presence of flags.
- The parameter is the file descriptor of the sending socket.

5.man recv:

NAME:

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

SYNOPSIS:

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

- The recvfrom and recvmsg calls are used to receive messages from a socket, and may be used to recv data on a socket whether or not it is connection oriented.
- If from is not NULL, and the underlying protocol provides the src addr , this src addr is filled in.
- The recv call is normally used only on a connection socket and is identical to recvfrom with a NULL from parameter

6.Man read:

NAME:

read, readonly, return

7.Man write:

NAME:

write- send a message to another user.

SYNOPSIS:

write user[ttyname]

DESCRIPTION:

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

8. ifconfig:

NAME:

ifconfig- configure a network interface.

SYNOPSIS:

ifconfig[interface]

ifconfig interface[aftype] options | address……

DESCRIPTION:

- ifconfig is used to configure the kernel resident network interfaces
- It is used at boot time to setup interfaces as necessary.
- After that, it is usually only needed when debugging or when system tuning is needed.
- If no arguments are given, ifconfig displays the status of the currently active interfaces.

9.Man bind

SYNOPSIS:

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

10. Man htons/ Man htonl

NAME:

htonl, htons, ntohl, ntohs - convert values between host and network byte order.

SYNOPSIS:

```
#include<netinet/in.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint32_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION:

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

11. Man gethostname:

NAME:

gethostname, sethostname- get/set host name.

SYNOPSIS:

```
#include<unistd.h>
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 current processor.
- The gethostname() returns a NULL terminated hostname(set earlier by sethostname()) in the array name that has a length of len bytes.
- In case the NULL terminated then hostname does not fit ,no error is returned, but the hostname is truncated
- It is unspecified whether the truncated hostname will be NULL terminated

12. Man gethostbyname:

NAME:

gethostbyname, gethostbyaddr, sethostent, endhostent, herror, hstr – error –get network host entry.

SYNOPSIS:

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

DESCRIPTION:

- The gethostbyname() returns a structure of type hostent for the given hostname.
- Name->hostname or IPV4/IPV6 with dot notation.
- gethostbyaddr()- struct of type hostent / host address length
- Address types- AF_INET, AF_INET6.
- sethostent() – stay open is true(1).
- TCP socket connection should be open during queries.
- Server queries for UDP datagrams.
- endhostent()- ends the use of TCP connection.
- Members of hostent structure:
    a) h_name
    b) h_aliases
    c) h_addrtype
    d) h_length
    e) h_addr-list
    f) h_addr.

RESULT:

Thus the basic functions used for Socket Programming was studied successfully

EX NO: 3 SIMPLE TCP/IP CLIENT SERVER COMMUNICATION
Date:

Aim:

To design and implement a simple TCP/IP client-server communication system using c programming.

Given Requirements:

1. Create a client-server communication system using TCP/IP.
2. The client and server should be able to exchange messages.
3. Implement error handling to ensure robust communication.
4. Develop both client and server applications using a programming language of your choice.

Technical Objective:

To implement an UDP Echo Client-Server application, where the Client on establishing a Connection with the Server, sends a string to the Server. The Server reads the String, prints it and echoes it back to the Client.

Methodology:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to a dynamically assigned port number.
- Bind the local host address to socket using the bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the client using accept function.
- Within an infinite loop, using the recv function receive message from the client and print it on the console.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and port number from the console.

- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Request a connection from the server using the connect function.
- Within an infinite loop, read message from the console and send the message to the server using the send function.

Code:

Server: tcpserver.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<arpa/inet.h>
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
int main(int asrgc,char*argv[])
{
   int bd,sd,ad;
   char buff[1024];
   struct sockaddr_in cliaddr,servaddr;
   socklen_t clilen;
   clilen=sizeof(cliaddr);
   bzero(&servaddr,sizeof(servaddr));
   /*Socket address structure*/
   servaddr.sin_family=AF_INET;
   servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
   servaddr.sin_port=htons(1999);
   /*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*/
   bd=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*)&cliaddr,&clilen);
    while(1)
    {
    bzero(&buff,sizeof(buff));
    /*Receiving the request message from the client*/
    recv(ad,buff,sizeof(buff),0);
    printf("Message received is %s\n",buff);
    }
}
```

Client: tcpclient.c

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<unistd.h>
#include<netinet/in.h>
#include<netdb.h>
#include<arpa/inet.h>
int main(int argc,char * argv[])
{
int cd,sd,ad;
char buff[1024];
struct sockaddr_in cliaddr,servaddr;
struct hostent *h;
/*This function looks up a hostname and it returns a pointer to a hostent
 structure that contains all the IPV4 address*/
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(1999);
 /*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)
{
printf("Enter the message: \n");
```

```
 /*Reads the message from standard input*/
 fgets(buff,100,stdin);
 /*Send function is used on client side to send data given by user on client
 side to the server*/
send(sd,buff,sizeof(buff)+1,0);
printf("\n Data Sent ");
//recv(sd,buff,strlen(buff)+1,0);
printf("%s",buff);
}
}
```

SAMPLE OUTPUT:

Client:

```
user@user-virtual-machine:~/CN_LAB/3$ gcc clint.c -o clin
user@user-virtual-machine:~/CN_LAB/3$ ./clin 127.0.0.1
Enter the message:
hi

 Data Sent hi
Enter the message:
hello srm ist trichy

 Data Sent hello srm ist trichy
Enter the message:
```

Server:

```
user@user-virtual-machine:~/CN_LAB/3$ gcc server.c -o serve
user@user-virtual-machine:~/CN_LAB/3$ ./serve
Server is running….
Message received is hi

Message received is
Message received is hello srm ist trichy

Message received is
```

Result:

Thus, a program to perform simple communication between client and server using TCP/IP was implemented.

EX NO: 4 UDP ECHO CLIENT SERVER COMMUNICATION
Date:

Aim:

To design and implement a UDP (User Datagram Protocol) Echo Client-Server Communication system using c programming.

Given Requirements:

1. A computer or a virtual environment with a suitable programming environment (e.g., Python).
2. Basic understanding of networking concepts and the UDP protocol.
3. Text editor or integrated development environment (IDE).

Methodology:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to SERVER_PORT, a macro defined port number.
- Bind the local host address to socket using the bind function.
- Within an infinite loop, receive message from the client using recvfrom function, print it on the console and send (echo) the message back to the client using sendto function.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address from the console.

- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Within an infinite loop, read message from the console and send the message to the server using the sendto function.
- Receive the echo message using the recvfrom function and print it on the console

Code:

Server: udpserver.c

```
#include <sys/socket.h>
#include <stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netinet/in.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);
   sd = socket(AF_INET, SOCK_DGRAM, 0);
   if (sd < 0) {
      perror("Cannot open Socket");
      exit(1);
   }
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
   servaddr.sin_port = htons(5669);
   if (bind(sd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
      perror("Error in binding the port");
      exit(1);
   }
   printf("Server is Running...\n");
```

```
    while (1) {
        bzero(buff, sizeof(buff));

        if (recvfrom(sd, buff, sizeof(buff), 0, (struct sockaddr*)&cliaddr, &clilen) < 0) {
            perror("Cannot receive data");
            exit(1);
        }
        printf("Message is received: %s\n", buff);

        if (sendto(sd, buff, strlen(buff), 0, (struct sockaddr*)&cliaddr, clilen) < 0) {
            perror("Cannot send data to client");
            exit(1);
        }
        printf("Sent data to UDP Client: %s\n", 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>
#include<stdlib.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:

```
user@user-virtual-machine:~/CN_LAB/4$ gcc udpserver.c -o udpserver
user@user-virtual-machine:~/CN_LAB/4$ ./udpserver
Server is Running...
Message is received: hi

Sent data to UDP Client: hi
```

Client:

```
user@user-virtual-machine:~/CN_LAB/4$ gcc udpclient.c -o client
user@user-virtual-machine:~/CN_LAB/4$ ./client 127.0.0.1
Enter Input data :
hi
Data sent to UDP Server:hi
Received Data from server: hi
Enter Input data :
```

Result:

Thus, the UDP ECHO client server communication is established by sending the message from the client to the server and server prints it and echoes the message back to the client.

EX NO: 5 CONCURRENT TCP/IP DAY-TIME SERVER
Date:

Aim:

To design and implement a Concurrent TCP/IP Day-Time Server, which allows multiple clients to connect and retrieve the current date and time using c programming.

Given Requirements:

1. Develop a server application that listens on a specific port.
2. Accept multiple client connections simultaneously.
3. Respond to client requests by providing the current date and time.
4. Ensure error handling for client-server communication.
5. Implement concurrent server architecture to handle multiple clients concurrently.

Technical Objectives:

- Learn socket programming in a Unix-like environment.
- Understand the basics of network protocols and communication.
- Implement concurrent server architecture using multi-threading or multiprocessing.
- Gain experience in error handling and robust server design.

Methodology:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to statically assigned port number.
- Bind the local host address to socket using the bind function.

- Within a for loop, accept connection request from the client using accept function.
- Use the fork system call to spawn the processes.
- Calculate the current date and time using the ctime() function. Change the format so that it is appropriate for human readable form and send the date and time to the client using the write function.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Request a connection from the server using the connect function.
- Within an infinite loop, receive the date and time from the server using the read function and print the date and time on the console.

Code:

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>
#include<stdlib.h>
int main(int argc,char *argv[])
{
   int sd,ad;
   char buff[1024];
   struct sockaddr_in servaddr,cliaddr;
```

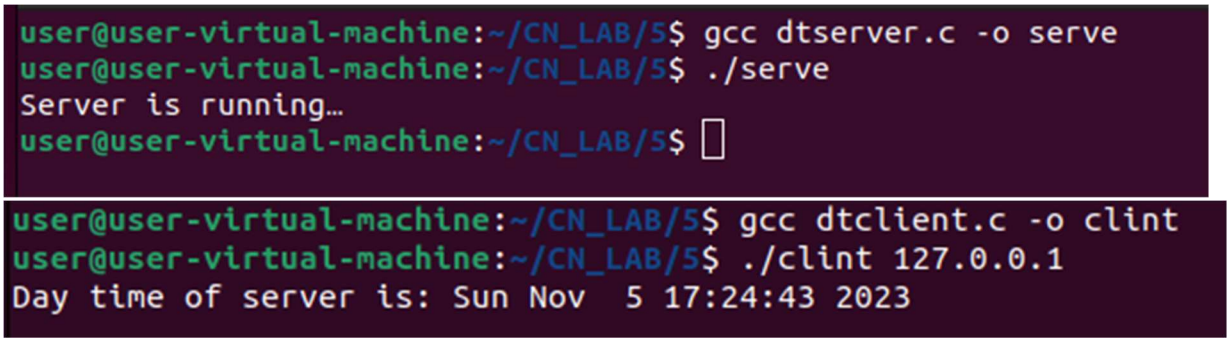
```
    //socklen_t clilen=sizeof(cliaddr);
    time_t t1;
    bzero(&servaddr,sizeof(servaddr));
    /*Socket address structure*/
    servaddr.sin_family=AF_INET;
    servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
    servaddr.sin_port=htons(1507);
    /*TCP socket is created, an Internet socket address 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<string.h>
#include<time.h>
#include<stdlib.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:



Result:

Thus the concurrent daytime client- server communication is established by sending the request message from the client to the concurrent server and the server sends its time to all the clients and displays it.

EX NO: 6 HALF DUPLEX CHAT USING TCP/IP
Date:

Aim:

The aim of this project is to design and implement a half-duplex chat application using TCP/IP.

Given Requirements:

- A computer with network connectivity
- Programming environment (e.g., Python, C++)
- Basic understanding of networking concepts
- Prior knowledge of socket programming

Technical Objectives:

- Designing a client-server architecture for the chat application.
- Implementing socket programming to establish connections.
- Allowing multiple clients to connect to the server.
- Enabling the server to relay messages between clients.
- Ensuring half-duplex communication, where one user can send a message while the other listens.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- 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 bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the Client using accept function.
- Fork the process to receive message from the client and print it on the console.
- Read message from the console and send it to the client.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Request a connection from the server using the connect function.
- Fork the process to receive message from the server and print it on the console.
- Read message from the console and send it to the server

Code:

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>
#include<string.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>
#include<string.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;
}
```

Output :

Server:

```
user@user-virtual-machine:~/CN_LAB/6$ gcc htserver.c -o server
user@user-virtual-machine:~/CN_LAB/6$ ./server
server is running…
Receive from the client:hi

Enter the input data:
hello client srm ist trichy
Data sent
Receive from the client:hello RA21110__________

Enter the input data:
```

Client:

```
user@user-virtual-machine:~/CN_LAB/6$ gcc htclinet.c -o clint
user@user-virtual-machine:~/CN_LAB/6$ ./clint 127.0.0.1
Enter the input data:
hi
Data sent
Received from the server:hello client srm ist trichy

Enter the input data:
hello RA21110__________
Data sent
```

Result:

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.

EX NO: 7 FULL DUPLEX CHAT USING TCP/IP
Date:

Aim:

The aim of this project is to design and implement a full-duplex chat application using the TCP/IP protocol, allowing users to exchange messages in real-time while maintaining a stable and efficient communication channel.

Given Requirements:

1. Develop a chat application that supports full-duplex communication.
2. Implement the application using the TCP/IP protocol.
3. Ensure that the application is user-friendly and provides a seamless chatting experience.
4. The application should be capable of running on multiple platforms.
5. Implement error handling and robustness in the system to ensure reliable communication

Methodology:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- 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 bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the Client using accept function.
- Fork the process to receive message from the client and print it on the console.
- Read message from the console and send it to the client.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Request a connection from the server using the connect function.
- Fork the process to receive message from the server and print it on the console.
- Read message from the console and send it to the server

Code:

Server: server.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<unistd.h>
#include<netdb.h>
#include<string.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","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);
   /*Fork system call is used to create a new process*/
   cpid=fork();
   if(cpid==0)
   {
      while(1)
```

```
        {
            bzero(&buff,sizeof(buff));
            /*Receiving the request from client*/
            recv(ad,buff,sizeof(buff),0);
            printf("Received message from the client:%s\n",buff);
        }
    }
    else
    {
        while(1)
        {
            bzero(&buff1,sizeof(buff1));
            printf("%s\n","Enter the input data:");
            /*Read the message from client*/
            fgets(buff1, sizeof(buff1), stdin);
            /*Sends the message to client*/
            send(ad,buff1,strlen(buff1)+1,0);
            printf("%s\n","Data sent…");
        }
    }
    return 0;
}
```

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>
#include<string.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, sizeof(buff), 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;
}
```

Source Output:

Server:

```
user@user-virtual-machine:~/CN_LAB/7$ gcc server.c -o fserv
user@user-virtual-machine:~/CN_LAB/7$ ./fserv
Server is running.......
Enter the input data:
Received message from the client:hi

Received message from the client:hi

hi
Data sent…
Enter the input data:
goodbye
Data sent…
Enter the input data:
Received message from the client:ok

ok
Data sent…
Enter the input data:
```

Client:

```
user@user-virtual-machine:~/CN_LAB/7$ gcc clint.c -o fclint
user@user-virtual-machine:~/CN_LAB/7$ ./fclint 127.0.0.1
Enter the input data:
hi
Data sent…
Enter the input data:
hi
Data sent…
Enter the input data:
Received message from the server:hi

Received message from the server:goodbye

ok
Data sent…
Enter the input data:
Received message from the server:ok
```

Result:

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.

EX NO: 8 **IMPLEMENTATION OF FILE TRANSFER**
Date: **PROTOCOL**

Aim:

To design the implementation of a File Transfer Protocol (FTP) system using c programming.

Given Requirements:

1. Develop a client-server architecture.
2. Implement secure and encrypted data transmission.
3. Support both upload and download operations.
4. Ensure user authentication for secure access.
5. Manage directory structures for clients and server.
6. Handle error and exception scenarios gracefully.
7. Provide a user-friendly interface for the FTP application.

Methodology:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- 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 bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the Client using accept function.
- Within an infinite loop, receive the file name from the Client.
- Open the file, read the file contents to a buffer and send the buffer to the Client.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Within an infinite loop, send the name of the file to be viewed to the Server.
- Receive the file contents, store it in a file and print it on the console

Code:

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", "Server Is Running....");
```

```
    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, size, 1, fp);
        send(ad, file, size, 0);
        fclose(fp); // Close the file after reading
    }
    return 0;
}
```

Source Output:

Client:

```
user@user-virtual-machine:~/CN_LAB/8$ gcc clint.c -o clin
user@user-virtual-machine:~/CN_LAB/8$
user@user-virtual-machine:~/CN_LAB/8$ ./clin 127.0.0.1
Enter the File Name:
hi.txt
File Output:
Srm ist trichy campus




Enter the File Name:
```

Server:

```
user@user-virtual-machine:~/CN_LAB/8$ gcc server.c -o serve
user@user-virtual-machine:~/CN_LAB/8$ ./serve
Server Is Running....
Segmentation fault (core dumped)
user@user-virtual-machine:~/CN_LAB/8$
```

Result:

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

EX NO: 9 REMOTE COMMAND EXECUTION USING UDP
Date:

Aim:

To design and implement a system for remote command execution using the User Datagram Protocol (UDP).

GIVEN REQUIREMENTS:

1. A networked environment where multiple systems can communicate.
2. Basic knowledge of network protocols and system administration.
3. A development environment, preferably with programming languages like Python or C++.
4. A target system for remote command execution.
5. A remote system to initiate and control the execution of commands.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- 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() system call.
- Within an infinite loop, receive the command to be executed from the client.
- Append text "> temp.txt" to the command.
- Execute the command using the "system()" system call.
- Send the result of execution to the Client using a file buffer.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname() function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
- Obtain the command to be executed in the server from the user.
- Send the command to the server.

- Receive the output from the server and print it on the console

Code:

Server: serve.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 <stdlib.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:clint.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;
}
```

Source Output:

Server:

```
user@user-virtual-machine:~/CN_LAB/9$ gcc udpremoteserver.c -o serv
user@user-virtual-machine:~/CN_LAB/9$ ./serv
sh: 1: vim: not found
Data Sent to UDPCLIENT  vim -vi
>file1Data Sent to UDPCLIENT  man vi
^C
```

```
VIM(1)                    General Commands Manual                    VIM(1)

NAME
       vim - Vi IMproved, a programmer's text editor

SYNOPSIS
       vim [options] [file ..]
       vim [options] -
       vim [options] -t tag
       vim [options] -q [errorfile]

       ex
       view
       gvim gview evim eview
       rvim rview rgvim rgview

DESCRIPTION
       Vim  is a text editor that is upwards compatible to Vi.  It can be used
       to edit all kinds of plain text.  It is especially useful  for  editing
       programs.

       There  are a lot of enhancements above Vi: multi level undo, multi win-
       dows and buffers, syntax highlighting, command line  editing,  filename
 Manual page vi(1) line 1 (press h for help or q to quit)
```

Client:

```
user@user-virtual-machine:~/CN_LAB/9$ gcc udpremoteclint.c -o clin
user@user-virtual-machine:~/CN_LAB/9$ ./clin 127.0.0.1

Enter the command to be executed vim -vi

Data SentRecieved From UDPSERVER
Enter the command to be executed man vi

Data SentRecieved From UDPSERVER
Enter the command to be executed^C
```

Result:

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

EX NO: 10 ARP IMPLEMENTATION USING UDP
Date:

Aim:

To design and implement Address Resolution Protocol (ARP) using User Datagram Protocol (UDP) as the underlying communication protocol using c programming.

Given Requirements:

1. Understanding of the Address Resolution Protocol (ARP).
2. Familiarity with the User Datagram Protocol (UDP).
3. Programming knowledge in a language like C/C++.
4. Access to a network environment for testing

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 is pinged.

Methodology:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Declare structures arpreq (as NULL structure, if required) and sockaddr_in.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET and sin_addr using inet_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 required Client.
- Using the ioctl() we get the ARP cache entry for the given IP address.
- 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 structure arpreq.
- Print the hardware address of the given IP address on the output console.

Code:

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>
#include<stdlib.h>
#include<string.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)); //
Removed extra *(ptr+5) from the format string
return 0;
}
```

Source Output:

```
user@user-virtual-machine:~/CN_LAB/10$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=7.15 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=10.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=7.57 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=10.6 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=128 time=7.91 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4010ms
rtt min/avg/max/mdev = 7.153/8.660/10.609/1.400 ms
user@user-virtual-machine:~/CN_LAB/10$ gcc apr.c -o outarp
user@user-virtual-machine:~/CN_LAB/10$ ./outarp 8.8.8.8
MAC Address For '8.8.8.8' : 0:0:0:0:0:0
```

Result:

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 STUDY OF IPV6 ADDRESSING & SUBNETTING
Date:

AIM:

To Study the IPV6 Addressing and Subnetting

What is IPv6:

As the number of internet devices—also known as the Internet of Things (IoT)—increases around theworld, more IP addresses are needed for these devices to communicate data. Consider smartphones,smartwatches, refrigerators, washing machines, smart TVs, and other electronic devices that require an IPaddress. All of these devices are now linked to the internet and have a unique IP address assigned to them. We'llfocus on IPv6, its characteristics, and why it'll be the Internet Protocol standard in this quick overview.

Before we go into the technicalities, there are a few things to know about IPv6:

1. IPv6 addresses are 128-bit (2128) and allow for 3.4 x 1038 unique IP addresses.
2. IPv6 is written in hexadecimal notation, with the colons separating eight groups of 16 bits, for a totalof 8 x 16 = 128, or bits. The following is an example of an IPv6 address:

➢ **Syntax of IPv6 Addresses:**

IPv4 addresses are represented in dotted-decimal format. The 32-bit address is divided along 8-bit boundaries. Each set of 8 bits is converted to its decimal equivalent and separated by periods. In contrast, IPv6 addresses are 128 bits divided along 16-bit boundaries. Each 16-bit block is converted to a 4-digit hexadecimal number and separated by colons. The resulting representation is called colon-hexadecimal.

– The preferred form is x:x:x:x:x:x:x:x, where the x's are the hexadecimal values of the eight 16-bit pieces of the address. For example:

```
FEDC:BA98:7654:3210:FEDC:BA98:7654:3210

1080:0:0:0:8:800:200C:417A
```

➢ The Addressing Space:

The amount of memory dedicated to all potential addresses for a computational object, such as a device, a file, a server, or a networked computer, is known as address space. A range of physical or virtual addresses accessible to a processor or reserved for a process is referred to as address space. Each address defines an entity's location as a unique identifier of single entities (unit of memory that can be addressed separately). Each computer device and process is given address space on the computer, which is a piece of the processor's address space. The address space of a processor is always constrained by the width of its address bus and registers. Flat address space, in which addresses are expressed as continuously growing integers starting at zero, and segmented address space, in which addresses are written as discrete segments

enhanced by offsets, are the two types of address space (values added to produce secondary addresses). Thinking is a procedure that allows address space to be changed from one format to another in some systems. In terms of IP address space, there has been concern that IPv4 (Internet Protocol Version 4) had not anticipated the enormous growth of the Internet, and that its 32-bit address space would not be adequate. For that reason, IPv6 has been developed with 128-bit address space.

Allocation of the IPv6 addressing space:

Allocation	Prefix (binary)	Fraction of Address Space
Reserved	0000 0000	1/256
Unassigned	0000 0001	1/256
Reserved for NSAP addresses	0000 001	1/128
Reserved for IPX addresses	0000 010	1/128
Unassigned	0000 011	1/128
Unassigned	0000 1	1/32
Unassigned	0001	1/16
Aggregatable global unicast addresses	001	1/8
Unassigned	010	1/8
Unassigned	011	1/8
Reserved for Geographic-based addresses	100	1/8
Unassigned	101	1/8
Unassigned	110	1/8
Unassigned	1110	1/16
Unassigned	1111 0	1/32
Unassigned	1111 10	1/64
Unassigned	1111 110	1/128
Unassigned	1111 1110 0	1/512
Link Local addresses	1111 1110 10	1/1024

➢ Types of IPv6 Addresses: Generally, IPv6 addresses is classified into 3. They are:

1. Unicast: This type is the address of a single interface. A packet forwarded to a unicast address is delivered only to the interface identified by that address.

2. Anycast: This type is the address of a set of interfaces typically belonging to different nodes. A packet forwarded to an anycast address is delivered to only one interface of the set (the nearest to the source node, according to the routing metric).

3. Multicast: This type is the address of a set of interfaces that typically belong to different nodes. A packet forwarded to a multicast address is delivered to all interfaces belonging to the set.

1. Unicast Addresses:

A unicast address identifies a single interface. When a network device sends a packet to a unicast address, the packet goes only to the specific interface identified by that address. Unicast addresses support a global address scope and two types of local address scopes.

A unicast address consists of n bits for the prefix, and 128 – n bits for the interface ID.

- Global unicast address—A unique IPv6 address assigned to a host interface. These addresses have a global scope and essentially the same purposes as IPv4 public addresses. Global unicast addresses are routable on the Internet.
- Link-local IPv6 address—An IPv6 address that allows communication between neighboring hosts that reside on the same link. Link-local addresses have a local scope, and cannot be used outside the link. They always have the prefix FE80::/10.
- Loopback IPv6 address—An IPv6 address used on a loopback interfaces. The IPv6 loopback address is 0:0:0:0:0:0:0:1, which can be notated as ::1/128.
- Unspecified address—An IPv6 unspecified address is 0:0:0:0:0:0:0:0, which can be notated as ::/128.

1. Aggregatable Global Unicast Addresses: Aggregate global unicast addresses are used for global communication. These addresses are similar in function to IPv4 addresses under classless interdomain routing (CIDR). The following table shows their format.

3 bits	45 bits	16 bits	64 bits
001	global routing prefix	subnet ID	interface ID

2. Geographic-Based Addresses:
   Geography addresses are those determined by country of origin. This type of address is only availablein the IPv4 address category. The data address table includes a ‘scope’ and a ‘authority‘

Scope	Authority
Multiregional	IANA
Europe	RIPE-NCC
Northern America	INTERNIC
Asia and Pacific	APNIC

3. Link Local Addresses: A link-local address is a network address that is valid only for communications within the network segment or the broadcast domain that the host is connected to. Link-local addresses are most often assigned automatically with a process known as stateless address autoconfiguration or link-local address autoconfiguration,[1] also known as automatic private IP addressing (APIPA) or auto-IP.
4. Site Local Addresses: Site-local addresses are designed to be used for addressing inside of a site without the need for a global prefix. A site-local address cannot be reached from another site. A site-local address is not automatically assigned to a node. It must be assigned using automatic or manual configuration.
5. The Unspecified Address: The address 0:0:0:0:0:0:0:0 is called the unspecified address. It will not be assigned to any node. It indicates the absence of an address. One example of its use is in the Source Address field of any IPv6 packets sent by an initializing host before it has learned its own address.
6. The Loopback Address: The IP address 127.0.0.1 is called a loopback address. Packets sent to this address never reach the network but are looped through the network interface card only. This can be used for diagnostic purposes to verify that the internal path through the TCP/IP protocols is working.
7. NSAP Addresses: Short for Network Service Access Point, NSAP is an address consisting of up to 20 octets that identify a computer or network connected to an ATM network. NSAP is defined in ISO/IEC 8348.
8. IPX Addresses: Internetwork Packet Exchange (IPX) is the network layerprotocol in the IPX/SPXprotocol suite. IPX is derived from Xerox Network Systems' IDP. It may act as a transport layer protocol as well.
    3. Multicast Addresses: A multicast address identifies a set of interfaces that typically belong to different nodes. When a network device sends a packet to a multicast address, the device broadcasts the packet to all interfaces identified by that address. IPv6 does not support broadcast addresses, but instead uses multicast addresses in this role. Multicast addresses support 16 different types of address scope, including node, link, site, organization, and global scope. A 4-bit field in the prefix identifies the address scope. The following types of multicast addresses can be used in an IPv6 subscriber access network:

- Solicited-node multicast address—Neighbor Solicitation (NS) messages are sent to this address.
- All-nodes multicast address—Router Advertisement (RA) messages are sent to this address.
- All-routers multicast address—Router Solicitation (RS) messages are sent to this address.

➢ Which addresses are generally used for a node?

1. Addresses of a Host:

- Its Link Local address for each interface
- Unicast addresses assigned to interfaces
- The loopback address
- All-Nodes multicast address
- Neighbor Discovery multicast addresses associated with all uni-cast and anycast addresses assigned to interfaces
- Multicast Addresses of groups to which the node belongs

2. Addresses of a Router:

- Its Link Local address for each interface
- Unicast addresses assigned to interfaces
- The loopback address
- The Subnet Router anycast address for all links on which it has interfaces
- Other anycast addresses assigned to interfaces
- All-nodes multicast address
- All-routers multicast address
- Neighbor Discovery multicast addresses associated with all uni-cast and anycast addresses assigned to interfaces
- Multicast addresses of groups to which the node belongs

Result:
Hence Study of IPV6 Addressing & Subnetting is completed successfully

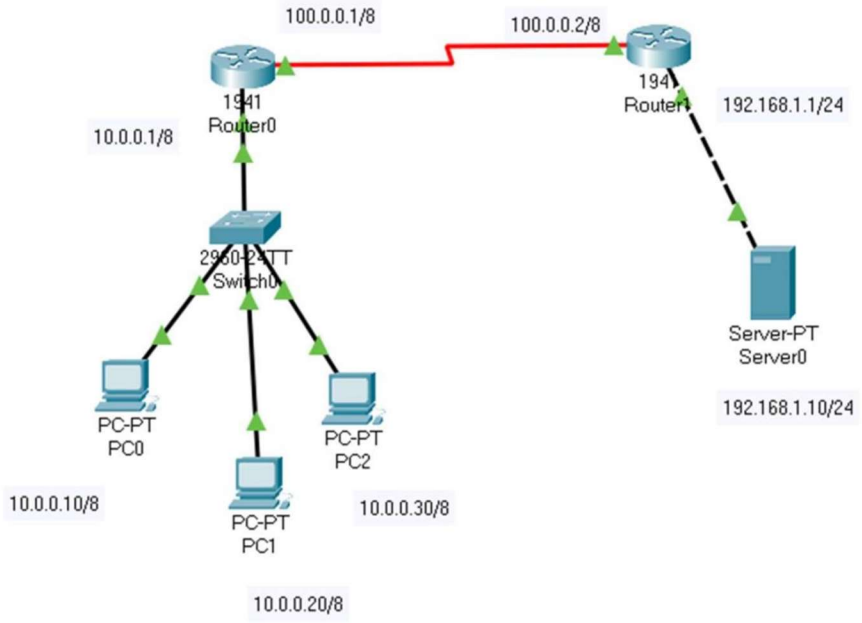
EX NO: 12 **IMPLEMENTATION OF NETWORK ADDRESS**
Date: **TRANSLATION**

Aim:

To study and perform Network Address Translation (NAT) using cisco packet tracer.

Procedure:

1. Assign the following topology with respective IP addresses to pc, routers, servers and connection between them



2. Configure static NAT configuration

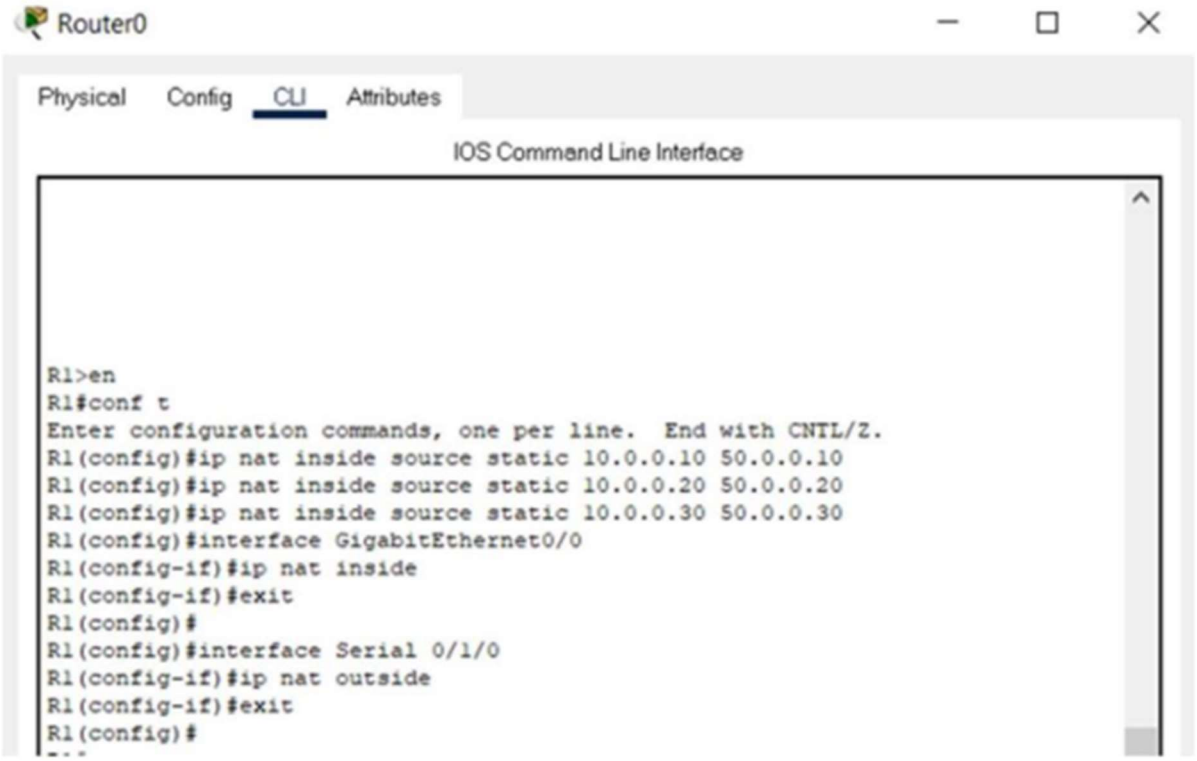
Since static NAT use manual translation, we have to map each inside local IP address (which needs a translation) with inside global IP address. Following command is used to map the inside local IP address with inside global IP address.

```
Router(config)#ip nat inside source static [inside local ip address] [inside global IP address]
```

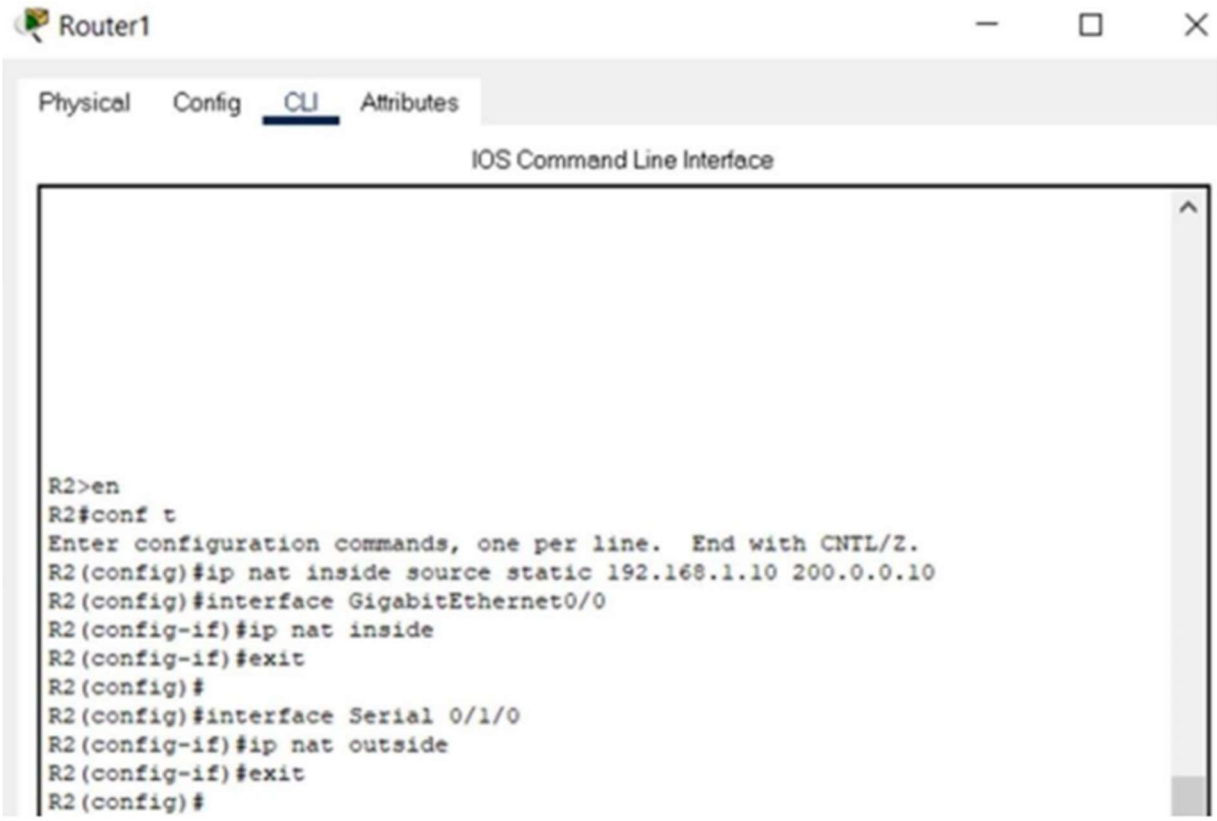
And use the following commands to define inside and outside network connection for your local and global IP addresses.

```
Router(config-if)#ip nat inside
Router(config-if)#ip nat outside
```

Static NAT configuration for Router0 connected with 3 pc's:



Static NAT configuration for Router0 connected with server:



3.Configure the IP routing

IP routing is the process which allows router to route the packet between different networks.

IP routing on router0:

```
R1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#ip route 200.0.0.0 255.255.255.0 100.0.0.2
R1(config)#no shutdown
                ^
```

IP routing on router1:

```
R2(config)#
R2(config)#ip route 50.0.0.0 255.0.0.0 100.0.0.1
```

4. Testing Static NAT Configuration

To test this setup click on any PC and Desktop and click Command Prompt.

• Run ipconfig command.

• Run ping 200.0.0.10 command.

• Run ping 192.168.1.10 command

First command verifies that we are testing from correct NAT device.

Second command checks whether we are able to access the remote device or not. A ping reply confirms that

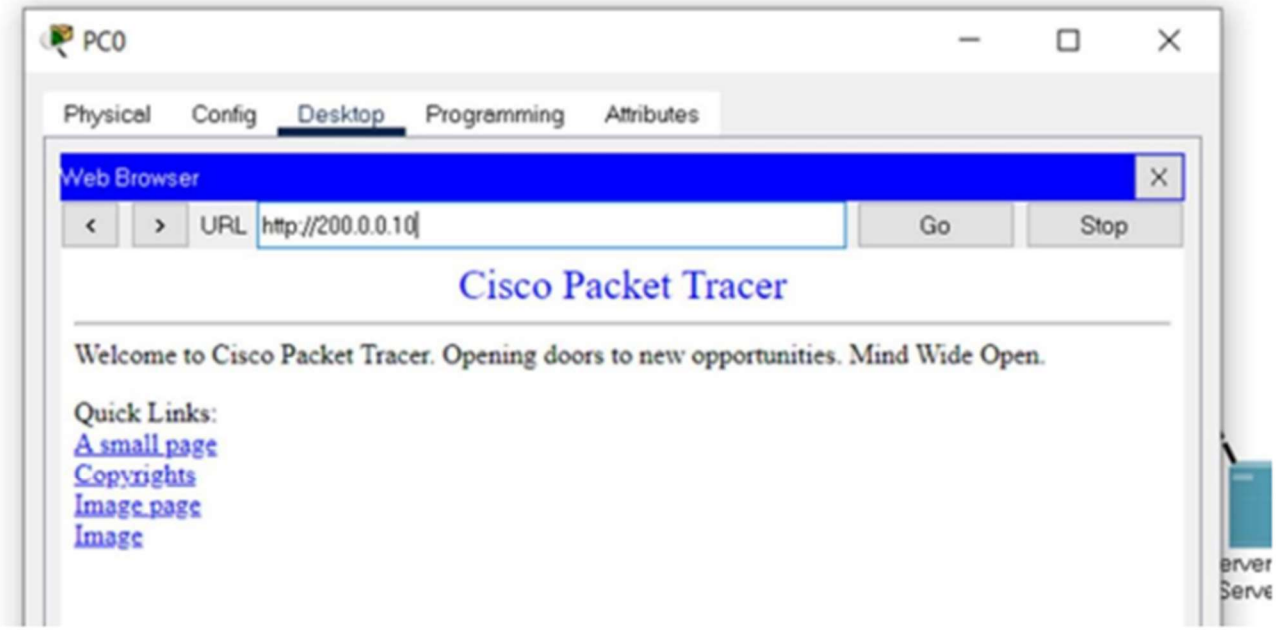
we are able to connect with remote device on this IP address.

Third command checks whether we are able to access the remote device on its actual IP address or not. A ping

error confirms that we are not able to connect with remote device on this IP address.

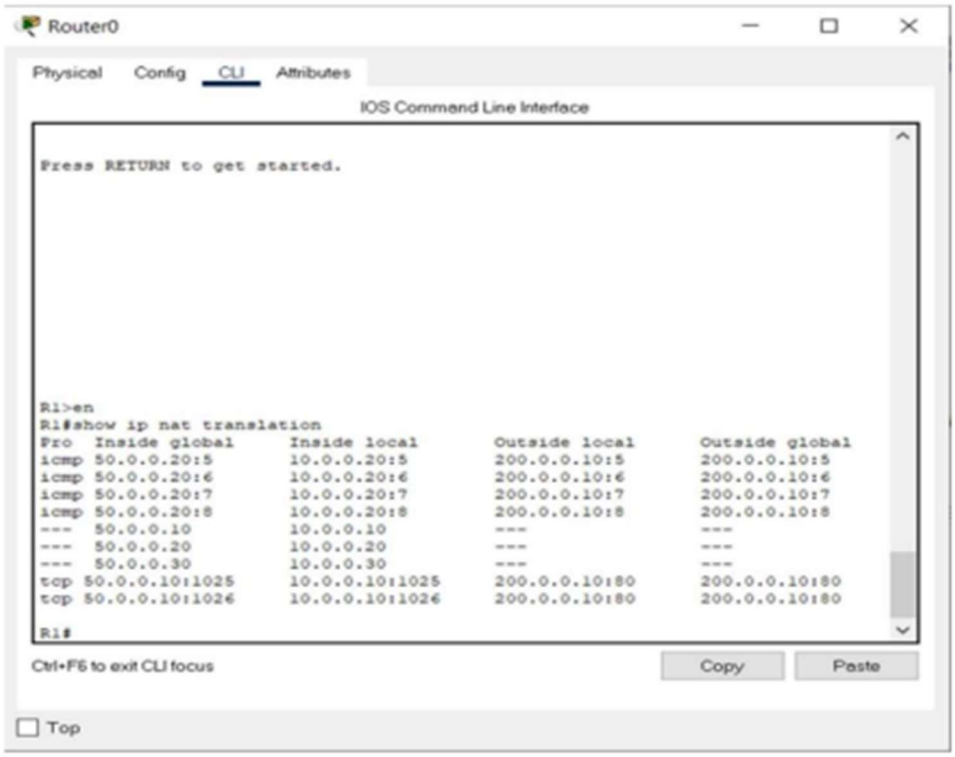

PC0
Physical
Config
Desktop
Programming
Attributes
Command Prompt
Reply from 10.0.0.1: Destination host unreachable.
Ping statistics for 192.168.1.10:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
C:\>ipconfig
FastEthernet0 Connection:(default port)
Connection-specific DNS Suffix..:
Link-local IPv6 Address.....: FE80::260:47FF:FE93:623B
IPv6 Address.....: ::
IPv4 Address.....: 10.0.0.10
Subnet Mask.....: 255.0.0.0
Default Gateway.....: ::
10.0.0.1
Bluetooth Connection:
Connection-specific DNS Suffix..:
Link-local IPv6 Address.....: ::
IPv6 Address.....: ::
IPv4 Address.....: 0.0.0.0
Subnet Mask.....: 0.0.0.0
Default Gateway.....: ::
0.0.0.0
C:\>ping 200.0.0.10
Pinging 200.0.0.10 with 32 bytes of data:
Reply from 200.0.0.10: bytes=32 time=10ms TTL=126
Reply from 200.0.0.10: bytes=32 time=1ms TTL=126
Reply from 200.0.0.10: bytes=32 time=2ms TTL=126
Reply from 200.0.0.10: bytes=32 time=8ms TTL=126
Ping statistics for 200.0.0.10:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 1ms, Maximum = 10ms, Average = 5ms
C:\>ping 192.168.1.10
Pinging 192.168.1.10 with 32 bytes of data:
Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.
Ping statistics for 192.168.1.10:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
C:\>
Top

Another way of testing is via browser:



We can also verify this translation on router with ***show ipnat translation*** command.

For router0:



For router1:

```
R2#show ip nat translation
Pro  Inside global     Inside local       Outside local      Outside global
---  200.0.0.10        192.168.1.10       ---                ---
tcp 200.0.0.10:80      192.168.1.10:80    50.0.0.10:1025     50.0.0.10:1025
tcp 200.0.0.10:80      192.168.1.10:80    50.0.0.10:1026     50.0.0.10:1026

R2#
```

Result:

Henceforth, Network Address Translation (NAT) using cisco packet tracer implemented and verified

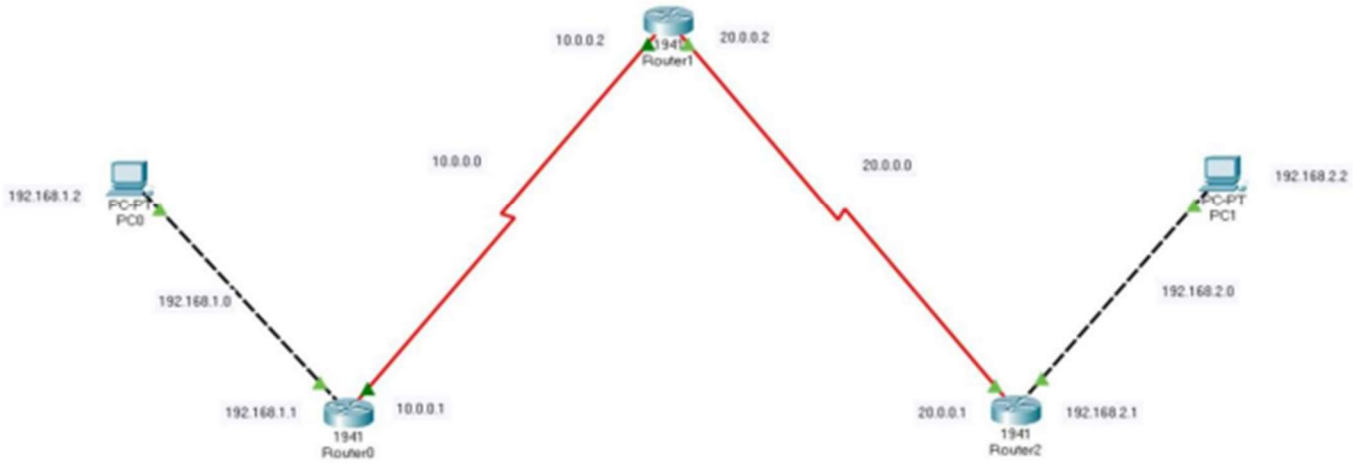
EX NO: 13 **IMPLEMENTATION OF VPN**
Date:

AIM :

To configure VPN using routers in Cisco Packet Tracer.

PROCEDURE :

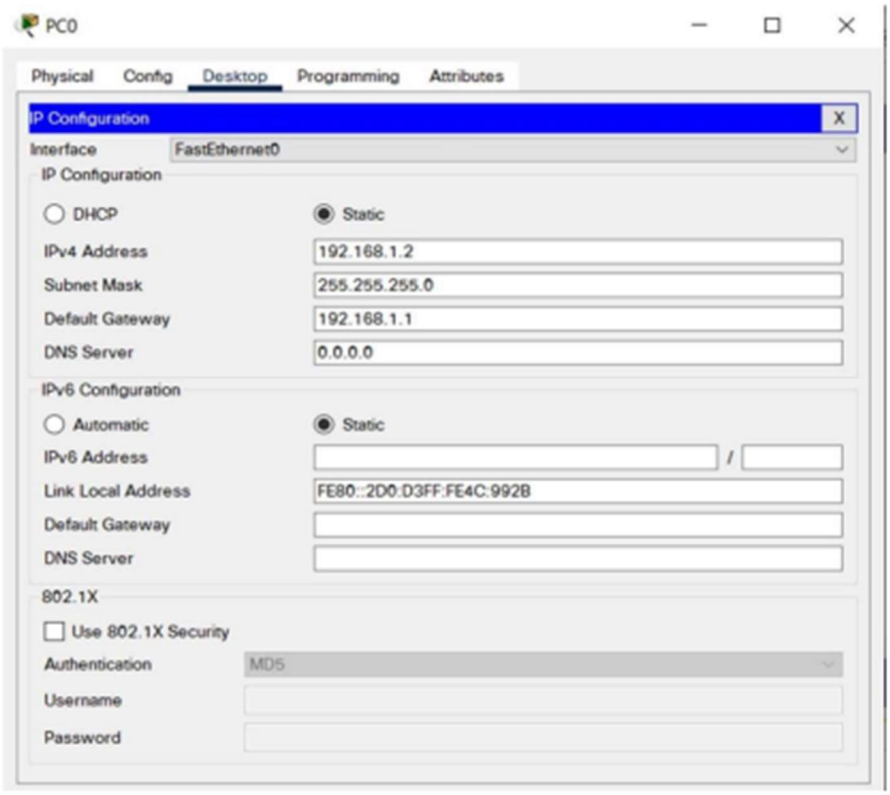
1 . Connect the devices as shown in the below figure.



2 . Initial IP configuration.

Device / Interface	IP Address	Connected with
PC0 / Fa0	192.168.1.2 /24	Router1 / Gig0/0
PC1 / Fa0	192.168.2.2 /24	Router2 / Gig0/0
Router1 / Se0/1/0	10.0.0.1 /8	Router 2 / Se0/1/0
Router2 / Se0/1/0	10.0.0.2 /8	Router 1 / Se0/1/0
Router2 / Se0/1/1	20.0.0.1 /8	Router3 / Se0/1/0
Router3 / Se0/1/0	20.0.0.2 /8	Router2 / Se0/1/1

3 .To assign IP address in Laptop click Laptop and click Desktop and IP configuration and Select Static and set IP address as given in above table.



Following the same way, configure the IP address in PC1.

4. We have to assign ip address on each and every interface of router

CONFIGURATION ON ROUTER1:

```
Router>enable

Router#config t

Router(config)#int gig0/0

Router(config-if)#ip add 192.168.1.1 255.255.255.0

Router(config-if)#no shut

Router(config-if)#exit

Router(config)#int se0/1/0

Router(config-if)#ip address 10.0.0.1 255.0.0.0

Router(config-if)#no shut
```

CONFIGURATION ON ROUTER2:

```
Router>enable
Router#config t
Router(config)#int se0/1/0
Router(config-if)#ip add 10.0.0.2 255.0.0.0
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#int se0/1/1
Router(config-if)#ip add 20.0.0.1 255.0.0.0
Router(config-if)#no shut
```

CONFIGURATION ON ROUTER3:

```
Router>enable
Router#config t
Router(config)#int se0/1/0
Router(config-if)#ip add 20.0.0.2 255.0.0.0
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#int gig0/0
Router(config-if)#ip add 192.168.2.1 255.255.255.0
Router(config-if)#no shut
```

5. Now it's time to do routing. Here we have to configure default routing.

DEFAULT ROUTING CONFIGURATION ON ROUTER1:

```
Router>enable
Router#config t
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#ip route 0.0.0.0 0.0.0.0 10.0.0.2
Router(config)#
```

DEFAULT ROUTING CONFIGURATION ON ROUTER3:

```
Router>enable
Router#config t
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#ip route 0.0.0.0 0.0.0.0 20.0.0.1
Router(config)#
```

6. NOW CHECK THE CONNECTION BY PINGING EACH OTHER.

First we go to Router1 and ping with Router3:

```
Router#ping 20.0.0.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 20.0.0.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 26/28/33 ms
```

Now we go to Router3 and test the network by pinging Router1 interface.

```
Router#ping 10.0.0.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:
!!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 25/28/32 ms
```

You can clearly see both routers pinging each other successfully.

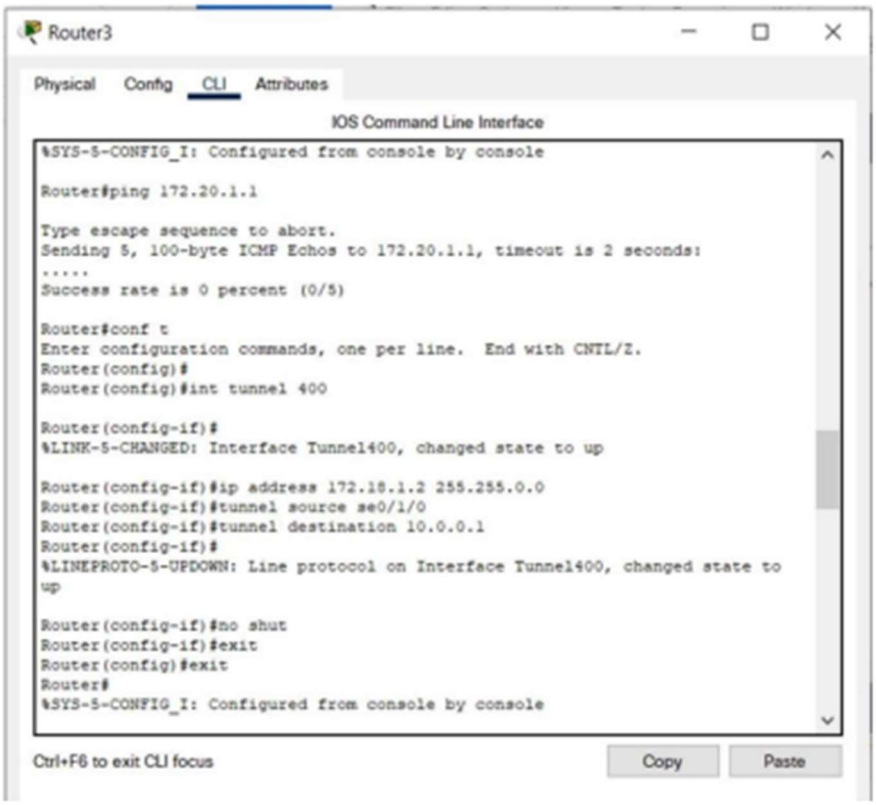
7. NOW CREATE VPN TUNNEL between Router1 and Router3:

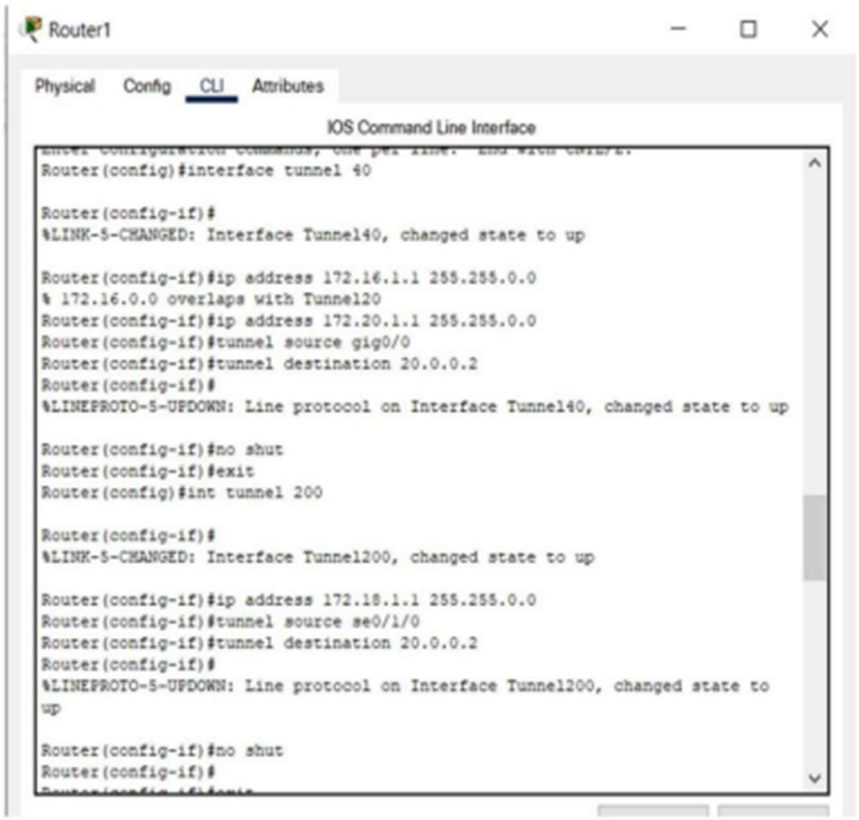
FIRST CREATE A VPN TUNNEL ON ROUTER1:

```
Router#config t
Router(config)#interface tunnel 200
Router(config-if)#ip address 172.18.1.1 255.255.0.0
Router(config-if)#tunnel source se0/1/0
Router(config-if)#tunnel destination 20.0.0.2
Router(config-if)#no shut
```

NOW CREATE A VPN TUNNEL ON ROUTER R3:

```
Router#config t
Router(config)#interface tunnel 400
Router(config-if)#ip address 172.18.1.2 255.255.0.0
Router(config-if)#tunnel source se0/1/0
Router(config-if)#tunnel destination 10.0.0.1
Router(config-if)#no shut
```


Router3
Physical
Config
CLI
Attributes
IOS Command Line Interface
%SYS-5-CONFIG_I: Configured from console by console
Router#ping 172.20.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.20.1.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
Router#conf t
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#
Router(config)#int tunnel 400
Router(config-if)#
%LINK-5-CHANGED: Interface Tunnel400, changed state to up
Router(config-if)#ip address 172.18.1.2 255.255.0.0
Router(config-if)#tunnel source se0/1/0
Router(config-if)#tunnel destination 10.0.0.1
Router(config-if)#
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel400, changed state to up
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console
Ctrl+F6 to exit CLI focus
Copy
Paste


Router1
Physical
Config
CLI
Attributes
IOS Command Line Interface
Router(config)#interface tunnel 40
Router(config-if)#
%LINK-5-CHANGED: Interface Tunnel40, changed state to up
Router(config-if)#ip address 172.16.1.1 255.255.0.0
% 172.16.0.0 overlaps with Tunnel20
Router(config-if)#ip address 172.20.1.1 255.255.0.0
Router(config-if)#tunnel source gig0/0
Router(config-if)#tunnel destination 20.0.0.2
Router(config-if)#
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel40, changed state to up
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#int tunnel 200
Router(config-if)#
%LINK-5-CHANGED: Interface Tunnel200, changed state to up
Router(config-if)#ip address 172.18.1.1 255.255.0.0
Router(config-if)#tunnel source se0/1/0
Router(config-if)#tunnel destination 20.0.0.2
Router(config-if)#
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel200, changed state to up
Router(config-if)#no shut
Router(config-if)#

8. Now test communication between these two routers again by pinging each other:

Router1

```
Router#ping 172.18.1.2

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 172.18.1.2, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 30/32/36 ms

Router#
```

Router2

```
Router#ping 172.18.1.1

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 172.18.1.1, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 33/45/83 ms
```

9. Now do routing for created VPN Tunnel on Both Router1 and Router3:

```
Router(config)#ip route 192.168.2.0 255.255.255.0 172.18.1.2

Router(config)#ip route 192.168.1.0 255.255.255.0 172.18.1.1
```

10. TEST VPN TUNNEL CONFIGURATION:

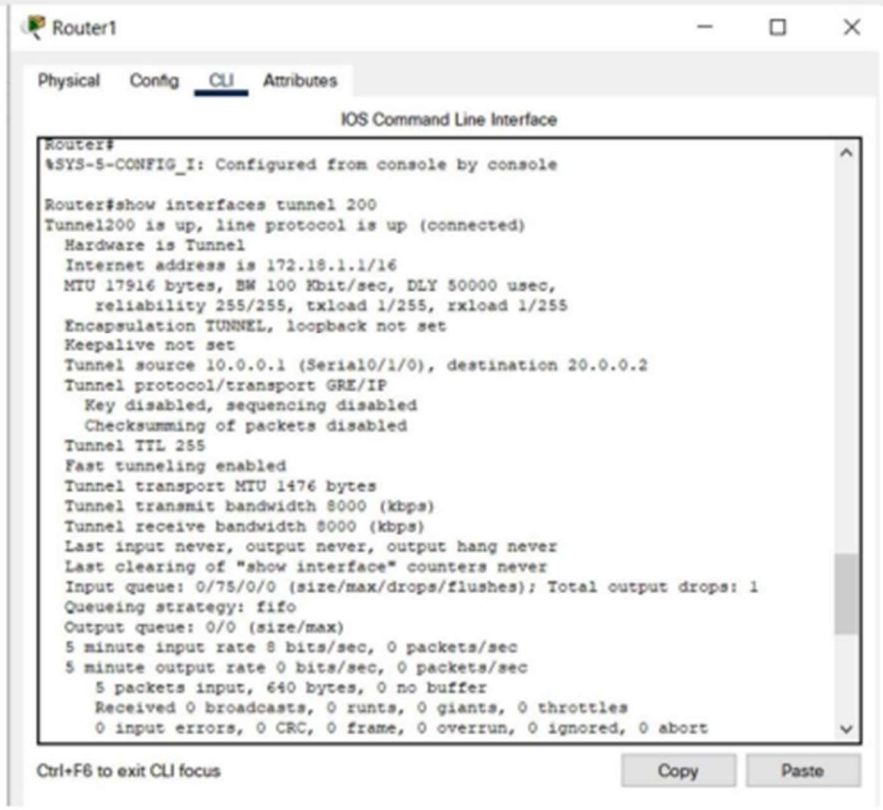
Now we have to test whether tunnel is created or not for Router1

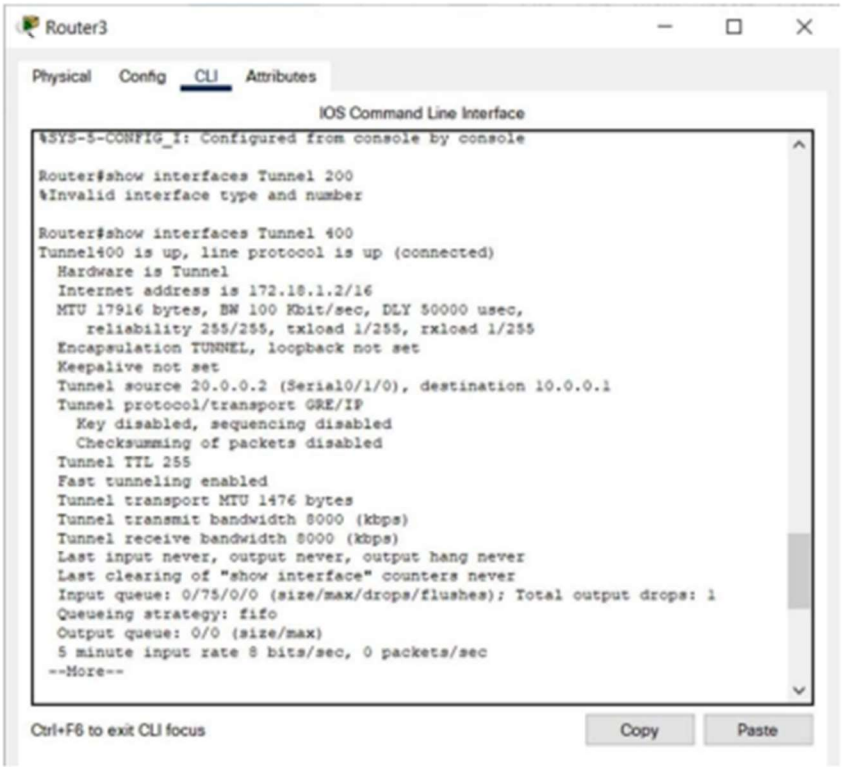
```
Router#show interfaces Tunnel 200

Tunnel200 is up, line protocol is up (connected)

Hardware is Tunnel

Internet address is 172.18.1.1/16
```

```
MTU 17916 bytes, BW 100 Kbit/sec, DLY 50000 usec,
reliability 255/255, txload 1/255, rxload 1/255
Encapsulation TUNNEL, loopback not set
Keepalive not set
Tunnel source 10.0.0.1 (FastEthernet0/1), destination 20.0.0.2
Tunnel protocol/transport GRE/IP
Key disabled, sequencing disabled
Checksumming of packets disabled
Tunnel TTL 255
Fast tunneling enabled
Tunnel transport MTU 1476 bytes
Tunnel transmit bandwidth 8000 (kbps)
Tunnel receive bandwidth 8000 (kbps)
Last input never, output never, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 1
Queueing strategy: fifo
Output queue: 0/0 (size/max)
5 minute input rate 32 bits/sec, 0 packets/sec
5 minute output rate 32 bits/sec, 0 packets/sec
52 packets input, 3508 bytes, 0 no buffer
Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
0 input packets with dribble condition detected
52 packets output, 3424 bytes, 0 underruns
0 output errors, 0 collisions, 0 interface resets
```

0 unknown protocol drops

0 output buffer failures, 0 output buffers swapped out





Now going to Router3 and test VPN Tunnel Creation:

```
Router #show interface Tunnel 400
Tunnel400 is up, line protocol is up (connected)
Hardware is Tunnel
Internet address is 172.18.1.2/16
MTU 17916 bytes, BW 100 Kbit/sec, DLY 50000 usec,
reliability 255/255, txload 1/255, rxload 1/255
Encapsulation TUNNEL, loopback not set
Keepalive not set
Tunnel source 20.0.0.2 (FastEthernet0/0), destination 10.0.0.1
Tunnel protocol/transport GRE/IP
Key disabled, sequencing disabled
Checksumming of packets disabled
Tunnel TTL 255
Fast tunneling enabled
Tunnel transport MTU 1476 bytes
Tunnel transmit bandwidth 8000 (kbps)
Tunnel receive bandwidth 8000 (kbps)
Last input never, output never, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 1
Queueing strategy: fifo


Output queue: 0/0 (size/max)
5 minute input rate 32 bits/sec, 0 packets/sec
```

5 minute output rate 32 bits/sec, 0 packets/sec

52 packets input, 3424 bytes, 0 no buffer

Received 0 broadcasts, 0 runts, 0 giants, 0 throttles

0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort

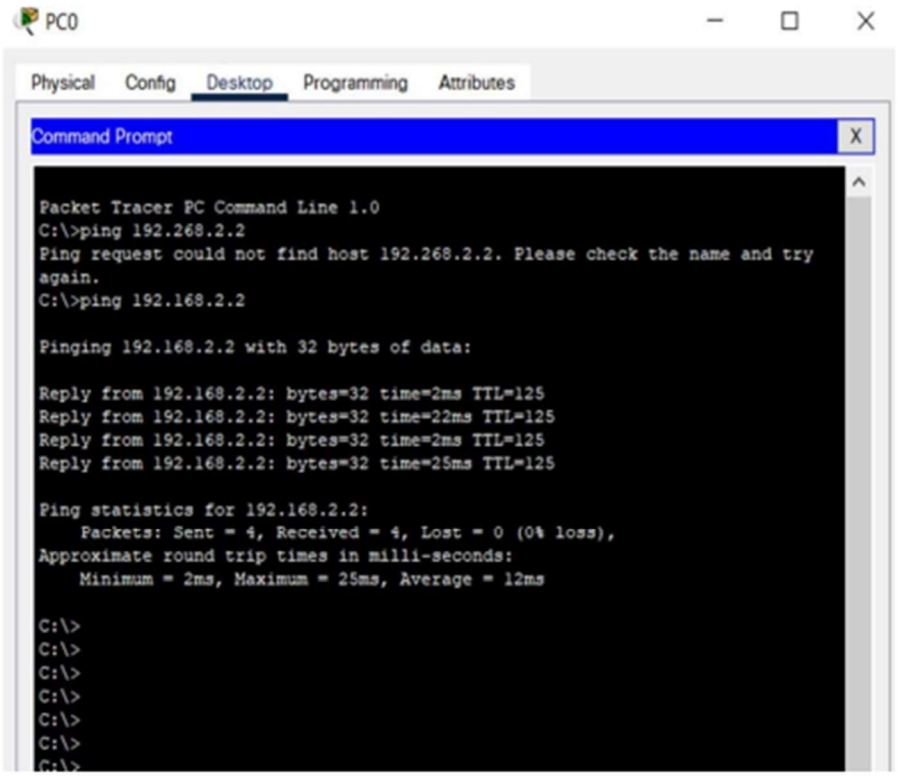
0 input packets with dribble condition detected

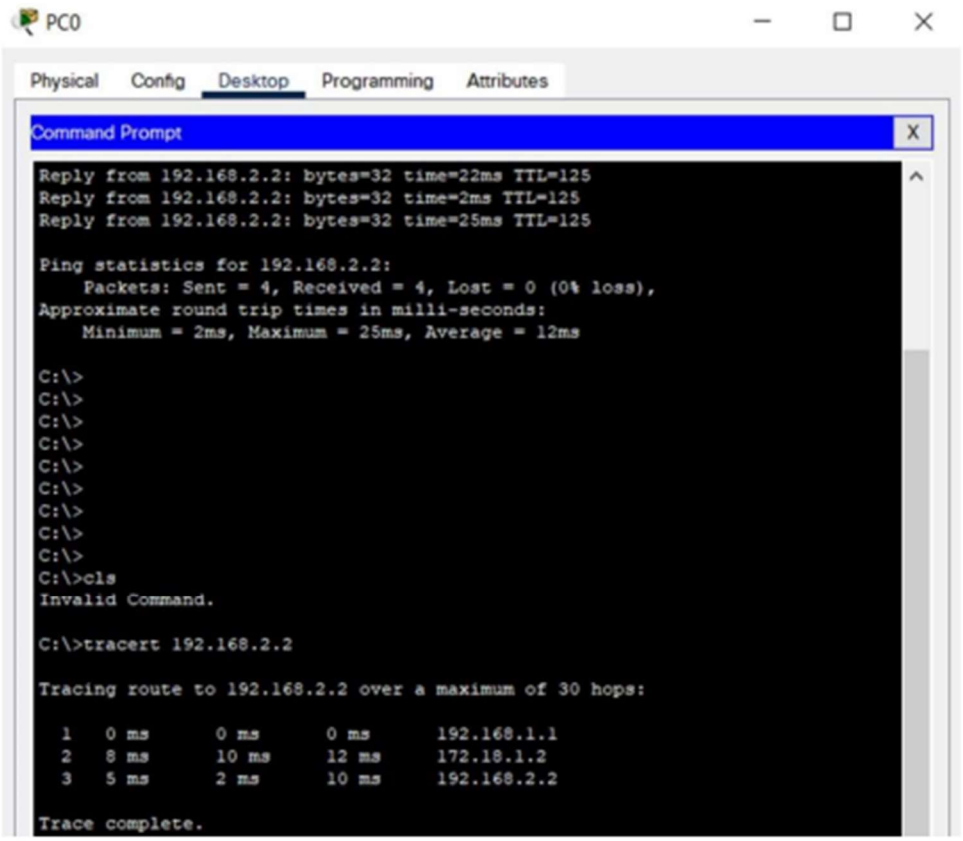
53 packets output, 3536 bytes, 0 underruns

0 output errors, 0 collisions, 0 interface resets

0 unknown protocol drops

11. Trace the VPN tunnel path.





RESULT:

Hence successfully, configured VPN using routers in Cisco Packet Tracer.

EX NO: 14 **COMMUNICATION USING HDLC**
Date:

AIM:

To configure HDLC Protocol using routers in Cisco Packet Tracer.

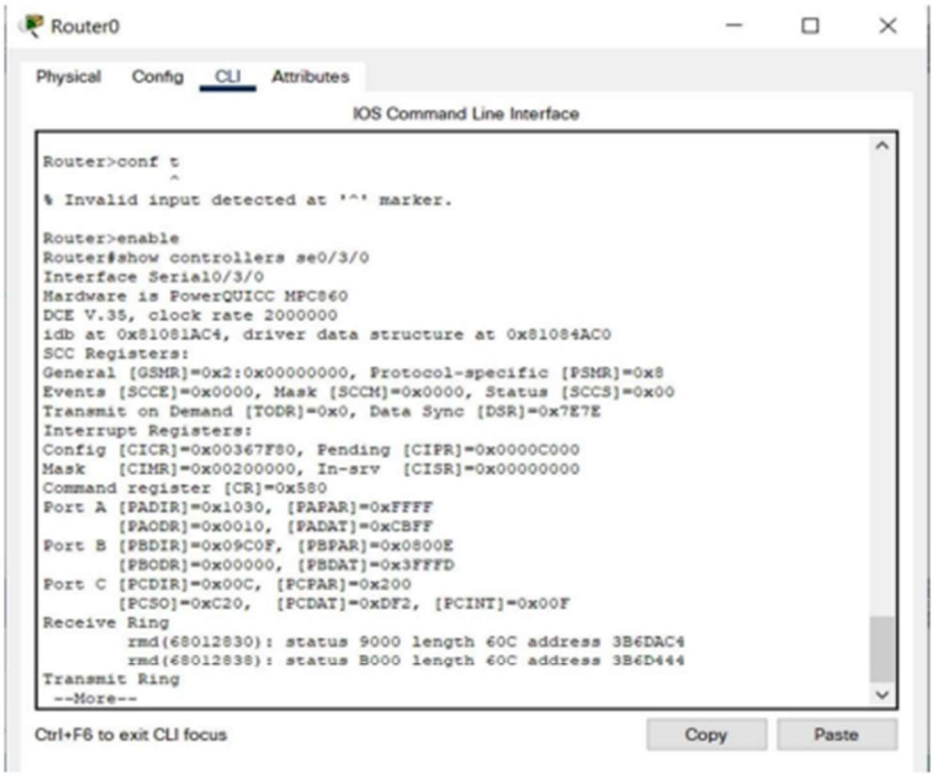
PROCEDURE:

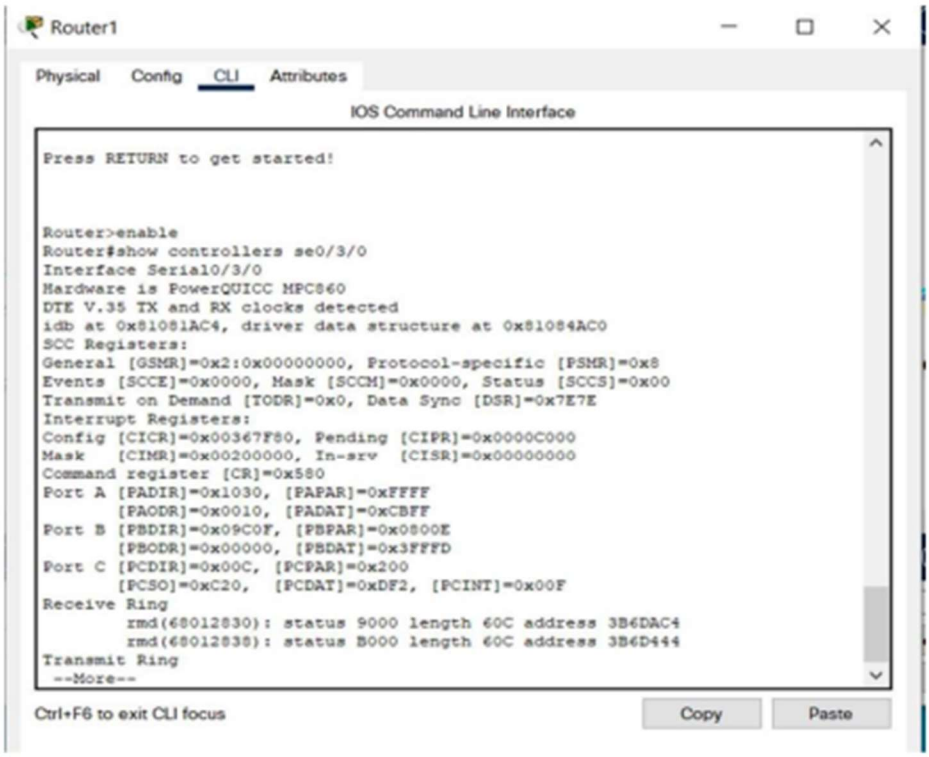
1. Connect the devices as shown in the below figure.

2 . Initial IP configuration.

Device / Interface	IP Address	Connected with
PC0 / Fa0	10.0.0.2 /8	Router0 / Fa0/0
PC1 / Fa0	20.0.0.2 /8	Router1 / Fa0/0
Router0 / Se0/3/0	192.168.1.2 /30	Router1 / Se0/3/0
Router1 / Se0/3/0	192.168.1.3 /30	Router0 / Se0/3/0

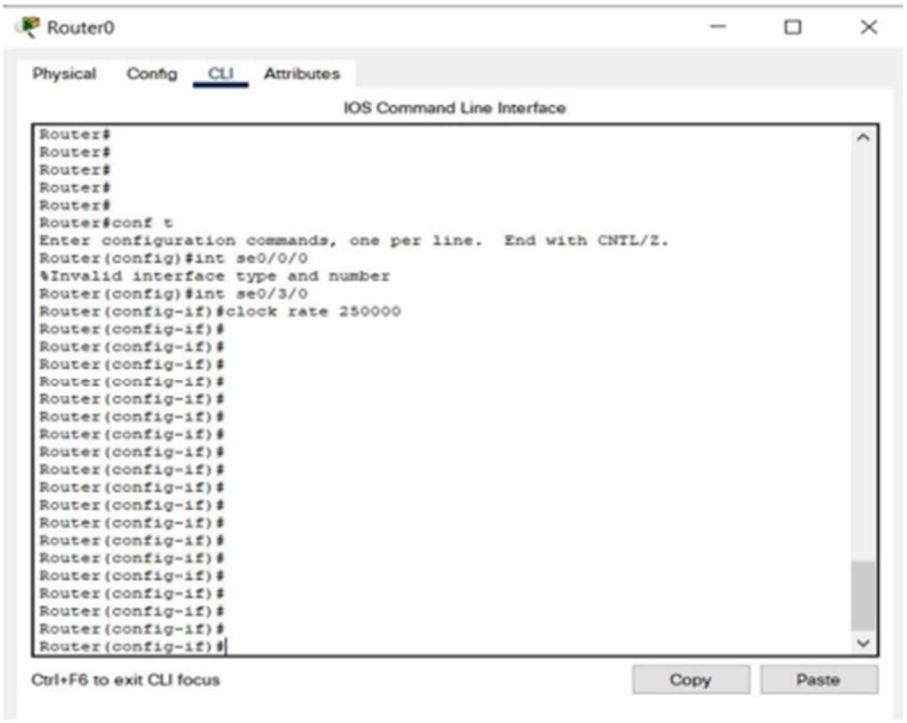
3 . Use the connected laptops to find the DCE and DTE routers



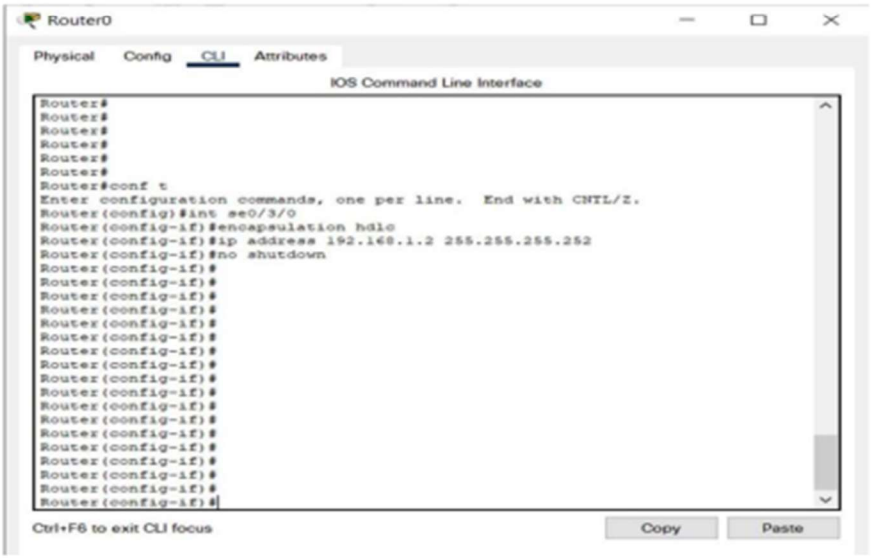


4. Configure the routers with the following parameters

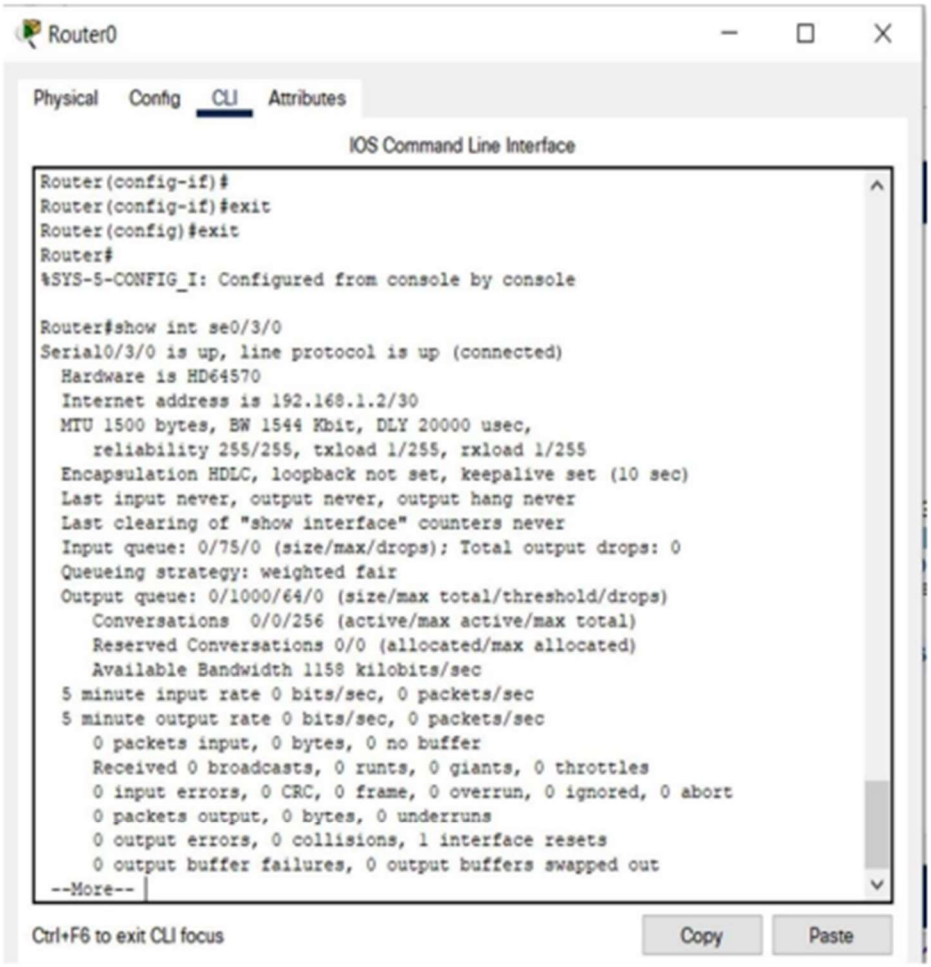
Router0 being the DCE, clock rate has to be configured on Router0 serial 0/3/0 interface.



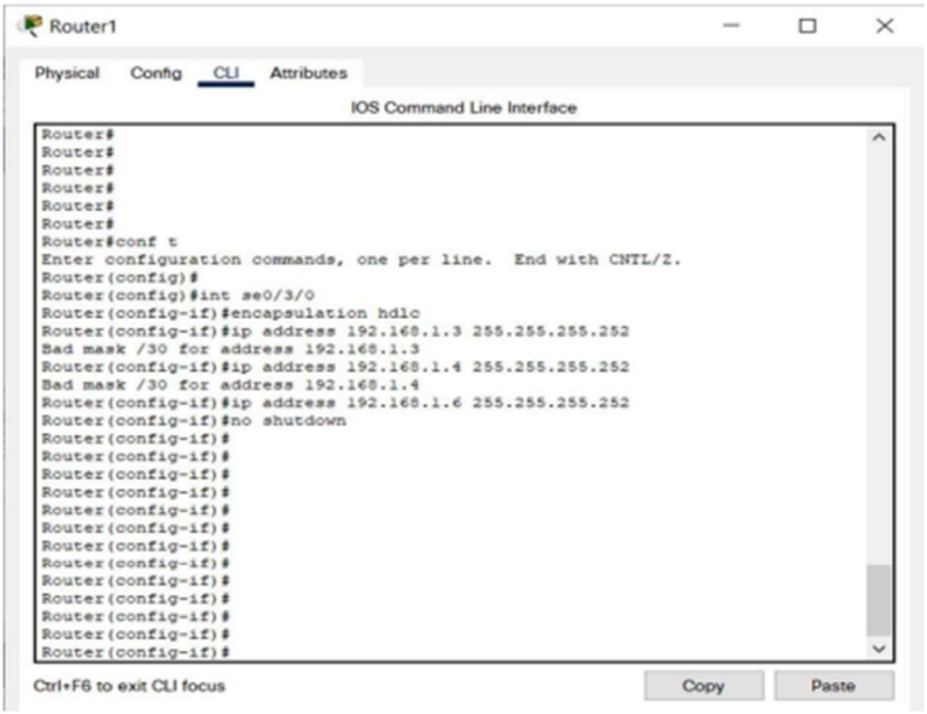
5. Then, configure HDLC encapsulation and IP address on Router0 serial 0/3/0 interface. The **encapsulation hdlc** configures HDLC protocol on the serial interface. Router0 being the DCE side of the serial link, the 192.168.1.3 /30 IP address is configured on Router0 serial 0/3/0 interface. Don't forget to enable the interface with a no shutdown command.



6. The show interfaces serial 0/3/0 confirms that HDLC encapsulation is enabled on the interface : Encapsulation HDLC, loopback not set, keepalive set (10 sec)



7. Finally, configure HDLC encapsulation and IP address on Router1 serial 0/3/0 interface. The link comes up as both routers are correctly configured.



8. NOW CHECK THE CONNECTION BY PINGING EACH OTHER.

First we go to Router0 and ping with Router1:

Router#ping 192.168.1.6

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 198.168.1.6, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 26/28/33 ms

Now we go to Router1 and test the network by pinging the Router0 interface.

Router#ping 192.168.1.2

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 192.168.1.2, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 25/28/32 ms

RESULT :

Hence successfully, configured HDLC Protocol using routers in Cisco Packet Tracer.

EX NO: 15 COMMUNICATION USING PPP
Date:

AIM:

To configure PPP using routers in Cisco Packet Tracer.

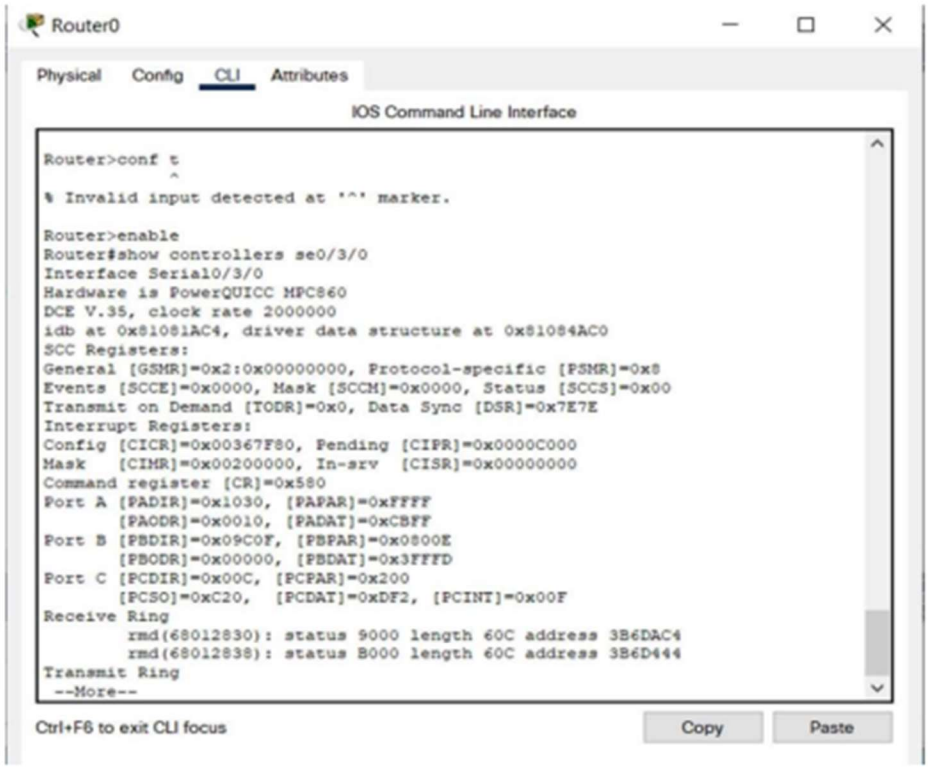
PROCEDURE:

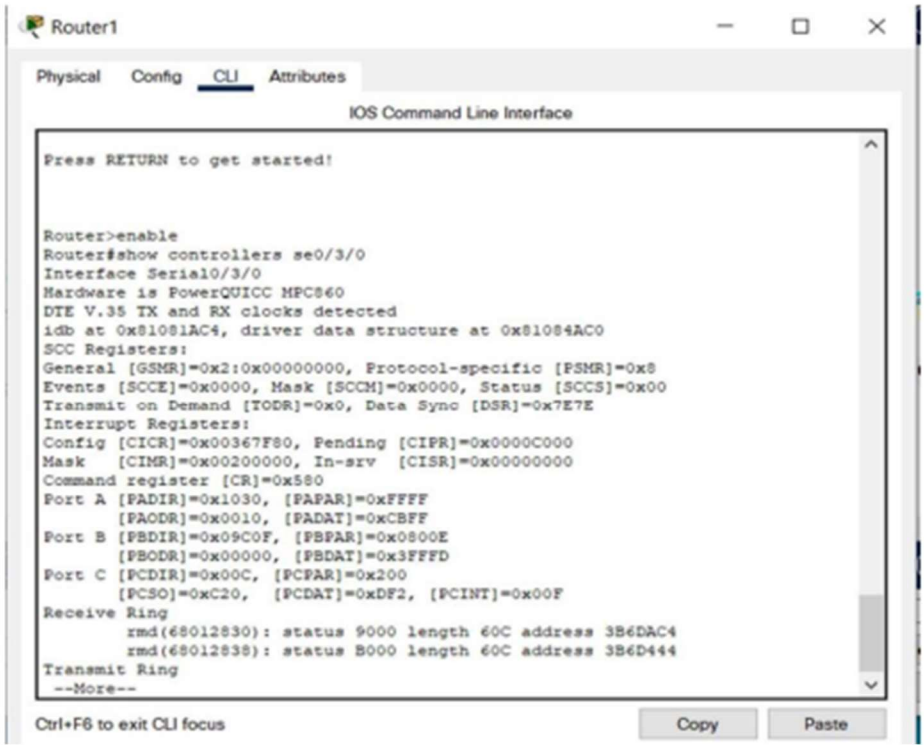
1 . Connect the devices as shown in the below figure.

2 . Initial IP configuration.

Device / Interface	IP Address	Connected with
PC0 / Fa0	10.0.0.2 /8	Router0 / Fa0/0
PC1 / Fa0	20.0.0.2 /8	Router1 / Fa0/0
Router0 / Se0/3/0	192.168.1.2 /30	Router1 / Se0/3/0
Router1 / Se0/3/0	192.168.1.3 /30	Router0 / Se0/3/0

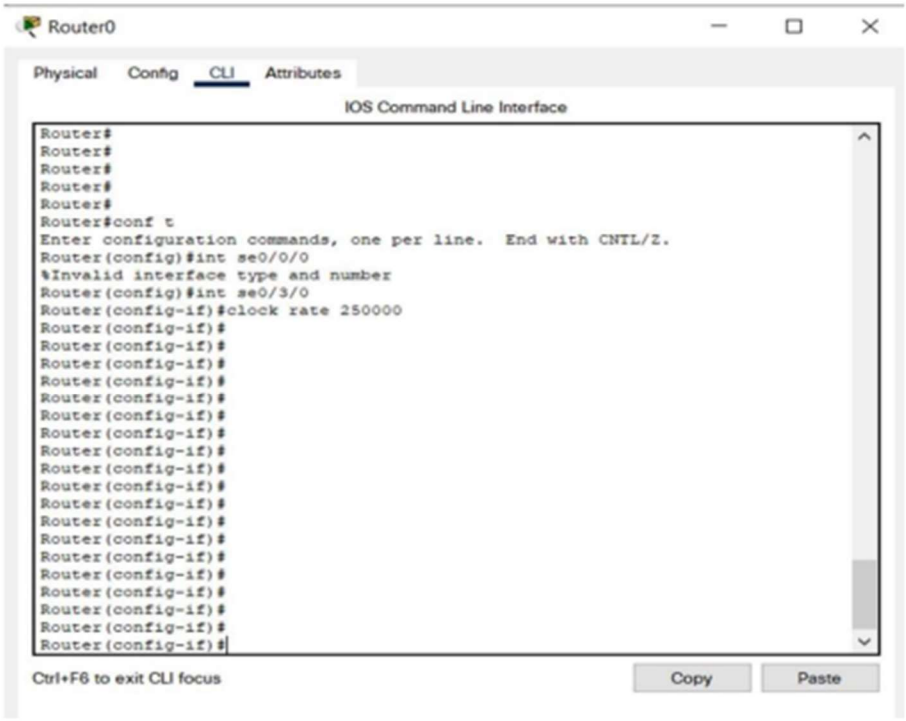
3 . Use the connected laptops to find the DCE and DTE routers


Router0
Physical
Config
CLI
Attributes
IOS Command Line Interface
Router>conf t
% Invalid input detected at '^' marker.
Router>enable
Router#show controllers se0/3/0
Interface Serial0/3/0
Hardware is PowerQUICC MPC860
DCE V.35, clock rate 2000000
idb at 0x81081AC4, driver data structure at 0x81084AC0
SCC Registers:
General [GSMR]=0x2:0x00000000, Protocol-specific [PSMR]=0x8
Events [SCCE]=0x0000, Mask [SCCM]=0x0000, Status [SCCS]=0x00
Transmit on Demand [TODR]=0x0, Data Sync [DSR]=0x7E7E
Interrupt Registers:
Config [CICR]=0x00367F80, Pending [CIPR]=0x0000C000
Mask [CIMR]=0x00200000, In-srv [CISR]=0x00000000
Command register [CR]=0x580
Port A [PADIR]=0x1030, [PAPAR]=0xFFFF
[PAODR]=0x0010, [PADAT]=0xCBFF
Port B [PBDIR]=0x09C0F, [PBPAR]=0x0800E
[PBODR]=0x00000, [PBDAT]=0x3FFFD
Port C [PCDIR]=0x00C, [PCPAR]=0x200
[PCSO]=0xC20, [PCDAT]=0xDF2, [PCINT]=0x00F
Receive Ring
rmd(68012830): status 9000 length 60C address 3B6DAC4
rmd(68012838): status B000 length 60C address 3B6D444
Transmit Ring
--More--
Ctrl+F6 to exit CLI focus
Copy
Paste

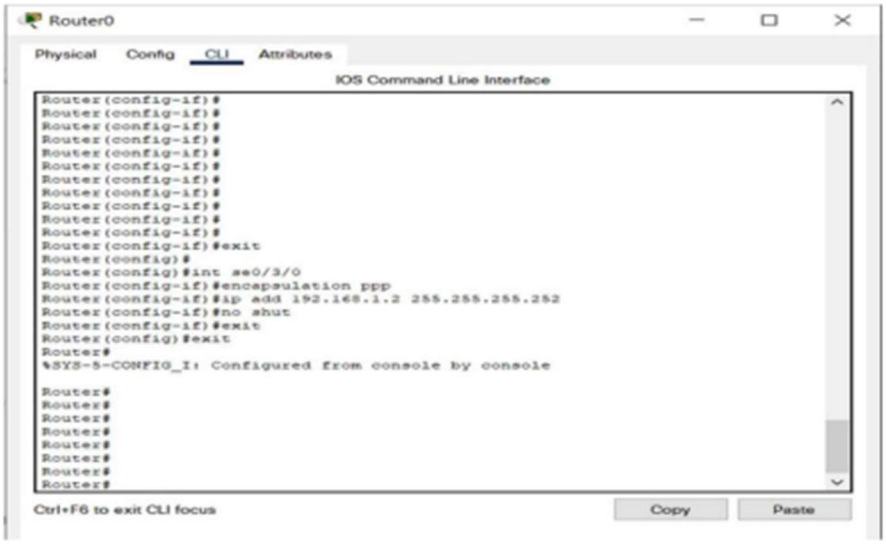

Router1
Physical
Config
CLI
Attributes
IOS Command Line Interface
Press RETURN to get started!
Router>enable
Router#show controllers se0/3/0
Interface Serial0/3/0
Hardware is PowerQUICC MPC860
DTE V.35 TX and RX clocks detected
idb at 0x81081AC4, driver data structure at 0x81084AC0
SCC Registers:
General [GSMR]=0x2:0x00000000, Protocol-specific [PSMR]=0x8
Events [SCCE]=0x0000, Mask [SCCM]=0x0000, Status [SCCS]=0x00
Transmit on Demand [TODR]=0x0, Data Sync [DSR]=0x7E7E
Interrupt Registers:
Config [CICR]=0x00367F80, Pending [CIPR]=0x0000C000
Mask [CIMR]=0x00200000, In-srv [CISR]=0x00000000
Command register [CR]=0x580
Port A [PADIR]=0x1030, [PAPAR]=0xFFFF
[PAODR]=0x0010, [PADAT]=0xCBFF
Port B [PBDIR]=0x09C0F, [PBPAR]=0x0800E
[PBODR]=0x00000, [PBDAT]=0x3FFFD
Port C [PCDIR]=0x00C, [PCPAR]=0x200
[PCSO]=0xC20, [PCDAT]=0xDF2, [PCINT]=0x00F
Receive Ring
rmd(68012830): status 9000 length 60C address 3B6DAC4
rmd(68012838): status B000 length 60C address 3B6D444
Transmit Ring
--More--
Ctrl+F6 to exit CLI focus
Copy
Paste

4. Configure the routers with the following parameters

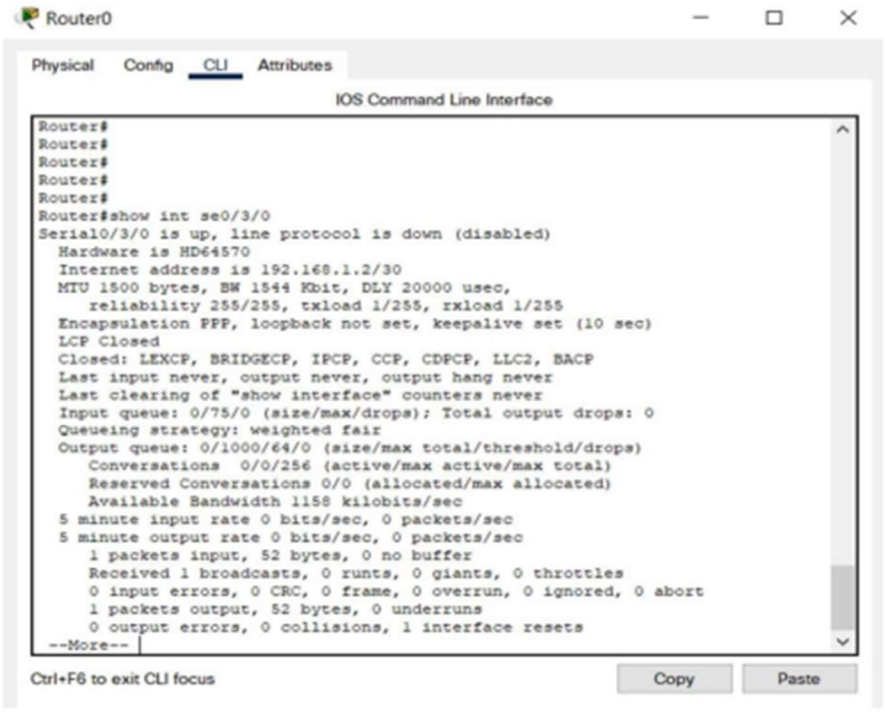
Router0 being the DCE, clock rate has to be configured on Router0 serial 0/3/0 interface.



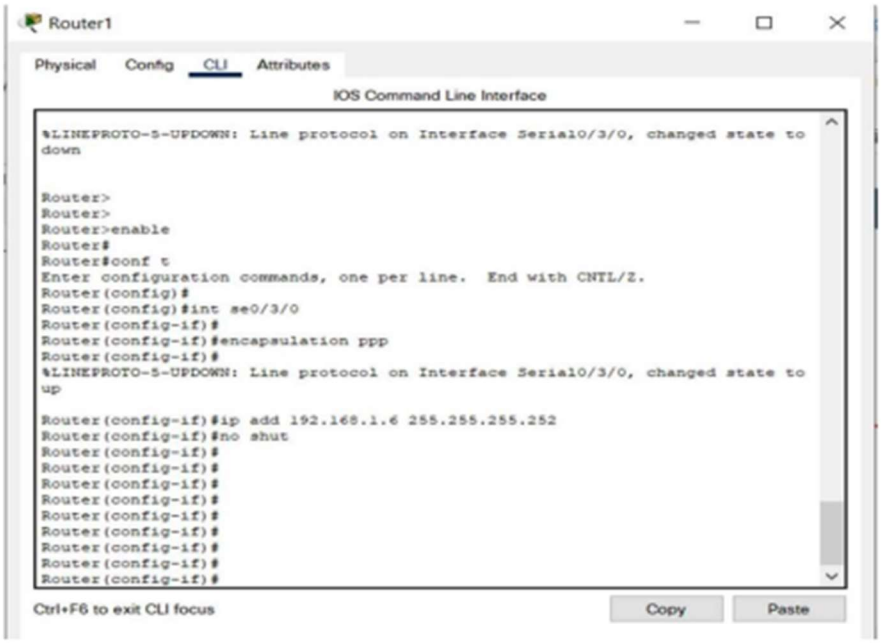
5. Then, configure PPP encapsulation and IP address on Router0 serial 0/3/0 interface. The **encapsulation ppp** configures PPP protocol on the serial interface. Router0 being the DCE side of the serial link, the 192.168.1.3 /30 IP address is configured on Router0 serial 0/3/0 interface. Don't forget to enable the interface with a no shutdown command.



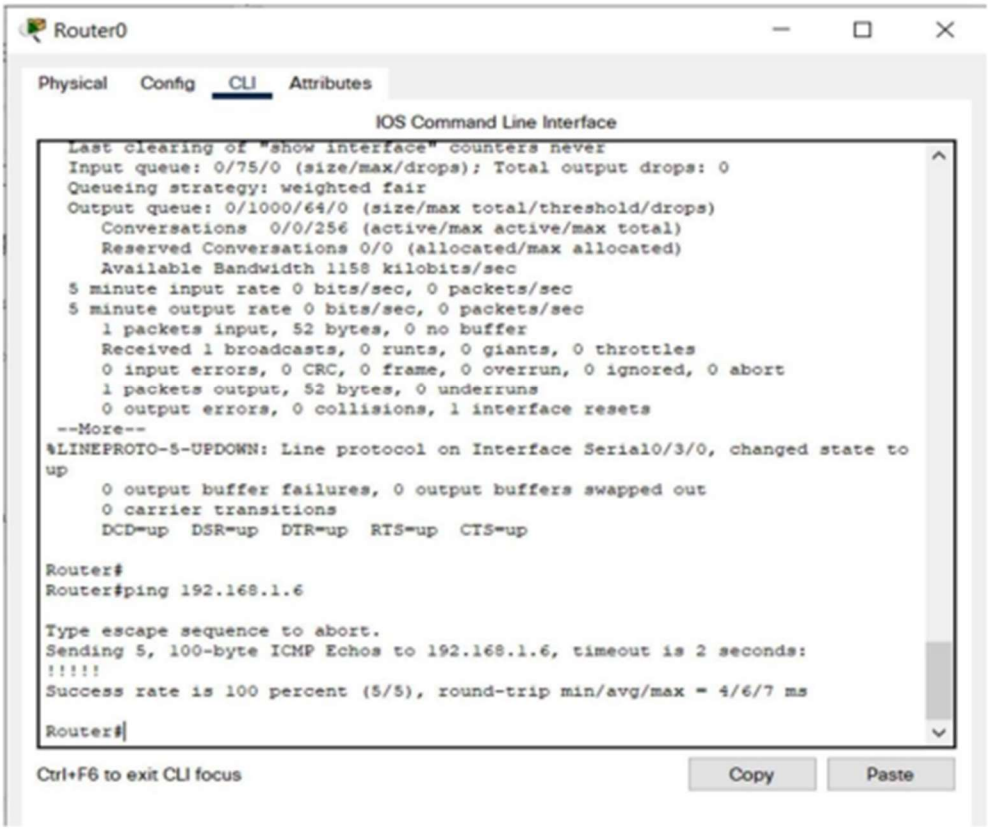
6. The show interfaces serial 0/3/0 confirms that PPP encapsulation is enabled on the interface : Encapsulation PPP, loopback not set, keepalive set (10 sec)



7. Finally, configure PPP encapsulation and IP address on Router1 serial 0/3/0 interface. The link comes up as both routers are correctly configured.



8. NOW CHECK THE CONNECTION BY PINGING EACH OTHER.First we go to Router0 and ping with Router1:



Now we go to Router1 and test the network by pinging the Router0 interface.

Result:

Hence successfully, configured PPP using routers in Cisco Packet Tracer.