

**PRINCE SHRI VENKATESHWARA
PADMAVATHY ENGINEERING COLLEGE
PONMAR, CHENNAI - 600 048.**



**DEPARTMENT OF INFORMATION TECHNOLOGY
(B.Tech. - V SEMESTER)**

**CS8581 –NETWORKS LABORATORY
(2022-2023)**

NAME : _____

REG. NO : _____

BRANCH : _____

COURSE : _____

**PRINCE SHRI VENKATESHWARA PADMAVATHY
ENGINEERING COLLEGE
PONMAR, CHENNAI- 600 048.**

Name :
Register No :
Semester :
Branch :

Certified that this is a Bonafide Record of the work done by
the above student in the **CS8581-NETWORKS LAB** during the
academic year **2022 – 2023**.

Signature of Faculty In-Charge

Signature of Principal

Submitted for Practical Examination held on

Internal Examiner

External Examiner

INDEX

EXPT NO.	DATE	PROGRAM	PAGE NO.	SIGNATURE
1.	12.08.2022	Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine.		
2.	26.08.2022	HTTP web client program to download a web page using TCP sockets		
3.a.	26.08.2022	Echo client and echo server Applications using TCP		
3.b.	02.09.2022	Chat Applications using TCP		
3.c.	02.09.2022	File Transfer Applications using TCP		
4.	09.09.2022	Simulation of DNS using UDP sockets		
5.	16.09.2022	simulating ARP /RARP protocols		
6.	23.09.2022	Study of Network simulator (NS)		
7.	30.09.2022	Simulation of Congestion Control Algorithms using NS		
8.	30.09.2022	Study of TCP/UDP performance using Simulation tool		
9.	14.10.2022	Simulation of Distance Vector Routing algorithm		
10.	14.10.2022	Simulation Link State Routing algorithm		
11.	21.10.2022	Performance evaluation of Routing protocols using Simulation tool.		
12.	28.10.2022	Simulation of error correction code (like CRC).		

CS8581 Networks Lab -Syllabus

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine.
2. Write a HTTP web client program to download a web page using TCP sockets.
3. Applications using TCP sockets like:
 - Echo client and echo server
 - Chat
 - File Transfer
4. Simulation of DNS using UDP sockets.
5. Write a code simulating ARP /RARP protocols.
6. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.
7. Study of TCP/UDP performance using Simulation tool.
8. Simulation of Distance Vector/ Link State Routing algorithm.
9. Performance evaluation of Routing protocols using Simulation tool.
10. Simulation of error correction code (like CRC).

INTRODUCTION TO SOCKET PROGRAMMING

AIM:

To study about the basics of Socket in Network Programming

Network Programming

Network programming involves writing programs that communicate with other programs across a computer N/W. One program is normally called the client, and the other the server. Common examples in the TCP/IP are web clients (browsers) & Web Servers, FTP clients & server and Telnet clients & servers.

To facilitate communication between unrelated processes, and to standardize network programming, an API is needed. There are two such APIs:

1. Sockets, sometimes called „Berkeley Sockets“
2. XTI (X/open transport interface)

Socket

In TCP/IP, an addressable point that consists of an IP address and a TCP or UDP port member that provides application with access to TCP/IP protocol is called Socket.

A socket is an abstraction that represents an endpoint of communication. The operations that can be performed on a socket include control operations (such as associating a port number with the socket, initiating or accepting a connection on the socket, or destroying the socket), data transfer operations (such as writing data through the socket to some other application, or reading data from some other application through the socket) and status operations (such as finding the IP address associated with the socket). The complete set of operations that can be performed on a socket constitutes the **Sockets API** (Application Programming Interface).

Structures

Structures are used in socket programming to hold information about the address. The generic socket address structure is defined below:

Structsockaddr

```
{
  unsigned short sa_family; /* address family */
  char sa_data[14]; /* 14 bytes of protocol address */
};
```

IPv4 Socket Address Structure

This structure is also called as “Internet socket address structure”. It is defined by including the <netinet/in.h> header.

```
struct in_addr {
  in_addr_t s_addr; /* 32-bit IPv4 address, network byte ordered */
  struct sockaddr_in{
```

```

};
unit_tsin_len; /* length of structure (16 byte) */
sa_family_tsin_family; /*AF_INET*/
in_port_tsin_port; /* 16-bit TCP or UDP port number */
/* network byte ordered */
struct in_addr sin_addr; /*32-bit Ipv4 address, network byte ordered */
char sin_zero[8]; /* unused – initialize to all zeroes */
};

```

Important functions

1.socket()

This function is called by both TCP server and client process to create an empty socket.

```

#include <sys/socket.h>
int socket (int family, int type, int protocol);

```

family: specifies the protocol family and is one of the constants below:

Family	description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing sockets
AF_KEY	Key sockets

type: indicates communications semantics

SOCK_STREAM - stream socket

SOCK_DGRAM - datagram socket

SOCK_RAW - raw socket

protocol: set to 0 except for raw sockets.

on success: socket descriptor (a small nonnegative integer), on error: -1

2. bind()

Returns The bind function assigns a local protocol address to a socket. The protocol address is the combination of either a 32-bit IPV4 address or a 128-bit IPV6 address, along with a 16-bit TCP or UDP port number.

```

#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);

```

sockfd: a socket descriptor returned by the socket function.

*addrlen: the size of the socket address structure.

Returns on success: 0, on error: -1

3. connect()

The connect function is used by a TCP client to establish a connection with a TCP server.

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

```
int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

sockfd: a socket descriptor returned by the socket function

*servaddr: a pointer to a socket address structure

addrlen: the size of the socket address structure

Returns on success: 0, on error: -1

4. listen()

The listen function is called only by a TCP server to convert an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to its socket.

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

```
int myaddr: a pointer to a protocol-specific address.
```

```
listen (int sockfd, int backlog);
```

sockfd: a socket descriptor returned by the socket function.

backlog: maximum number of connections that the kernel should queue for this socket.

Returns on success: 0, on error: -1

5. accept()

The accept function is called by the TCP server to return the next completed connection from the front of the completed connection queue.

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

```
int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

sockfd: This is the same socket descriptor as in listen call.

*cliaddr: used to return the protocol address of the connected peer process

*addrlen: length of the address.

Returns on success: a new (connected) socket descriptor, on error: -1

6. close()

The close function is used to close a socket and terminate a TCP connection.

```
#include <unistd.h>
```

```
int close (int sockfd);
```

sockfd: This socket descriptor is no longer useable.

Returns on success: 0, on error: -1

7. read()

The read function is used to receive data from the specified socket.

```
#include <unistd.h>
```

```
ssize_t read(int sockfd, const void * buf, size_t nbytes);
```

sockfd: a socket descriptor returned by the socket function.

buf: buffer to store the data.

nbytes: size of the buffer

Returns: number of bytes read if OK, 0 on EOF, -1 on error

8. write()

The write function is used to send the data through the specified socket.

```
#include <unistd.h>
```

```
ssize_t write(int sockfd, const void * buf, size_t nbytes);
```

sockfd: a socket descriptor returned by the socket function.

buf: buffer to store the data.

nbytes: size of the buffer

Returns: number of bytes written if OK, 0 on EOF, -1 on error

9. sendto()

This function is similar to the write function, but additional arguments are required.

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

```
ssize_t sendto(int sockfd, const void *buff, size_t nbytes, int flag,
```

```
const struct sockaddr *to, socklen_t addrlen);
```

sockfd – socket descriptor

*buff – pointer to buffer to write from.

nbytes – number of bytes to write.

to – socket address structure containing the protocol address of where the data is to be sent.

addrlen – size of the socket address structure

Returns: number of bytes read or written if OK, -1 on error

10. recvfrom()

This function is similar to the read function, but additional arguments are required.

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

```
ssize_t recvfrom(int sockfd, void *buff, size_t nbytes, int flag,  
struct sockaddr *from, socklen_t *addrlen);
```

sockfd – socket descriptor

*buff – pointer to buffer to read.

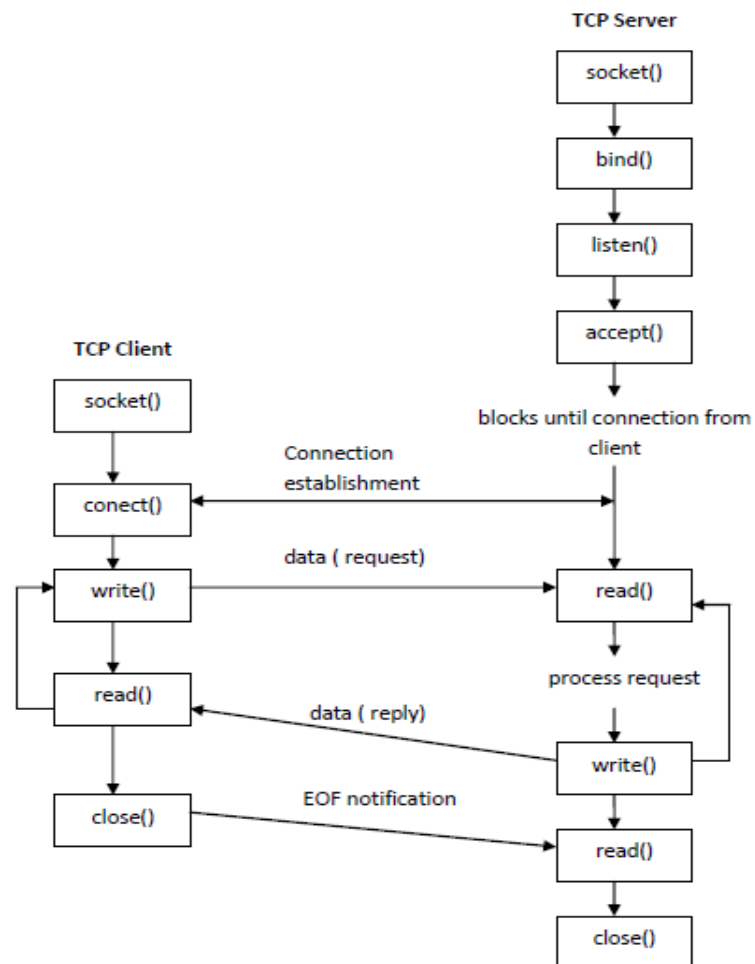
nbytes – number of bytes to read.

addrlen – size of the socket address structure

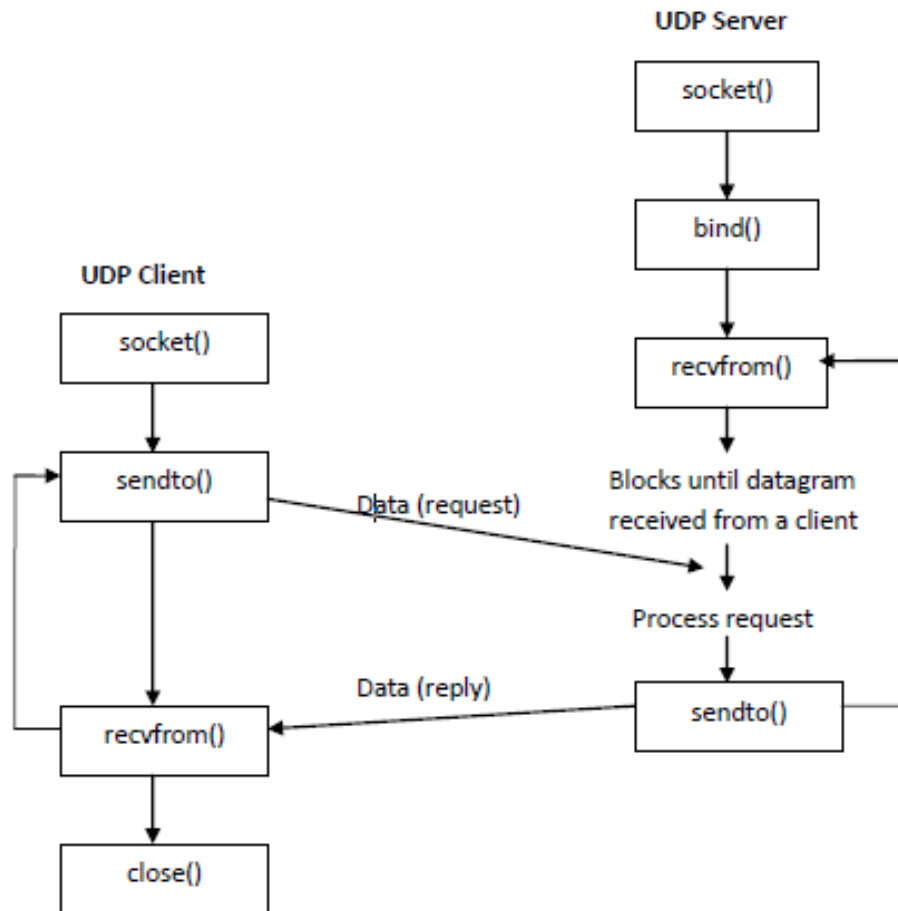
from – socket address structure of who sent the datagram.

Returns: number of bytes read or written if OK, -1 on error

Socket functions for connection-oriented communication



Socket functions for connection-less communication



Program to create a socket

```

/* Ex-1 Program To Create a Socket */
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
int main()
{
    int sockfd1, sockfd2;
    sockfd1=socket(AF_INET, SOCK_STREAM, 0);
    sockfd2=socket(PF_INET, SOCK_DGRAM, 0);
    if(sockfd1==-1)
    {
        printf("socket1 not created\n");
    }
}

```

```

    }
else
{
printf("\nsocket1 created and \t socket1 File Descriptor value is %d \n",sockfd1);
if(sockfd2== -1)
{
printf("\nsocket2 creation error");
}
else
{
printf("\nSocket2 created and \t socket2 descriptor value is %d\n",sockfd2);
}}
}

```

Program to Bind a socket

```

/* Ex-2 Program to Bind a Socket */
#include<stdio.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#define PORTNO 2000
int main()
{
int sockfd,i=PORTNO;
struct sockaddr_in myaddr;
if((sockfd=socket(AF_INET,SOCK_STREAM,0))== -1)
{
printf("\nsocket creation error\n");
}
myaddr.sin_family=AF_INET;
myaddr.sin_port=htons(PORTNO);
myaddr.sin_addr.s_addr=INADDR_ANY;
memset(&(myaddr.sin_zero),'\0',8);
if(bind(sockfd,(struct sockaddr *)&myaddr,sizeof(struct sockaddr))!= -1)
{
printf("\nsocket is Binded port %d\n",i);
}
else
{
printf("\nBinding error");
}
}

```

Program to implement listen() system call

```

/* Ex-3 Program to implement LISTEN system call */
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>

```

```

#include<netinet/in.h>
#include<stdlib.h>
#define PORT 3550
#define BACKLOG 12
main()
{
    int fd;
    struct sockaddr_in server;
    struct sockaddr_in client;
    int sin_size;
    int x;
    if((fd=socket(AF_INET,SOCK_STREAM,0))==-1)
    {
        printf("\nsocket() error\n");
        exit(-1);
    }
    server.sin_family=AF_INET;
    server.sin_port=htons(PORT);
    server.sin_addr.s_addr=INADDR_ANY;
    bzero(&(server.sin_zero),8);
    if(bind(fd,(struct sockaddr*)&server,sizeof(struct sockaddr))==-1)
    {
        printf("\n bind() error\n");
        exit(-1);
    }
    x=listen(fd,BACKLOG);
    if(x==-1)
    {
        printf("\nlisten() error\n");
        exit(-1);
    }
    else
    {
        printf("Server is in listening mode\n");
    }
    close(fd);
}

```

Program for accept() system call

```

/* Ex-4 Program to implement ACCEPT system calls */
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<unistd.h>

```

```

#include<string.h>
#include<net/if_arp.h>
#define PORT 2586
#define BACKLOG 4
main()
{
    int fd,fd1;
    struct sockaddr_in server;
    struct sockaddr_in client;
    int sin_size;
    if((fd=socket(AF_INET,SOCK_STREAM,0))==-1)
    {
        printf("\nsocket() error\n");
        exit(-1);
    }
    server.sin_family=AF_INET;
    server.sin_port=htons(PORT);
    server.sin_addr.s_addr=INADDR_ANY;
    bzero(&(server.sin_zero),8);
    if(bind(fd,(struct sockaddr*)&server,sizeof(struct sockaddr))==-1)
    {
        printf("\nbind() error\n");
        exit(-1);
    }
    if(listen(fd,BACKLOG)==-1)
    {
        printf("\nlisten() error\n");
        exit(-1);
    }
    printf("\nServer is accept mode\n");
    while(1)
    {
        sin_size=sizeof(struct sockaddr_in);
        if((fd1=accept(fd,(struct sockaddr*)&client,&sin_size))==-1)
        {
            printf("\naccept() error\n");
            exit(-1);
        }
        else
        {
            printf("\nserver is in accept mode");
            printf("\nyou got a connection from %s\n",inet_ntoa(client.sin_addr));
            close(fd1);
        }
    }
}

```

OUTPUT:**/* Ex-1 Program To Create a Socket */**

```
[prince03@prince ~]$ vi createsocket.c
[prince03@prince ~]$ cc createsocket.c
[prince03@prince ~]$ ./a.out
```

```
socket1 created and      socket1 File Descriptor value is 3
Socket2 created and      socket2 descriptor value is 4
```

/* Ex-2 Program to Bind a Socket */

```
[prince03@prince ~]$ vi bindsocket.c
[prince03@prince ~]$ cc bindsocket.c
[prince03@prince ~]$ ./a.out
```

```
socket is Binded port 2000
```

/* Ex-3 Program to implement LISTEN system call */

```
[prince03@prince ~]$ vi listen.c
[prince03@prince ~]$ cc listen.c
[prince03@prince ~]$ ./a.out
```

```
Server is in listening mode
```

/* Ex-4 Program to implement ACCEPT system calls */

```
[prince03@prince ~]$ vi accept.c
[prince03@prince ~]$ cc accept.c
[prince03@prince ~]$ ./a.out
```

```
server is in accept mode
```

EXP NO : 1
DATE : 12.08.2022

Commands

AIM:

To Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute

1.Tcpdump

The tcpdump utility allows you to capture packets that flow within your network to assist in network troubleshooting. The following are several examples of using tcpdump with different options. Traffic is captured based on a specified filter.

Options Description

-D Print a list of network interfaces.

Specify an interface on which

-i to capture.

-c Specify the number of packets to receive.

-v, -vv, -vvv Increase the level of detail (verbosity).

-w Write captured data to a file.

-r Read captured data from a file.

Many other options and arguments can be used with tcpdump. The following are some specific examples of the power of the tcpdump utility.

1. Display traffic between 2 hosts

To display all traffic between two hosts (represented by variables host1 and host2):

```
# tcpdump host host1 and host2
```

2. Display traffic from a source or destination host only

To display traffic from only a source (src) or destination (dst) host: # tcpdump src host

```
# tcpdump dst host
```

3. Display traffic for a specific protocol

Provide the protocol as an argument to display only traffic for a specific protocol, for example tcp, udp, icmp, arp:

```
# tcpdump protocol
```

For example to display traffic only for the tcp traffic :

```
# tcpdump tcp
```

4. Filtering based on source or destination port

To filter based on a source or destination port:#

```
tcpdump src port ftp
```

```
# tcpdump dst port http
```

2.Netstat

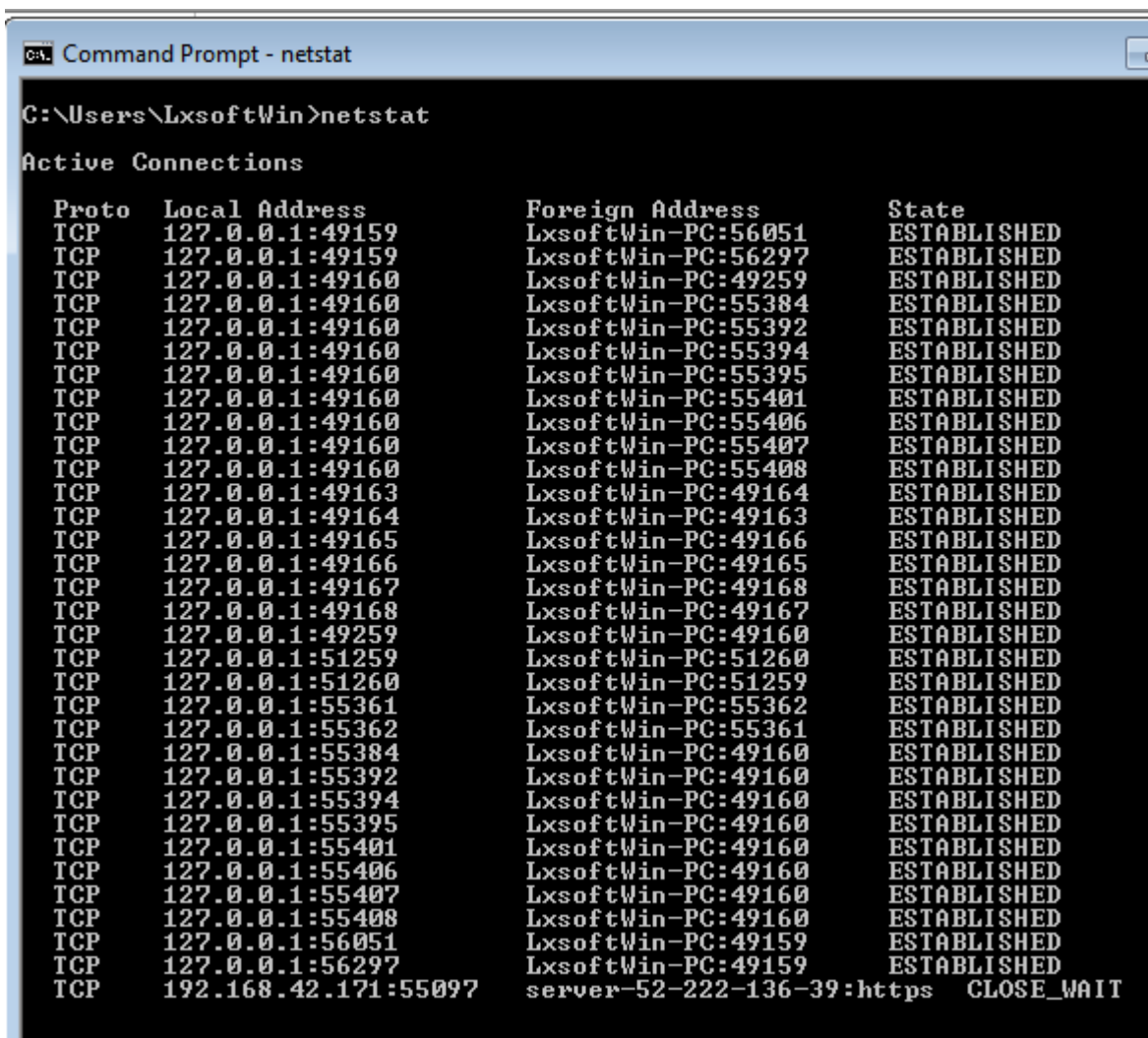
Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems. Netstat provides information and statistics about protocol sinuse and current TCP/IP network connections. The Windows help screen (analogous to a Linux or UNIX for netstat reads as follows:

Displays protocol statistics and current TCP/IP network

connections.NETSTAT -a -b -e -n -o -p proto -r -s -v interval

-a	Displays all connections and listening ports.
-b	Displays the executable involved in creating each connection or listening port. In some cases well-known executables host multiple independent components, and in these cases the sequence of components involved in creating the connection or listening port is displayed. In this case the executable name is in [] at the bottom, on top is the component it called, and so forth until TCP/IP was reached. Note that this option can be time-consuming and will fail unless you have sufficient permissions.
-e	Displays Ethernet statistics. This may be combined with the -s option.
-n	Displays addresses and port numbers in numerical form.
-o	Displays the owning process ID associated with each connection.
-p proto	Shows connections for the protocol specified by proto; proto may be any of: TCP, UDP, TCPv6, or UDPv6. If used with the -s option to display per-protocol statistics, proto may be any of: IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP, or UDPv6.

-r	Displays the routing table.
-s	Displays per-protocol statistics. By default, statistics are shown for IP, IPv6, ICMP,ICMPv6, TCP, TCPv6, UDP, and UDPv6; the -p option may be used to specify a subset of the default.
-v	When used in conjunction with -b, will display sequence of components involved in creating the connection or listening port for all executables.
interval	Redisplays selected statistics, pausing interval seconds between each display. Press CTRL+C to stop displaying statistics. If omitted, net stat will print the current configuration information once.



```

C:\> Command Prompt - netstat

C:\Users\LxsoftWin>netstat

Active Connections

Proto Local Address          Foreign Address         State
TCP    127.0.0.1:49159          LxsoftWin-PC:56051     ESTABLISHED
TCP    127.0.0.1:49159          LxsoftWin-PC:56297     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:49259     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:55384     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:55392     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:55394     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:55395     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:55401     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:55406     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:55407     ESTABLISHED
TCP    127.0.0.1:49160          LxsoftWin-PC:55408     ESTABLISHED
TCP    127.0.0.1:49163          LxsoftWin-PC:49164     ESTABLISHED
TCP    127.0.0.1:49164          LxsoftWin-PC:49163     ESTABLISHED
TCP    127.0.0.1:49165          LxsoftWin-PC:49166     ESTABLISHED
TCP    127.0.0.1:49166          LxsoftWin-PC:49165     ESTABLISHED
TCP    127.0.0.1:49167          LxsoftWin-PC:49168     ESTABLISHED
TCP    127.0.0.1:49168          LxsoftWin-PC:49167     ESTABLISHED
TCP    127.0.0.1:49259          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:51259          LxsoftWin-PC:51260     ESTABLISHED
TCP    127.0.0.1:51260          LxsoftWin-PC:51259     ESTABLISHED
TCP    127.0.0.1:55361          LxsoftWin-PC:55362     ESTABLISHED
TCP    127.0.0.1:55362          LxsoftWin-PC:55361     ESTABLISHED
TCP    127.0.0.1:55384          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:55392          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:55394          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:55395          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:55395          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:55401          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:55406          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:55407          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:55408          LxsoftWin-PC:49160     ESTABLISHED
TCP    127.0.0.1:56051          LxsoftWin-PC:49159     ESTABLISHED
TCP    127.0.0.1:56297          LxsoftWin-PC:49159     ESTABLISHED
TCP    192.168.42.171:55097     server-52-222-136-39:https CLOSE_WAIT

```

Syntax

netstat [-a] [-e] [-n] [-o] [-p Protocol] [-r] [-s] [Interval]

Parameters

- a Display all active TCP connections and the TCP and UDP ports on which the computer is listening.
- e Displays Ethernet statistics, such as the number of bytes and packets sent and received. This parameter can be combined with -s.
- n Displays active TCP connections, however, addresses and port numbers are expressed numerically and no attempt is made to determine names.

Displays active TCP connections and includes the processID (PID) for each connection. You can find the application based on the PID on the Processes tab in Windows Task Manager. This parameter can be combined with -a, -n, and -p.

Shows connections for the protocol specified by Protocol. In this Protocol can be tcp, udp, tcpv6, or udpv6. If this parameter is used with -s to display statistics by protocol, Protocol can be tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6, or ipv6.

Display statistics by protocol. By default, statistics are shown for the UDP, ICMP, and IP protocols. If the IPv6 protocol for Windows XP is installed, statistics are shown for the TCP over IPv6, UDP over IPv6, ICMPv6, and IPv6 protocols. The -p parameter can be used to specify a set of protocols.
- r Displays the contents of the IP routing table. This is equivalent to the route print command.
- /? - Displays help at the command prompt.

3.Ifconfig

In Windows, **ipconfig** is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address

information of a Windows computer. It also allows some control over active [TCP/IP](#) connections. **Ipconfig** replaced the older winipcfg utility.

Using ipconfig

From the command prompt, type **ipconfig** to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapters.

Parameter:

	parameter, ipconfig displays only the IP address, subnet mask, and default gateway values for each adapter. Adapters can represent physical interfaces, such as installed network adapters, or logical interfaces, such as dial-up connections.
/renew[Adapter]	Renews DHCP configuration for all adapters (if an adapter is not specified) or for a specific adapter if the Adapter parameter is included. This parameter is available only on computers with adapters that are configured to obtain an IP address automatically. To specify an adapter name, type the adapter name that appears when you use ipconfig without parameters.
/release[Adapter]	Sends a DHCPRELEASE message to the DHCP server to release the current DHCP configuration and discard the IP address configuration for either all adapters (if an adapter is not specified) or for a specific adapter if the Adapter parameter is included. This parameter disables TCP/IP for adapters configured to obtain an IP address automatically. To specify an adapter name, type the adapter name that appears when you use ipconfig without parameters.
/flushdns	Flushes and resets the contents of the DNS client resolver cache. During DNS troubleshooting, you can use this procedure to discard negative cache entries from the cache, as well as any other entries that have been added dynamically.
/displaydns	Displays the contents of the DNS client resolver cache, which includes both entries preloaded from the local Hosts file and any recently obtained resource records for name queries resolved by the computer. The DNS Client service uses this information to resolve frequently queried names quickly, before querying its configured DNS servers.
/registerdns	Initiates manual dynamic registration for the DNS names and IP addresses that are configured on the computer. You can use this parameter to troubleshoot a failed DNS name registration or resolve a dynamic update problem between a client and the DNS server without rebooting the client computer. The DNS settings in the advanced properties of the TCP/IP protocol determine which names are registered in DNS.

/showclassid	Adapter Displays the DHCP class ID for a specified adapter. To see the DHCPclass ID for all adapters, use the asterisk (*) wildcard character in place of Adapter. This parameter is available only on computers with adapters that are configured to obtain an IP address automatically.
/setclassid	Adapter [ClassID] Configures the DHCP class ID for a specified adapter. To set the DHCP class ID for all adapters, use the asterisk (*) wildcard character in place of Adapter. This parameter is available only on computers with adapters that are configured to obtain an IP address automatically. If a DHCPclass ID is not specified, the current class ID is removed.

Syntax

```
ipconfig [/all] [/renew [Adapter]] [/release [Adapter]] [/flushdns]
[/displaydns] [/registerdns] [/showclassid Adapter] [/setclassid Adapter
[ClassID]]
```

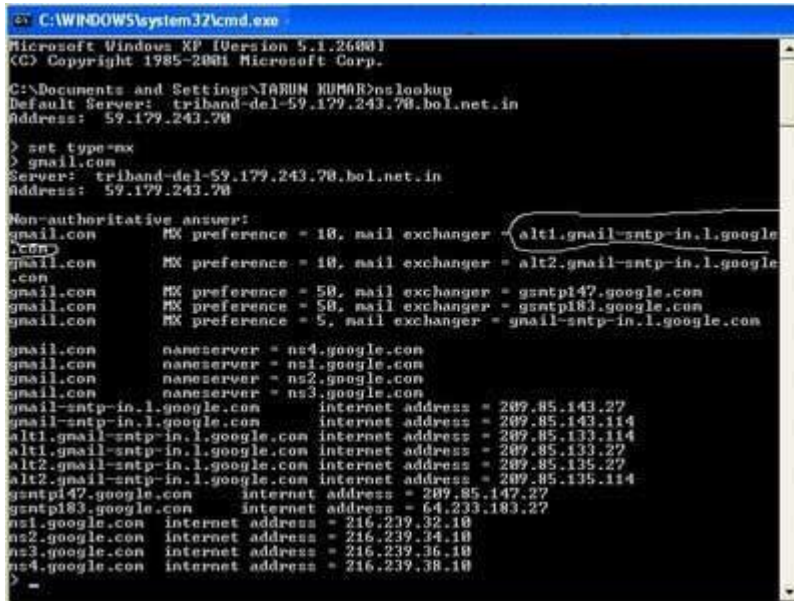
Examples:

Ipconfig	To display the basic TCP/IP configuration for all adapters
ipconfig /all	To display the full TCP/IP configuration for all adapters

ipconfig /renew "Local AreaConnection"	To renew a DHCP-assigned IP address configuration for only the Local Area Connection adapter
ipconfig /flushdns	To flush the DNS resolver cache when troubleshooting DNS name resolution problems
ipconfig /showclassid Local	To display the DHCP class ID for all adapters with names that start with Local
ipconfig /setclassid "Local AreaConnection" TEST	To set the DHCP class ID for the Local Area Connection adapter to TEST

1. Nslookup

The **nslookup** (which stands for *name server lookup*) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System.



```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\TARUN KUMAR>nslookup
Default Server: triband-del-59.179.243.78.bol.net.in
Address: 59.179.243.78

> set type=mx
> gmail.com
Server: triband-del-59.179.243.78.bol.net.in
Address: 59.179.243.78

Non-authoritative answer:
gmail.com      MX preference = 10, mail exchanger = alt1.gmail-smtp-in.l.google
               .com
gmail.com      MX preference = 10, mail exchanger = alt2.gmail-smtp-in.l.google
               .com
gmail.com      MX preference = 50, mail exchanger = smtp147.google.com
gmail.com      MX preference = 50, mail exchanger = smtp183.google.com
gmail.com      MX preference = 5, mail exchanger = gmail-smtp-in.l.google.com

gmail.com      nameserver = ns4.google.com
gmail.com      nameserver = ns1.google.com
gmail.com      nameserver = ns2.google.com
gmail.com      nameserver = ns3.google.com
gmail-smtp-in.l.google.com internet address = 209.85.143.27
gmail-smtp-in.l.google.com internet address = 209.85.143.114
alt1.gmail-smtp-in.l.google.com internet address = 209.85.133.114
alt1.gmail-smtp-in.l.google.com internet address = 209.85.133.27
alt2.gmail-smtp-in.l.google.com internet address = 209.85.135.27
alt2.gmail-smtp-in.l.google.com internet address = 209.85.135.114
smtp147.google.com internet address = 209.85.147.27
smtp183.google.com internet address = 64.233.183.27
ns1.google.com internet address = 216.239.32.10
ns2.google.com internet address = 216.239.34.10
ns3.google.com internet address = 216.239.36.10
ns4.google.com internet address = 216.239.38.10
  
```

2. traceroute

Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination.

Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values.

The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop.

tracert www.google.com

With the tracert command shown above, we're asking tracert to show us the path from the local computer all the way to the network device with the hostname www.google.com.

Tracing route to www.l.google.com [209.85.225.104] over a maximum of 30 hops:

```

  0  <1 ms <1 ms <1 ms 10.1.0.1
  1  35 ms 19 ms 29 ms 98.245.140.1
  
```

3 11 ms 27 ms 9 ms te-0-3.dnv.comcast.net [68.85.105.201]

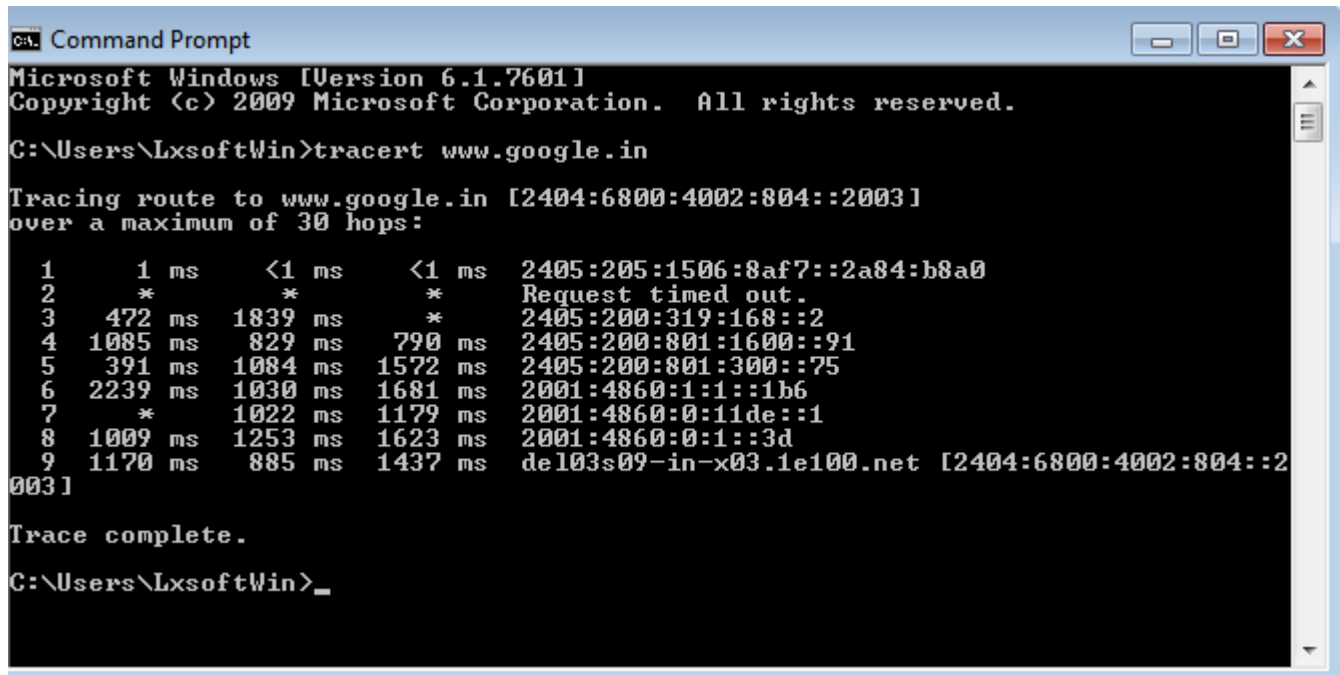
...

13 81 ms 76 ms 75 ms 209.85.241.37

14 84 ms 91 ms 87 ms 209.85.248.102

15 76 ms 112 ms 76 ms iy-f104.1e100.net [209.85.225.104]

Trace complete.



```

C:\Users\LxsoftWin>tracert www.google.in

Tracing route to www.google.in [2404:6800:4002:804::2003]
over a maximum of 30 hops:
  0  1 ms  <1 ms  <1 ms  2405:205:1506:8af7::2a84:b8a0
  1  *      *      *      Request timed out.
  2  472 ms 1839 ms *      2405:200:319:168::2
  3  1085 ms 829 ms 790 ms 2405:200:801:1600::91
  4  391 ms 1084 ms 1572 ms 2405:200:801:300::75
  5  2239 ms 1030 ms 1681 ms 2001:4860:1:1::1b6
  6  *      1022 ms 1179 ms 2001:4860:0:11de::1
  7  1009 ms 1253 ms 1623 ms 2001:4860:0:1::3d
  8  1170 ms 885 ms 1437 ms del03s09-in-x03.1e100.net [2404:6800:4002:804::2003]

Trace complete.

C:\Users\LxsoftWin>_

```

EXP NO : 2
DATE : 26.08.2022

Web client program to download a web page using TCP sockets.

AIM:

To write a HTTP web client program to download a web page using TCP sockets.

ALGORITHM:

1. Start the process
2. Read the web page which we want to download and upload
3. Using socket system calls make necessary operations to download the web page.
4. The web page content will be displayed.
5. Stop the process.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h> /* close() */
#include <sys/socket.h>
#include <netdb.h>

int main(void)
{
    int sock;
    char host[] = "www.google.com";
    char port[] = "80";
    struct addrinfo hints, *res;
    char message[] = "GET / HTTP/1.1\nHost: www.google.com\n\n";
    unsigned int i;
    char buf[1024];
    int bytes_read;
    int status;

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    status = getaddrinfo(host, port, &hints, &res);
    if (status != 0) {
        perror("getaddrinfo");
        return 1;
    }
    sock = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    if (sock == -1) {
        perror("socket");
        return 1;
    }
    status = connect(sock, res->ai_addr, res->ai_addrlen);
    if (status == -1) {
        perror("connect");
        return 1;
    }
}
```

```

freeaddrinfo(res);
send(sock, message, strlen(message), 0);

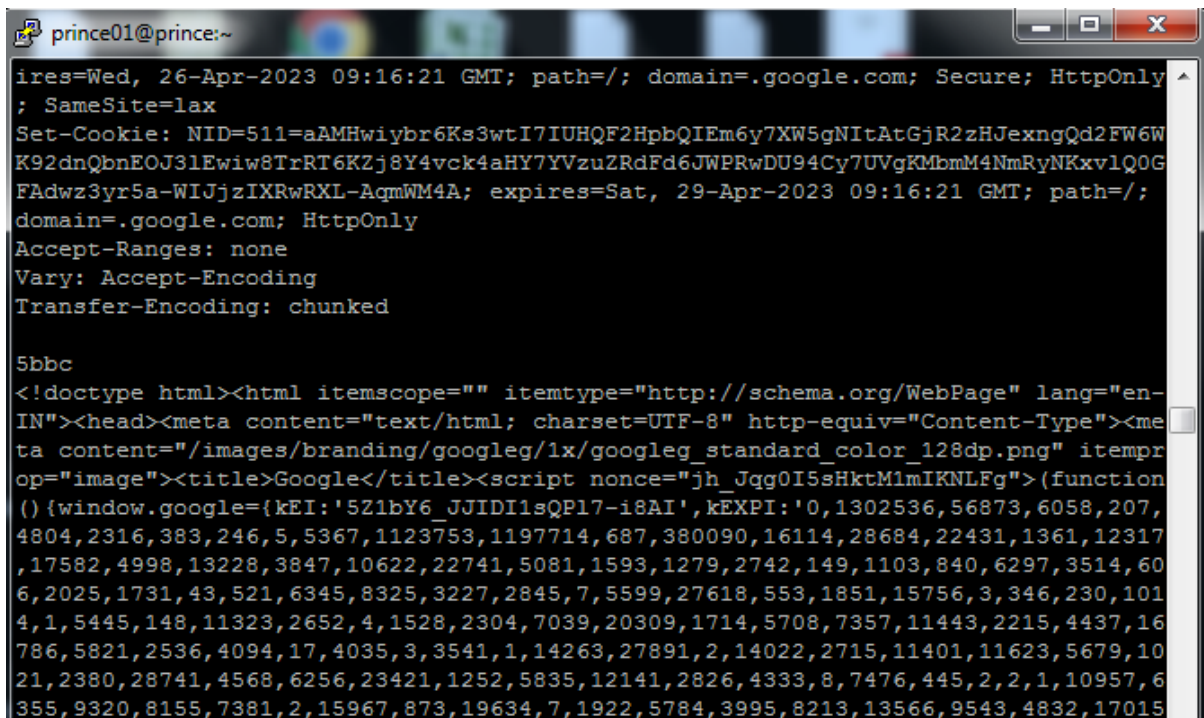
do {
    bytes_read = recv(sock, buf, 1024, 0);
    if (bytes_read == -1) {
        perror("recv");
    }
    else {
        printf("%.5s", bytes_read, buf);
    }
} while (bytes_read > 0);

close(sock);

return 0;
}

```

OUTPUT:



```

prince01@prince:~
ires=Wed, 26-Apr-2023 09:16:21 GMT; path=/; domain=.google.com; Secure; HttpOnly
; SameSite=lax
Set-Cookie: NID=511=aAMHwiYbr6Ks3wtI7IUHQF2HpbQIE6y7XW5gNIAtGjR2zHJexngQd2FW6W
K92dnQbnEOJ3lEwiw8TrRT6KZj8Y4vck4aHY7YVzuZRdFd6JWPRwDU94Cy7UVgKMbmM4NmRyNKxv1Q0G
FAdwz3yr5a-WIjzIXRwRXL-AqmWM4A; expires=Sat, 29-Apr-2023 09:16:21 GMT; path=/;
domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

5bbc
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-
IN"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><me
ta content="/images/branding/google/1x/google_standard_color_128dp.png" itempr
op="image"><title>Google</title><script nonce="jh_JqgOI5sHktM1mIKNLFg">(function
() {window.google={kEI:'5Z1bY6_JJIDIIsQP17-i8AI',kEXPI:'0,1302536,56873,6058,207,
4804,2316,383,246,5,5367,1123753,1197714,687,380090,16114,28684,22431,1361,12317
,17582,4998,13228,3847,10622,22741,5081,1593,1279,2742,149,1103,840,6297,3514,60
6,2025,1731,43,521,6345,8325,3227,2845,7,5599,27618,553,1851,15756,3,346,230,101
4,1,5445,148,11323,2652,4,1528,2304,7039,20309,1714,5708,7357,11443,2215,4437,16
786,5821,2536,4094,17,4035,3,3541,1,14263,27891,2,14022,2715,11401,11623,5679,10
21,2380,28741,4568,6256,23421,1252,5835,12141,2826,4333,8,7476,445,2,2,1,10957,6
355,9320,8155,7381,2,15967,873,19634,7,1922,5784,3995,8213,13566,9543,4832,17015

```

RESULT:

Thus the Web client program to download a web page using TCP sockets has been executed successfully.

EXP NO : 3(A)
DATE : 26.08.2022

TCP echo client server

AIM:

To write a program for TCP echo client server.

ALGORITHM:

SERVER:

- STEP 1: Start
- STEP 2: Declare the variables for the socket
- STEP 3: Specify the family, protocol, IP address and port number
- STEP 4: Create a socket using socket() function
- STEP 5: Bind the IP address and Port number
- STEP 6: Listen and accept the client's request for the connection
- STEP 7: Read the client's message
- STEP 8: Display the client's message
- STEP 9: Close the socket
- STEP 10: Stop

CLIENT:

- STEP 1: Start
- STEP 2: Declare the variables for the socket
- STEP 3: Specify the family, protocol, IP address and port number
- STEP 4: Create a socket using socket() function
- STEP 5: Call the connect() function
- STEP 6: Read the input message
- STEP 7: Send the input message to the server
- STEP 8: Display the server's echo
- STEP 9: Close the socket
- STEP 10: Stop

PROGRAM:

SERVER:

```
#include<stdio.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5035
int main(int argc,char**argv)

{
    int sockfd,newsockfd,length;
    struct sockaddr_in serv_addr,cli_addr;
    char buffer[4096];
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=INADDR_ANY;
    serv_addr.sin_port=htons(SERV_TCP_PORT);
    printf("&quot;\nStart&quot;");
    bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    printf("&quot;\nListening...&quot;");
```

```

printf("&quot;\n&quot;");
listen(sockfd,5);
clength=sizeof(cli_addr);
newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr,&clength);
printf("&quot;\nAccepted&quot;");
printf("&quot;\n&quot;");
read(newsockfd,buffer,4096);
printf("&quot;\nClient message:%s&quot;,buffer);
write(newsockfd,buffer,4096);
printf("&quot;\n&quot;");
close(sockfd);
return 0;
}

```

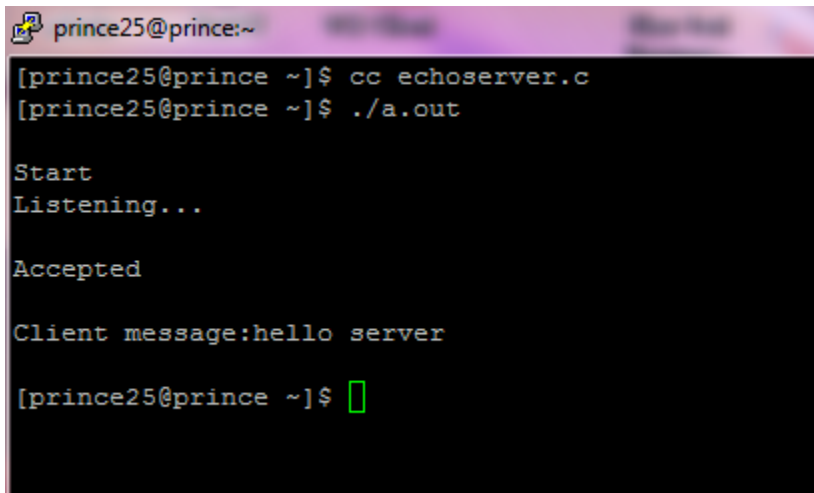
CLIENT:

```

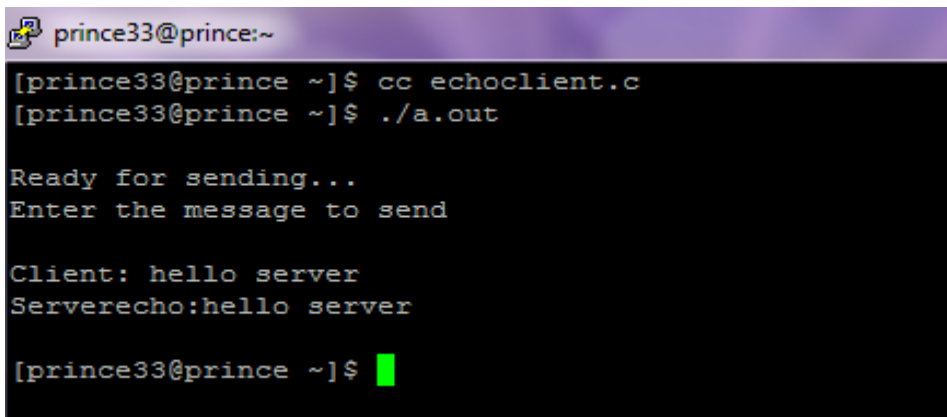
#include<stdio.h>;
#include<sys/types.h>;
#include<sys/socket.h>;
#include<netinet/in.h>;
#include<netdb.h>;
#define SERV_TCP_PORT 5035
int main(int argc,char*argv[])
{
    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[4096];
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    serv_addr.sin_family=AF_INET;

    serv_addr.sin_addr.s_addr=inet_addr("&quot;127.0.0.1&quot;");
    serv_addr.sin_port=htons(SERV_TCP_PORT);
    printf("&quot;\nReady for sending...&quot;");
    connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    printf("&quot;\nEnter the message to send\n&quot;");
    printf("&quot;\nClient: &quot;");
    fgets(buffer,4096,stdin);
    write(sockfd,buffer,4096);
    printf("&quot;Server echo:%s&quot;,buffer);
    printf("&quot;\n&quot;");
    close(sockfd);
    return 0;
}

```

OUTPUT:**SERVER:**A screenshot of a terminal window with a dark background and light-colored text. The prompt is 'prince25@prince:~'. The user enters 'cc echoserver.c' and then './a.out'. The program outputs 'Start', 'Listening...', 'Accepted', and 'Client message:hello server'. The prompt returns to the user.

```
prince25@prince:~  
[prince25@prince ~]$ cc echoserver.c  
[prince25@prince ~]$ ./a.out  
  
Start  
Listening...  
  
Accepted  
  
Client message:hello server  
  
[prince25@prince ~]$
```

CLIENT:A screenshot of a terminal window with a dark background and light-colored text. The prompt is 'prince33@prince:~'. The user enters 'cc echoclient.c' and then './a.out'. The program outputs 'Ready for sending...', 'Enter the message to send', 'Client: hello server', and 'Serverecho:hello server'. The prompt returns to the user.

```
prince33@prince:~  
[prince33@prince ~]$ cc echoclient.c  
[prince33@prince ~]$ ./a.out  
  
Ready for sending...  
Enter the message to send  
  
Client: hello server  
Serverecho:hello server  
  
[prince33@prince ~]$
```

RESULT:

Thus the Echo Clinet Server program has been executed successfully.

EXP NO : 3(B)
DATE : 02.09.2022

TCP CHAT APPLICATION

AIM:

To write a program for TCP chat between client and server.

ALGORITHM:

SERVER:

- STEP 1: Start
- STEP 2: Declare the variables for the socket
- STEP 3: Specify the family, protocol, IP address and port number
- STEP 4: Create a socket using socket() function
- STEP 5: Bind the IP address and Port number
- STEP 6: Listen and accept the client's request for the connection
- STEP 7: Read the client's message
- STEP 8: Display the client's message
- STEP 9: Continue the chat
- STEP 10: Terminate the chat
- STEP 11: Close the socket
- STEP 12: Stop

CLIENT:

- STEP 1: Start
- STEP 2: Declare the variables for the socket
- STEP 3: Specify the family, protocol, IP address and port number
- STEP 4: Create a socket using socket() function
- STEP 5: Call the connect() function
- STEP 6: Read the input message
- STEP 7: Send the input message to the server
- STEP 8: Display the server's reply
- STEP 9: Continue the chat
- STEP 10: Terminate the chat
- STEP 11: Close the socket
- STEP 12: Stop

PROGRAM:

SERVER:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5035
int main(int argc,char**argv)
{
    int sockfd,newsockfd,clength;
    struct sockaddr_inserv_addr,cli_addr;
    char buffer[4096];
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=INADDR_ANY;
```

```

serv_addr.sin_port=htons(SERV_TCP_PORT);
bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
listen(sockfd,5);
clength=sizeof(cli_addr);
newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr,&clength);
read(newsockfd,buffer,4096);
while(buffer!="quit")
{
    printf("\nClient message: %s",buffer);
    printf("\nType your message : ");
    fgets(buffer,4096,stdin);
    write(newsockfd,buffer,4096);
    printf("\n");
    read(newsockfd,buffer,4096);
}
close(sockfd);
return 0;
}

```

CLIENT:

```

#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5035
int main(int argc,char**argv)
{
    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[4096];
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
    serv_addr.sin_port=htons(SERV_TCP_PORT);
    connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    printf("\nEnter the message to send : ");
    fgets(buffer,4096,stdin);
    fputs(buffer,stdout);
    while(buffer!="quit")
    {
        if(buffer=="quit")
            break;
        write(sockfd,buffer,4096);
        read(sockfd,buffer,4096);
        printf("\n");
        printf("\nServer message:\t%s",buffer);
        printf("\nType your message:\t");
        fgets(buffer,4096,stdin);
    }
    close(sockfd);
    return(0);
}

```

OUTPUT:**SERVER:**

```

prince23@prince:~
Red Hat Enterprise Linux Server release 5.4 (Tikanga)
Kernel 2.6.18-164.el5xen on an i686
login: prince23
Password:
Last login: Tue Sep 27 11:57:26 from 192.168.1.172
[prince23@prince ~]$ vi chatserver.c
[prince23@prince ~]$ cc chatserver.c
[prince23@prince ~]$ ./a.out

Client message: hello
Type your message :
Client message: how are you?
Type your message : I AM FINE
Client message: OK
Type your message : █

```

CLIENT:

```

prince23@prince:~
Red Hat Enterprise Linux Server release 5.4 (Tikanga)
Kernel 2.6.18-164.el5xen on an i686
login: prince23
Password:
Last login: Tue Sep 27 12:13:42 from 192.168.1.175
[prince23@prince ~]$ vi chatclient.c
[prince23@prince ~]$ cc chatclient.c
[prince23@prince ~]$ ./a.out

Enter the message to send : hello
hello

Server message:
Type your message:      how are you?

Server message: I AM FINE
Type your message:      OK
█

```

RESULT:

Thus the TCP Chat program has been executed successfully.

EXP NO : 3(C)
DATE : 02.09.2022

FTP Using TCP

AIM:

To write a program for File transfer using TCP.

ALGORITHM:

CLIENT:

1. Start the program
2. Declare the variables and structures required.
3. A socket is created and the connect function is executed.
4. The file is opened.
5. The data from the file is read and sent to the server.
6. The socket is closed.
7. The program is stopped.

SERVER:

1. Start the program.
2. Declare the variables and structures required.
3. The socket is created using the socket function.
4. The socket is binded to the specific port.
5. Start listening for the connections.
6. Accept the connection from the client.
7. Create a new file.
8. Receives the data from the client.
9. Write the data into the file.
10. The program is stopped.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#define SIZE 1024

void sendfile(_FILE *fp, int sockfd){
    int n;
    char data[SIZE] = {0};

    while(fgets(data, SIZE, fp) != NULL) {
        if (send(sockfd, data, sizeof(data), 0) == -1) {
            perror("[-]Error in sending file.");
            exit(1);
        }
        bzero(data, SIZE);
    }
}

int main()
{
```

```

char *ip = "127.0.0.1";
int port = 8080;
int e;
int sockfd;
struct sockaddr_in server_addr;
FILE *fp;
char *filename = "send.txt";

sockfd = socket(AF_INET, SOCK_STREAM, 0);
if(sockfd < 0) {
    perror("[-]Error in socket");
    exit(1);
}
printf("[+]Server socket created successfully.\n");

server_addr.sin_family = AF_INET;
server_addr.sin_port = port;
server_addr.sin_addr.s_addr = inet_addr(ip);

e = connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
if(e == -1) {
    perror("[-]Error in socket");
    exit(1);
}
printf("[+]Connected to Server.\n");

fp = fopen(filename, "r");
if (fp == NULL) {
    perror("[-]Error in reading file.");
    exit(1);
}

send_file(fp, sockfd);
printf("[+]File data sent successfully.\n");

printf("[+]Closing the connection.\n");
close(sockfd);

return 0;
}

```

Server:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#define SIZE 1024

void write_file(int sockfd){
    int n;
    FILE *fp;
    char *filename = "recv.txt";
    char buffer[SIZE];

    fp = fopen(filename, "w");
    while (1) {

```



```

    n = recv(sockfd, buffer, SIZE, 0);
    if (n <= 0){
        break;
        return;
    }
    fprintf(fp, "%s", buffer);
    bzero(buffer, SIZE);
}
return;
}

int main(){
    char *ip = "127.0.0.1";
    int port = 8080;
    int e;

    int sockfd, new_sock;
    struct sockaddr_in server_addr, new_addr;
    socklen_t addr_size;
    char buffer[SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0) {
        perror("[-]Error in socket");
        exit(1);
    }
    printf("[+]Server socket created successfully.\n");

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = port;
    server_addr.sin_addr.s_addr = inet_addr(ip);

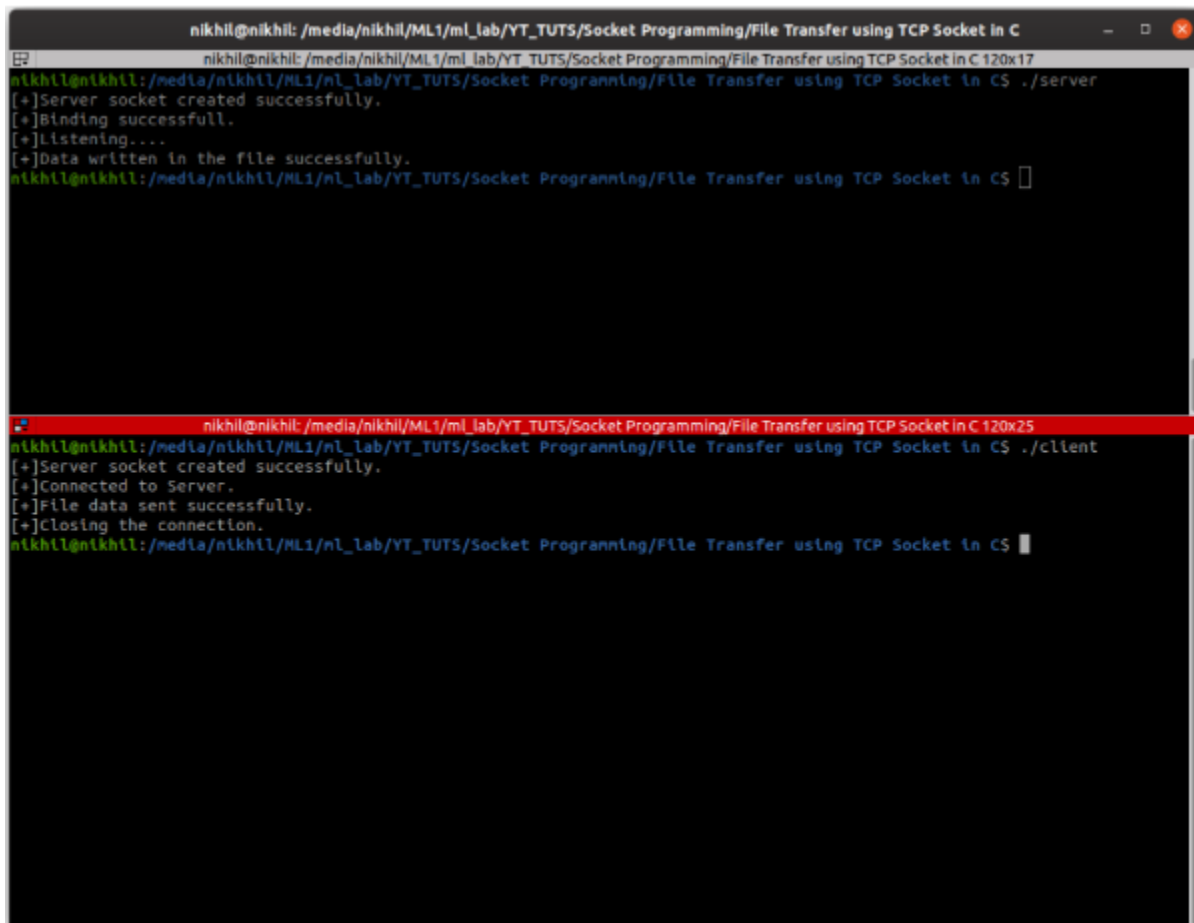
    e = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
    if(e < 0) {
        perror("[-]Error in bind");
        exit(1);
    }
    printf("[+]Binding successfull.\n");

    if(listen(sockfd, 10) == 0){
        printf("[+]Listening....\n");
    }else{
        perror("[-]Error in listening");
        exit(1);
    }

    addr_size = sizeof(new_addr);
    new_sock = accept(sockfd, (struct sockaddr*)&new_addr, &addr_size);
    write_file(new_sock);
    printf("[+]Data written in the file successfully.\n");

    return 0;
}

```

OUTPUT:

```
nikhil@nikhil: /media/nikhil/ML1/ml_lab/YT_TUTS/Socket Programming/File Transfer using TCP Socket in C
nikhil@nikhil: /media/nikhil/ML1/ml_lab/YT_TUTS/Socket Programming/File Transfer using TCP Socket in C 120x17
nikhil@nikhil: /media/nikhil/ML1/ml_lab/YT_TUTS/Socket Programming/File Transfer using TCP Socket in CS ./server
[+]Server socket created successfully.
[+]Binding successfull.
[+]Listening....
[+]Data written in the file successfully.
nikhil@nikhil: /media/nikhil/ML1/ml_lab/YT_TUTS/Socket Programming/File Transfer using TCP Socket in CS █

nikhil@nikhil: /media/nikhil/ML1/ml_lab/YT_TUTS/Socket Programming/File Transfer using TCP Socket in C 120x25
nikhil@nikhil: /media/nikhil/ML1/ml_lab/YT_TUTS/Socket Programming/File Transfer using TCP Socket in CS ./client
[+]Server socket created successfully.
[+]Connected to Server.
[+]File data sent successfully.
[+]Closing the connection.
nikhil@nikhil: /media/nikhil/ML1/ml_lab/YT_TUTS/Socket Programming/File Transfer using TCP Socket in CS █
```

Result:

Thus the FTP using TCP has been executed successfully.

EXP NO : 4**DATE : 09.09.2022****DOMAIN NAME SYSTEM****AIM:**

To write a c-program to implement Domain name system using TCP and UDP

ALGORITHM:**SERVER:**

1. Create a socket for the DNS server
2. Bind the socket
3. Listen for DNS client connection
4. Accept the connection
5. Maintain the list of domain names and their IP addresses
6. Receive the domain name from the client
7. Check whether the domain name is available in the server if it is present send the corresponding IP address to the DNS client
8. Stop the execution

CLIENT:

1. Create a socket for the DNS client
2. Bind the socket
3. Connect with the DNS server
4. Send the request in the form of domain name to the DNS server for getting its corresponding IP address
5. Receive the IP corresponding IP address from the server and print it in its console
6. Stop the execution

PROGRAM:**SERVER:**

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<error.h>
#define max 100
typedef struct sockaddr SA;
struct dns
{
    char dname[25];
    char ipadr[25];
};
int main()
{
    int sfd,cfd,len,i,n,sport;
    struct dns r;
    struct dnsd[]={
        {"www.annauniv.edu","10.0.0.1"},
```

```

        {"www.yahoo.com", "20.0.0.1"},
        {"www.rediffmail.com", "30.0.0.1"}
    };
    n=sizeof(d)/50;
    struct sockaddr_in server, client;
    char buff[max];
    printf("Enter the port no");
    scanf("%d", &sport);
    sfd=socket(AF_INET, SOCK_STREAM, 0);
    bzero(&server, sizeof(server));
    server.sin_family=AF_INET;
    server.sin_port=htons(sport);
    server.sin_addr.s_addr=htonl(0);
    bind(sfd, (SA*)&server, sizeof(server));
    len=sizeof(client);
    listen(sfd, 2);
    cfd=accept(sfd, (SA*)&client, &len);
    bzero(&r, sizeof(struct dns));
    read(cfd, &r, sizeof(struct dns));
    printf("\nDomain name = %s", r.dname);
    for(i=0; i<n; i++)
    {
        if(!strcmp(r.dname, d[i].dname))
        {
            strcpy(r.ipadr, d[i].ipadr);
            write(cfd, &r, sizeof(struct dns));
            break;
        }
    }
    if(i==n)
    {
        strcpy(r.ipadr, "Error: Domain not found");
        write(cfd, &r, sizeof(struct dns));
    }
    close(cfd);
    close(sfd);
}

```

CLIENT:

```

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<error.h>
#define max 100

typedef struct sockaddr SA;
struct dns
{
    char dname[25];
    char ipadr[25];
};

int main()
{
    int cfd, cport;

```

```

char buff[max];
struct sockaddr_in server;
struct dns r;
printf("Enter the port no");
scanf("%d",&cport);
cfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&server,sizeof(server));
server.sin_family=AF_INET;
server.sin_port=htons(cport);
server.sin_addr.s_addr=htonl(0);
connect(cfd,(SA*)&server,sizeof(server));
printf("\nEnter domain name = ");
scanf("%s",r.dname);
write(cfd,&r,sizeof(struct dns));
bzero(&r,sizeof(struct dns));
read(cfd,&r,sizeof(struct dns));
printf("ipaddr = %s",r.ipadr);
close(cfd);
}

```

OUTPUT:**SERVER:**

```

[prince23@prince ~]$ cc dnsserver.c
[prince23@prince ~]$ ./a.out
Enter the port no 3981

[prince23@prince ~]$ █

```

CLIENT:

```

[prince23@prince ~]$ cc dnsclient.c
[prince23@prince ~]$ ./a.out
Enter the port no 3981

Enter domain name = www.annauniv.edu
ipaddr = 10.0.0.1[prince23@prince ~]$ █

```

RESULT:

Thus the DNS program has been executed successfully.

EXP NO : 5**DATE : 16.09.2022****ARP/RARP****AIM:**

To write a c-program to implement ARP/RARP using TCP and UDP

ALGORITHM:

- 1: Initialize the string of ip addresses
- 2: For every ip address, assign an ethernet address
- 3: To Perform ARP, enter the ip address
- 4: The equivalent ethernet address is displayed
- 5: If the ethernet address is not found, then 'not found' message is displayed
- 6: To Perform RARP, enter the ethernet address
- 7: The equivalent ip address is displayed
- 8: If the ip address is not found, then 'not found' message is displayed
- 9: Provide option to perform both ARP and RARP
- 10: Choose Exit option to terminate the program

PROGRAM:

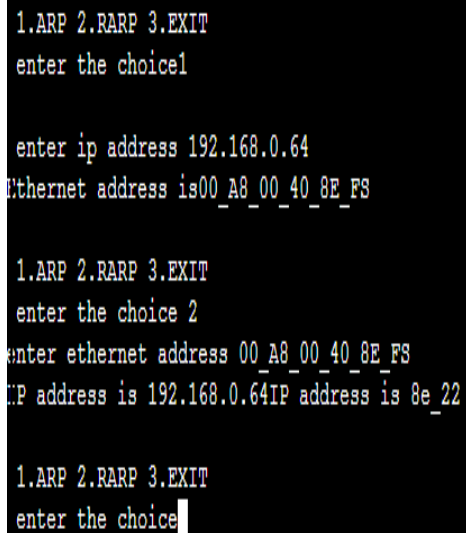
```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
char ip[10][20]={"192.168.0.64","192.168.0.60","192.168.0.68","132.147.3.3"};
char
et[10][20]={"00_A8_00_40_8E_FS","00_16_17_31_8e_22","00_16_17_31_8E_F7","00_16_17_31_8E_08
"};
char ipaddr[20],etaddr[20];
int i,op;
int x=0,y=0;
while(1)
{
printf("\n\n 1.ARP 2.RARP 3.EXIT");
printf("\n enter the choice");
scanf("%d",&op);
switch(op)
{
case 1:
printf("\n enter ip address");
scanf("%s",ipaddr);
for(i=0;i<=20;i++)
{
if(strcmp(ipaddr,ip[i])==0)
{
printf("Ethernet address is%s",et[i]);
x=1;
} }
if(x==0)
printf("invalid ip address");
x=0;
break;
case 2:
```

```

printf("enter ethernet address");
scanf("%s",etaddr);
for(i=0;i<=20;i++)
{
if(strcmp(etaddr,et[i])==0)
{
printf("IP address is %s",ip[i]);
y=1;
} }
if(y==0)
printf("Invalid ethernet address");
y=0;
break;
case 3:
exit(0);
} } }

```

OUTPUT:



```

1.ARP 2.RARP 3.EXIT
enter the choice1

enter ip address 192.168.0.64
Ethernet address is 00_A8_00_40_8E_FS

1.ARP 2.RARP 3.EXIT
enter the choice 2
enter ethernet address 00_A8_00_40_8E_FS
IP address is 192.168.0.64IP address is 8e_22

1.ARP 2.RARP 3.EXIT
enter the choice

```

RESULT:

Thus the ARP/RARP c program is executed successfully.

EXP NO : 6**DATE : 23.09.2022****STUDY OF NETWORK SIMULATOR (NS)****AIM:**

To study about NS2 simulator in detail.

THEORY:

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool, thanks to substantial contributions from the players in the field. Among these are the University of California and Cornell University who developed the REAL network simulator,¹ the foundation which NS is based on. Since 1995 the Defense Advanced Research Projects Agency (DARPA) supported development of NS through the Virtual Inter Network Testbed (VINT) project. Currently the National Science Foundation (NSF) has joined the ride in development. Last but not the least, the group of Researchers and developers in the community are constantly working to keep NS2 strong and versatile.

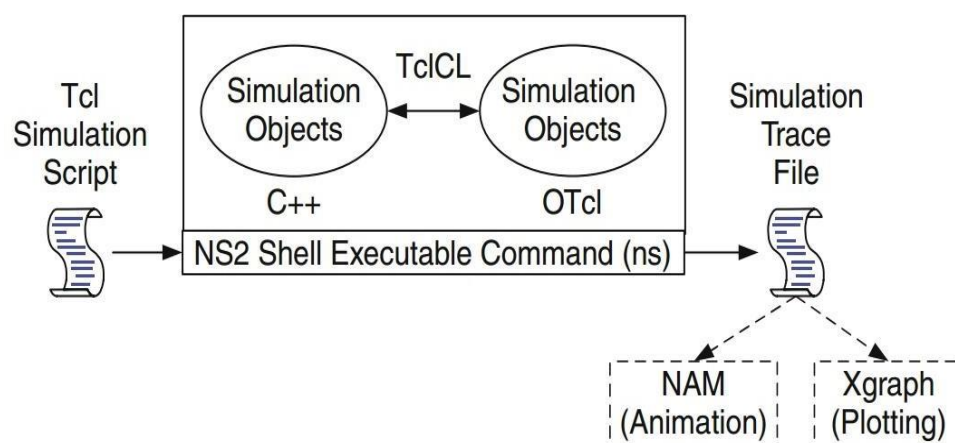
BASIC ARCHITECTURE:

Fig. 2.1. Basic architecture of NS.

Figure 2.1 shows the basic architecture of NS2. NS2 provides users with an executable command ns

which takes on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable commandns.

In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

The C++ and the OTcl are linked together using TclCL. Mapped to a C++ object, variables in the OTcl domains are sometimes referred to as handles. Conceptually, a handle (e.g., n as a Node handle) is just a string (e.g.,_o10) in the OTcl domain, and does not contain any functionality. Instead, the functionality (e.g., receiving a packet) is defined in the mapped C++ object (e.g., of class Connector). In the OTcl domain, a handle acts as a frontend which interacts with users and otherOTclobjects. It may defines its own procedures and variables to facilitate the interaction. Note that the member procedures and variables in the OTcl domain are called instance procedures (instprocs) and instance variables (instvars), respectively. Before proceeding further, the readers are encouraged to learn C++ and OTcl languages. We refer the readers to [14] for the detail of C++, while a brief tutorial of Tcl and OTcl tutorial are given in Appendices A.1 and A.2, respectively.

NS2 provides a large number of built-in C++ objects. It is advisable to use these C++ objects to set up a simulation using a Tcl simulation script. However, advance users may find these objects insufficient. They need to develop their own C++ objects, and use aOTcl configuration interface to put together these objects. After simulation, NS2 outputs either text-based or animation-based simulation results. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used. To analyze a particular behaviour of the network, users can extract a relevant subset of text-based data and transform it to a more conceivablepresentation.

CONCEPT OVERVIEW:

NS uses two languages because simulator has two different kinds of things it needs to do. On one hand,detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios.

In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. ns meets both of these needs with two languages, C++ andOTcl.

Tcl scripting

Tcl is a general purpose scripting language. [Interpreter]

- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to its usage.

Basics of TCL

Syntax: command arg1 arg2 arg3

Hello World!

```
puts stdout{Hello, World!} Hello, World!
```

Variables Command

Substitution set a 5 set len

```
[string length foobar]
```

```
set b $a set len [expr [string length foobar] + 9]
```

Wired TCL Script Components

Create the event scheduler

Open new files & turn on

the tracing Create the

nodes

Setup the links

Configure the traffic type (e.g., TCP,

UDP, etc) Set the time of traffic

generation (e.g., CBR, FTP) Terminate

the simulation

NS Simulator Preliminaries.

1. Initialization and termination aspects of the nssimulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

```
set ns [new Simulator]
```

Which is thus the first line in the tcl script. This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using —opencommand:

#Open the Trace file

```
set tracefile1 [open out.tr w]
```

```
$ns trace-all $tracefile1
```

#Open the NAM trace file

```
set namfile [open out.nam w]
```

```
$ns namtrace-all $namfile
```

The above creates a dta trace file called out.tr and a nam visualization trace file called out.nam.

Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called —tracefile1 and —namfile respectively. Remark that they begins with a # symbol. The second line open the file —out.tr to be used for writing, declared with the letter —w. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

Define a “finish”

```
procedure Proc
```

```
finish { } {
```

```
global ns tracefile1 namfile
```

```
$ns
```

```
flush-
```

```
trace
```

```
Close
```

```
$trace
```

```
file1
```

```
Close
```

```
$namf
```

```
ile
```

```
Exec
```

```
namout.na
```

```
m& Exit 0
```

```
}
```

Definition of a network of links and nodes

The way to define a node is

```
set n0 [$ns node]
```

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

```
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
```

Which means that \$n0 and \$n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

To define a directional link instead of a bi-directional one, we should replace —duplex-link by —simplex-link.

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

```
#set Queue Size of link (n0-n2) to 20
```

```
$ns queue-limit $n0 $n2 20
```

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

```
set tcp [new Agent/TCP]
```

The command **\$ns attach-agent \$n0 \$tcp** defines the source node of the tcp connection.

The command **set sink [new Agent /TCPSink]** Defines the behavior of the destination node of TCP and assigns to it a pointer called sink.

#Setup a UDP

```
connection set
```

```
udp[new
```

```
Agent/UDP]
```

```
$ns attach-agent
```

```
$n1 $udp set null
```

[new Agent/Null]

\$ns attach-agent \$n5 \$null

\$ns connect \$udp \$null

\$udp set fid_2

#setup a CBR over UDP connection

The below shows the definition of a CBR application using a UDP agent

The command **\$ns attach-agent \$n4 \$sink** defines the destination node. The command **\$ns connect \$tcp \$sink** finally makes the TCP connection between the source and destination nodes.

set cbr [new Application/Traffic/CBR]

\$cbr attach-agent \$udp

\$cbr set packetSize_ 100

\$cbr set rate_ 0.01Mb

\$cbr set random_ false

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes. This can be changed to another value, say 552bytes, using the command **\$tcp set packetSize_ 552**.

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **\$tcp set fid_ 1** that assigns to the TCP connection a flow identification of —1. We shall later give the flow identification of —2 to the UDP connection.

RESULT:

Thus the Network Simulator 2 has been studied in detail

EXP NO : 7**DATE : 30.09.2022****CONGESTION CONTROL IN TCP****AIM:**

To Simulate the study of congestion control algorithm using NS2

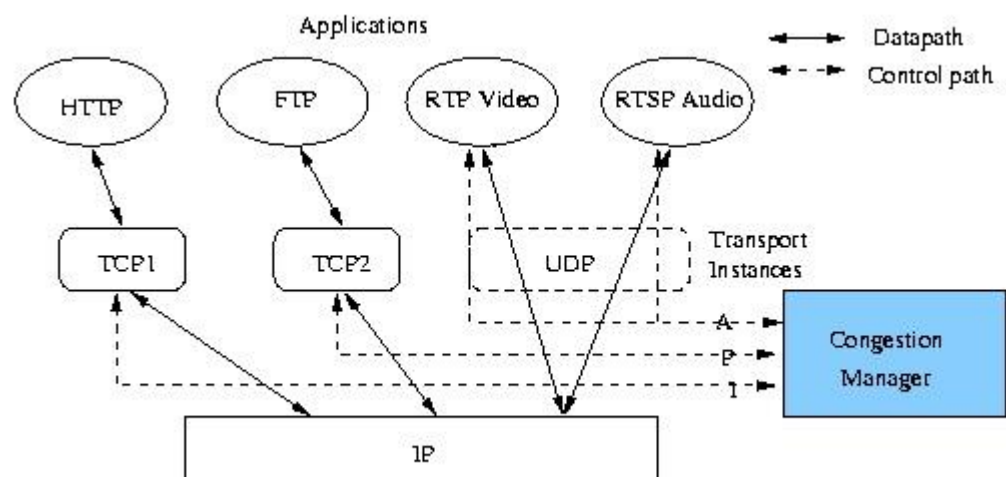
CONGESTION CONTROL:

- Congestion refers to a network state where a node or link carries so much data that it may deteriorate network service quality, resulting in queuing delay, frame or data packet loss and the blocking of new connections.
- In Congestion control, end systems throttle back in order to avoid congesting the network. The mechanism is similar to end-to-end flow controls, but the intention is to reduce congestion in the network, not the receiver.
- Network simulator 2 (Ns2 Program for congestion control outputs better results).

Techniques available in congestion control:

Open Loop Technique and closed Loop Technique are utilized in ns2 program for congestion control.

- Open loop
 - Acknowledgement policy.
 - Retransmission policy.
 - Discarding policy.
 - Window policy.
 - Admission policy.
- Closed loop
 - Explicit feedback.
 - Back pressure.
 - Implicit feedback.



- Choke packet.

TCL SCRIPT FOR CONGESTION CONTROL IN TCP

```

set ns [new Simulator]
set f [ open congestion.tr w ]
$ns trace-all $f
set nf [ open congestion.namw ]
$ns namtrace-all $nf
$ns color 1 Red
$ns color 2 Blue
$ns color 3 White
$ns color 4 Green
#to create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
# to create the link between the nodes with bandwidth, delay and queue
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 200ms DropTail
$ns duplex-link $n3 $n4 0.5Mb 40ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 30ms DropTail
# Sending node with agent as Reno Agent
set tcp1 [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp1
set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp2
set tcp3 [new Agent/TCP/Reno]
$ns attach-agent $n2 $tcp3
set tcp4 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp4
$tcp1 set fid_ 1
$tcp2 set fid_ 2
$tcp3 set fid_ 3
$tcp4 set fid_ 4
# receiving (sink) node

```

```

set sink1 [new Agent/TCPSink]
$ns attach-agent $n4 $sink1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4
# establish the traffic between the source and sink
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2
$ns connect $tcp3 $sink3
$ns connect $tcp4 $sink4
# Setup a FTP traffic generator on "tcp"
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ftp3 set type_ FTP
set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4
$ftp4 set type_ FTP
set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0
set p1 [new Agent/Ping]
$ns attach-agent $n4 $p1
#Connect the two agents
$ns connect $p0 $p1
# Method call from ping.cc file
Agent/Ping instprocrcv {from rtt} {
$self instvar node_
puts "node [$node_ id] received ping answer from \
$from with round-trip-time $rttms."
}
# start/stop the traffic

```

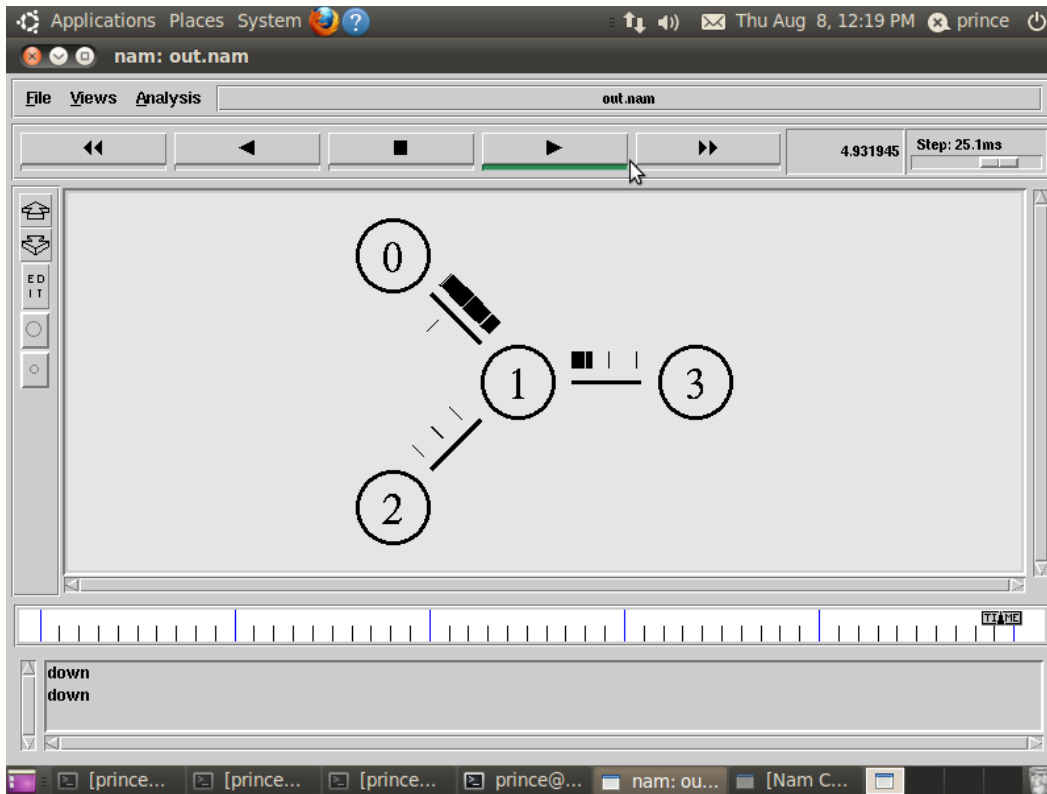


```

$ns at 0.2 "$p0 send"
$ns at 0.3 "$p1 send"
$ns at 0.5 "$ftp1 start"
$ns at 0.6 "$ftp2 start"
$ns at 0.7 "$ftp3 start"
$ns at 0.8 "$ftp4 start"
$ns at 66.0 "$ftp4 stop"
$ns at 67.0 "$ftp3 stop"
$ns at 68.0 "$ftp2 stop"
$ns at 70.0 "$ftp1 stop"
$ns at 70.1 "$p0 send"
$ns at 70.2 "$p1 send"
# Set simulation end time
$ns at 80.0 "finish"

# procedure to plot the congestion window
# cwnd_ used from tcp-reno.cc file
proc plotWindow {tcpSourceoutfile} {
    global ns
    set now [$ns now]
    set cwnd_ [$tcpSource set cwnd_]
    # the data is recorded in a file called congestion.xg.
    puts $outfile "$now $cwnd_"
    $ns at [expr $now+0.1] "plotWindow $tcpSource $outfile"
}
set outfile [open "congestion.xg" w]
$ns at 0.0 "plotWindow $tcp1 $outfile"
proc finish {} {
    exec namcongestion.nam&
    exec xgraphcongestion.xg -geometry 300x300 &
    exit 0
}
# Run simulation
$ns run

```

OUTPUT:**RESULT:**

Thus the Congestion Control Algorithm has been Simulated and studied.

EXP NO : 8
DATE : 30.09.2022

NS2 SIMULATION WITH TCP AND UDP PACKETS

AIM:

To simulate the study of tcp and udp using NS2

TRANSMISSION CONTROL PROTOCOL:

In theory, a transport layer protocol could be a very simple software routine, but the TCP protocol cannot be called simple. Why use a transport layer which is as complex as TCP? The most important reason depends on IP's unreliability. In fact all the layers below TCP are unreliable and deliver the datagram hop-by-hop. The IP layer delivers the datagram hop-by-hop and does not guarantee delivery of a datagram; it is a connectionless system. IP simply handles the routing of datagram's; and if problems occur, IP discards the packet without a second thought, generating an error message back to the sender in the process. The task of ascertaining the status of the datagram sent over a network and handling the resending of information if parts have been discarded falls to TCP.

TCP manages the flow of datagram's from the higher layers, as well as incoming datagram's from the IP layer.

It has to ensure that priorities and security are respected. TCP must be capable of handling the termination of an application above it that was expecting incoming datagram's, as well as failures in the lower layers. TCP also must maintain a state table of all data streams in and out of the TCP layer. The isolation of these services in a separate layer enables applications to be designed without regard to flow control or message reliability.

Without the TCP layer, each application would have to implement the services themselves, which is a waste of resources.

TCP resides in the transport layer, positioned above IP but below the upper layers and their applications, as shown in the Figure below. TCP resides only on devices that actually process datagram's, ensuring that the datagram has gone from the source to target machines. It does not reside on a device that simply routes datagram's, so there is no TCP layer in a gateway. This makes sense, because on a gateway the datagram has no need to go higher in the layered model than the IP layer.

User Datagram Protocol:

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
```

```
set nt [open out.tr w]
```

```
$ns trace-all $nt
```

```
proc finish { } {
```

```
    global ns nf
```

```
    $ns flush-trace
```

```
    close $nf
```

```
    exec namout.nam&
```

```
    exit 0
```

```
}
```

```
set h1 [$ns node]
```

```
set r1 [$ns node]
```

```
set h2 [$ns node]
```

```
set h3 [$ns node]
```

```
set r2 [$ns node]
```

```
set h4 [$ns node]
```

```
$ns duplex-link $h1 $r1 10Mb 20ms DropTail
```

```
$ns duplex-link $h2 $r1 10Mb 20ms DropTail
```

```
$ns duplex-link $r1 $r2 1.5Mb 20ms DropTail
```

```
$ns duplex-link $r2 $h3 10Mb 20ms DropTail
```

```
$ns duplex-link $r2 $h4 5Mb 20ms DropTail
```

```
$ns duplex-link-op $h1 $r1 orient right-down
```

```
$ns duplex-link-op $h2 $r1 orient right-up
```

```
$ns duplex-link-op $r1 $r2 orient right
```

```
$ns duplex-link-op $r2 $h3 orient right-up
```

```
$ns duplex-link-op $r2 $h4 orient right-down
```

```
set udp [new Agent/UDP]
```

```
$ns attach-agent $h2 $udp
```

```
set cbr [new Application/Traffic/CBR]
```

```
#$cbr set interval_ 0.0005
```

```
$cbr attach-agent $udp
```

```
set null [new Agent/Null]
```

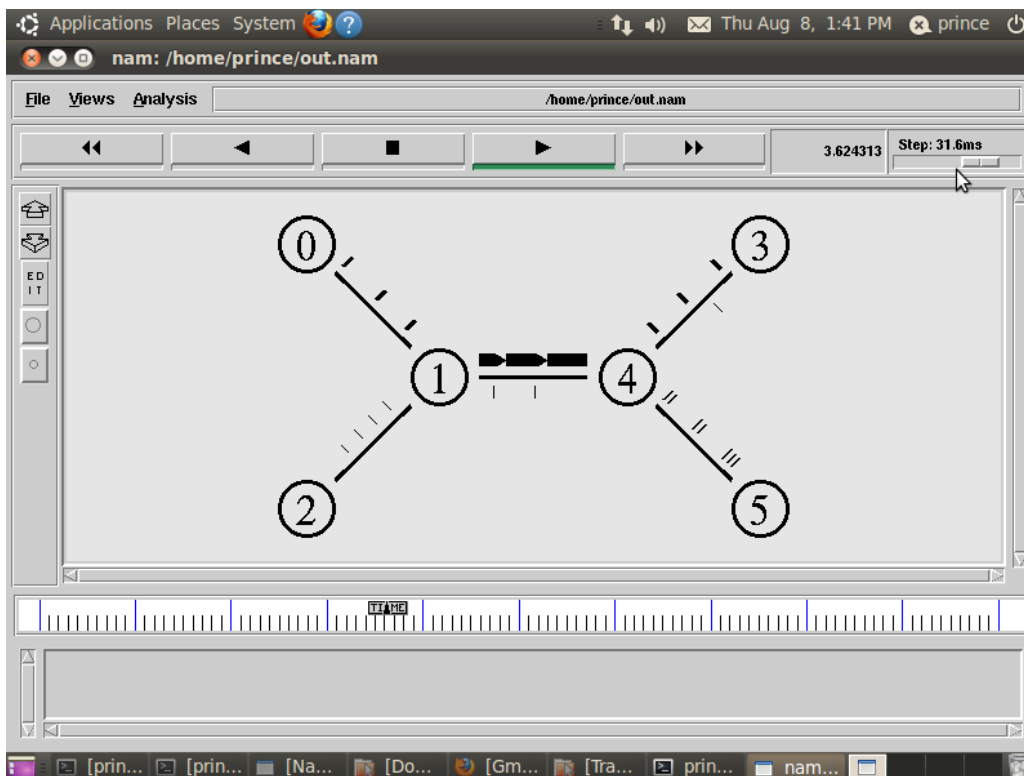
```
$ns attach-agent $h4 $null
```

```
set tcp [new Agent/TCP]
$ns attach-agent $h1 $tcp
```

```
set ftp [new Application/FTP]
#$ftp set interval_ 0.0005
$ftp attach-agent $tcp
```

```
set sink [new Agent/TCPSink]
$ns attach-agent $h3 $sink
$ns connect $udp $null
$ns connect $tcp $sink
$ns at 0.0 "$cbr start"
$ns at 0.0 "$ftp start"
$ns at 10.0 "finish"
$ns run
```

OUTPUT:



RESULT:

Thus the NS2 with TCP and UDP packets has been Simulated and studied.

EXP NO : 9(A)
DATE : 14.10.2022

SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM

AIM:

To simulate and study the Distance Vector routing algorithm using simulation.

SOFTWARE REQUIRED:

NS-2

THEORY:

Distance Vector Routing is one of the routing algorithm in a Wide Area Network for computing shortest path between source and destination. The Router is one main devices used in a wide area network. The main task of the router is Routing. It forms the routing table and delivers the packets depending upon the routes in the table- either directly or via an intermediate devices.

Each router initially has information about its all neighbors. Then this information will be shared among nodes.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different dataflows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose distance vector routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

PROGRAM:

```
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

set tr [open out.tr w]
$ns trace-all $tr

proc finish { } {
    global nf ns tr
    $ns flush-trace
    close $tr
    exec namout.nam&
    exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
```

```

set n3 [$ns node]

$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

set udp [new Agent/UDP]
$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp

set null [new Agent/Null]
$ns attach-agent $n3 $null

$ns connect $tcp $sink
$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3

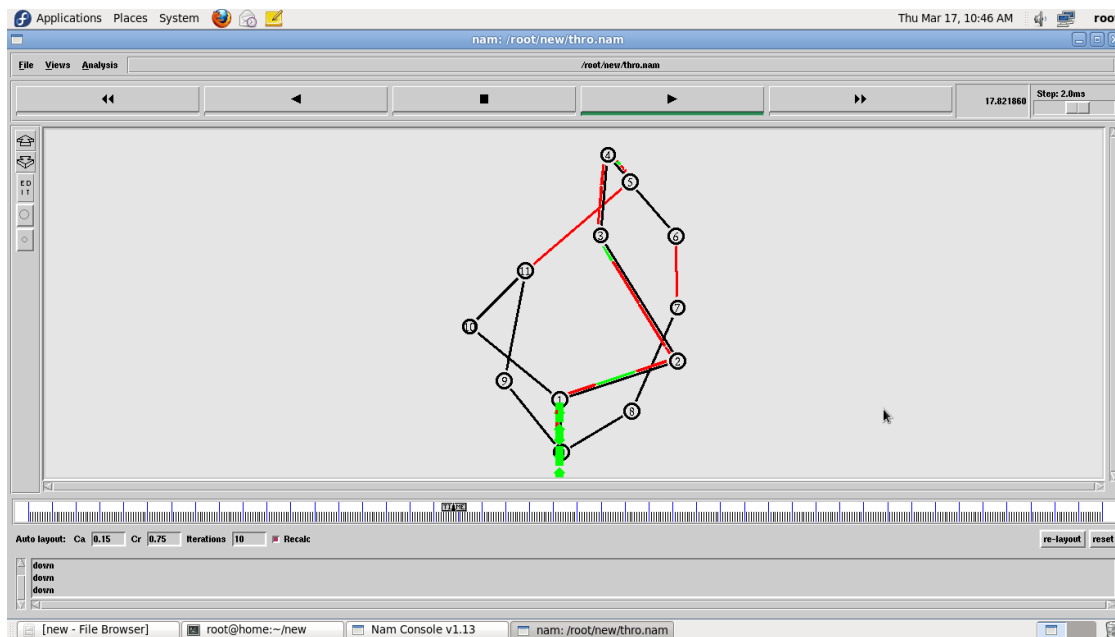
$ns rtproto DV

$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"

$ns at 5.0 "finish"

$ns run

```

OUTPUT:**RESULT:**

Thus the Distance vector Routing Algorithm has been Simulated and studied.

EXP NO : 10

DATE : 14.10.2022

SIMULATION OF LINK STATE ROUTING ALGORITHM

AIM:

To simulate and study the link state routing algorithm using simulation.

SOFTWARE REQUIRED:

NS-2

THEORY:

In **link state routing**, each router shares its knowledge of its neighborhood with every other router in the internet work. (i) **Knowledge about Neighborhood:** Instead of sending its entire routing table a router sends info about its neighborhood only. (ii) **To all Routers:** each router sends this information to every other router on the internet work not just to its neighbor. It does so by a process called **flooding**. (iii) **Information sharing when there is a change:** Each router sends out information about the neighbors when there is change.

PROCEDURE:

The Dijkstra algorithm follows four steps to discover what is called the **shortest path tree** (routing table) for each router: The algorithm begins to build the tree by identifying its roots. The root router's tree is the router itself. The algorithm then attaches all nodes that can be reached from the root. The algorithm compares the tree's temporary arcs and identifies the arc with the lowest cumulative cost. This arc and the node to which it connects are now a permanent part of the shortest path tree. The algorithm examines the database and identifies every node that can be reached from its chosen node. These nodes and their arcs are added temporarily to the tree.

The last two steps are repeated until every node in the network has become a permanent part of the tree.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different dataflows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose Link state routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

PROGRAM:

```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]

$ns namtrace-all $nf
proc finish { } {
    global ns nr nf
    $ns flush-trace
    close $nf
    close $nr
    exec namthro.nam &
```

exit 0

```
for { set i 0 } { $i < 12 } { incre i 1 } { set
n($i) [$ns node]}
```

```
for { set i 0 } { $i < 8 } { incre i 1 } {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail}
```

```
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail
```

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0
```

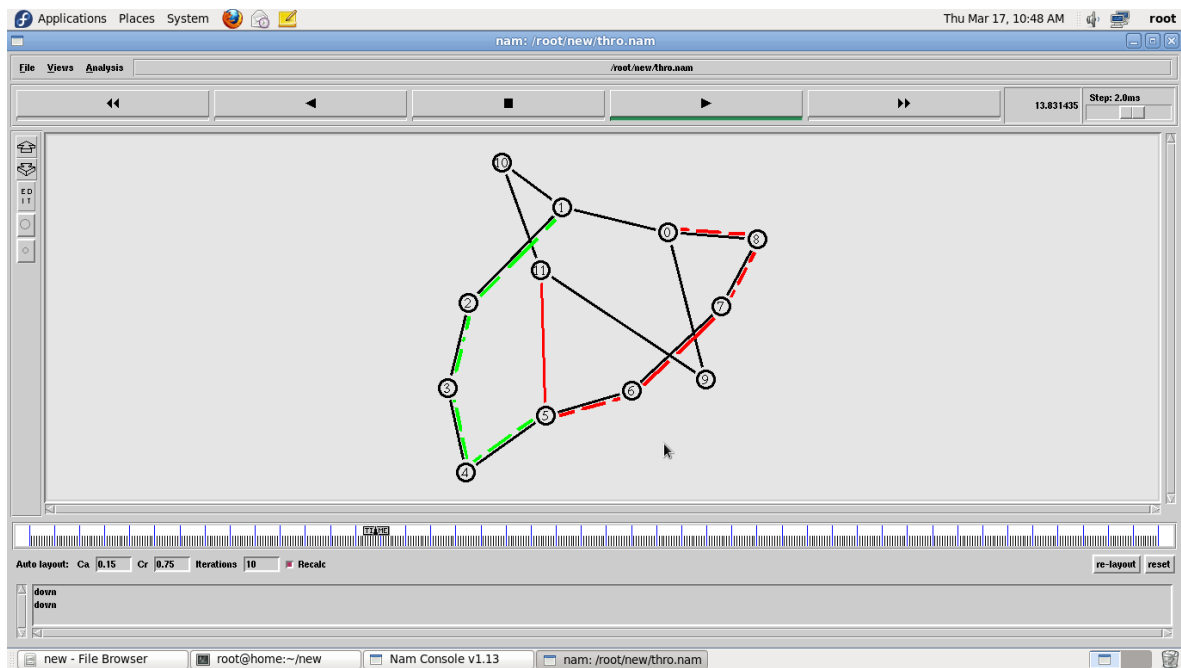
```
set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0
```

```
$ns rtproto LS
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)
```

```
$udp0 set fid_1
$udp1 set fid_2
$ns color 1 Red
$ns color 2 Green
```

```
$ns at 1.0 "$cbr0start"
$ns at 2.0 "$cbr1start"
```

```
$ns at 45 "finish"
$ns run
```

OUTPUT:**RESULT:**

Thus the Link State Routing Algorithm has been Simulated and studied

EXP NO : 11
DATE : 21.10.2022

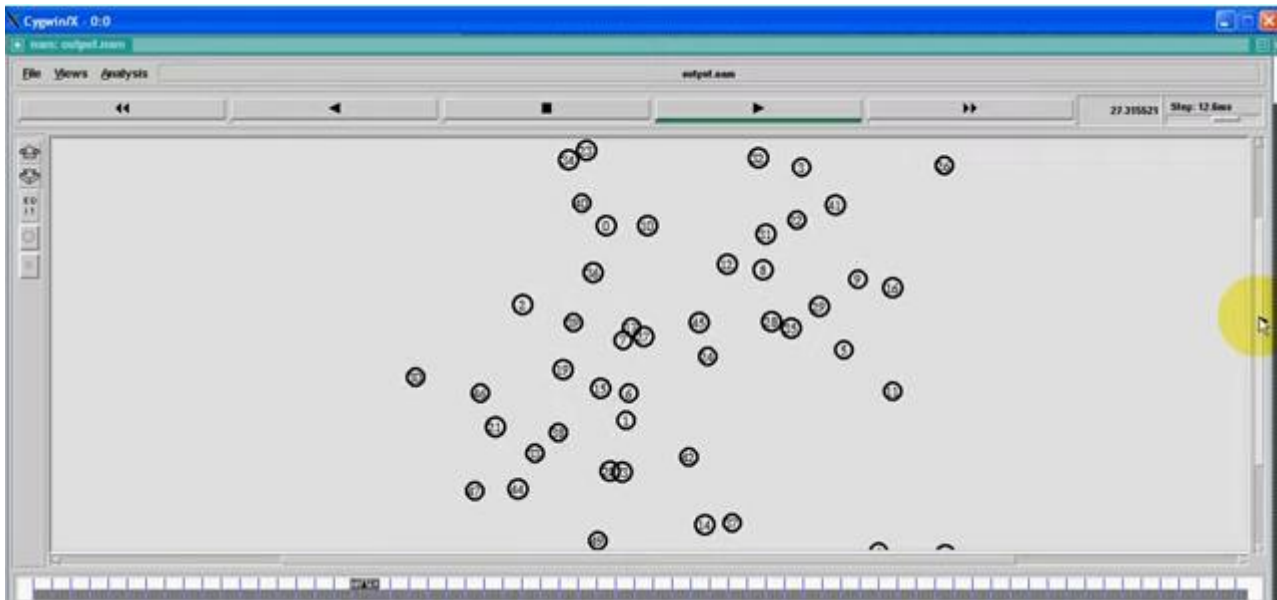
PERFORMANCE EVALUATION OF ROUTING PROTOCOL

AIM:

To compare the various Routing Protocols performance using NS-2

PROGRAM:

```
setns [newSimulator]
$nsmulticast
set f [open out.trw]
$ns trace-all $f
$ns namtrace-all [open out.namw]
$ns color 1 red
$ns color 30 purple
$ns color 31 green
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n1 1.5Mb 10ms DropTail
$ns duplex-link $n0 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n0 $n3 1.5Mb 10ms DropTail
$ns duplex-link-op $n1 orientright
$ns duplex-link-op $n2 orientright-up
$ns duplex-link-op $n3 orientright-down
$ns duplex-link-op $n0 $n1 queuePos
0.5 setmethandle [$ns mrtprotoDm {}]
set cbr0 [new Application/Traffic/CBR]
set udp0 [new Agent/UDP]
$cbr0 attach-agent $udp0
$ns attach-agent $n1 $udp0
$udp0 set dst_0x8001
set cbr1 [new Application/Traffic/CBR]
set udp1 [new Agent/UDP]
$cbr1 attach-agent $udp1
$udp1 set dst_0x8002
$udp1 set class_1
$ns attach-agent $n3 $udp1
set rcvr [new Agent/LossMonitor]
$ns at 1.2 "$n2 join-group $rcvr 0x8002"
$ns at 1.25 "$n2 leave-group $rcvr 0x8002"
$ns at 1.3 "$n2 join-group $rcvr 0x8002"
$ns at 1.35 "$n2 join-group $rcvr 0x8002"
$ns at 1.0 "cbr0 start"
$ns at 1.1 "cbr1 start"
$ns at 2.0 "finish" proc finish { } {
    global ns
    $ns flush-trace
    puts "running nam..." exec nam out.nam
    &exit 0
}
$ns run
```

OUTPUT:**Result:**

Thus the performance evaluation of routing protocol has been executed successfully.

EXP NO : 12**DATE : 28.10.2022****CRC PROGRAM****AIM:**

To write a program for TCP connection establishment.

ALGORITHM:

- 1: Start the process.
- 2: Enter the total number of data and divisor value.
- 3: Read the data to be sent along with divisor value.
- 4: Calculate the CRC value.
- 5: Display the CRC bits.
- 6: Stop the process.

PROGRAM:

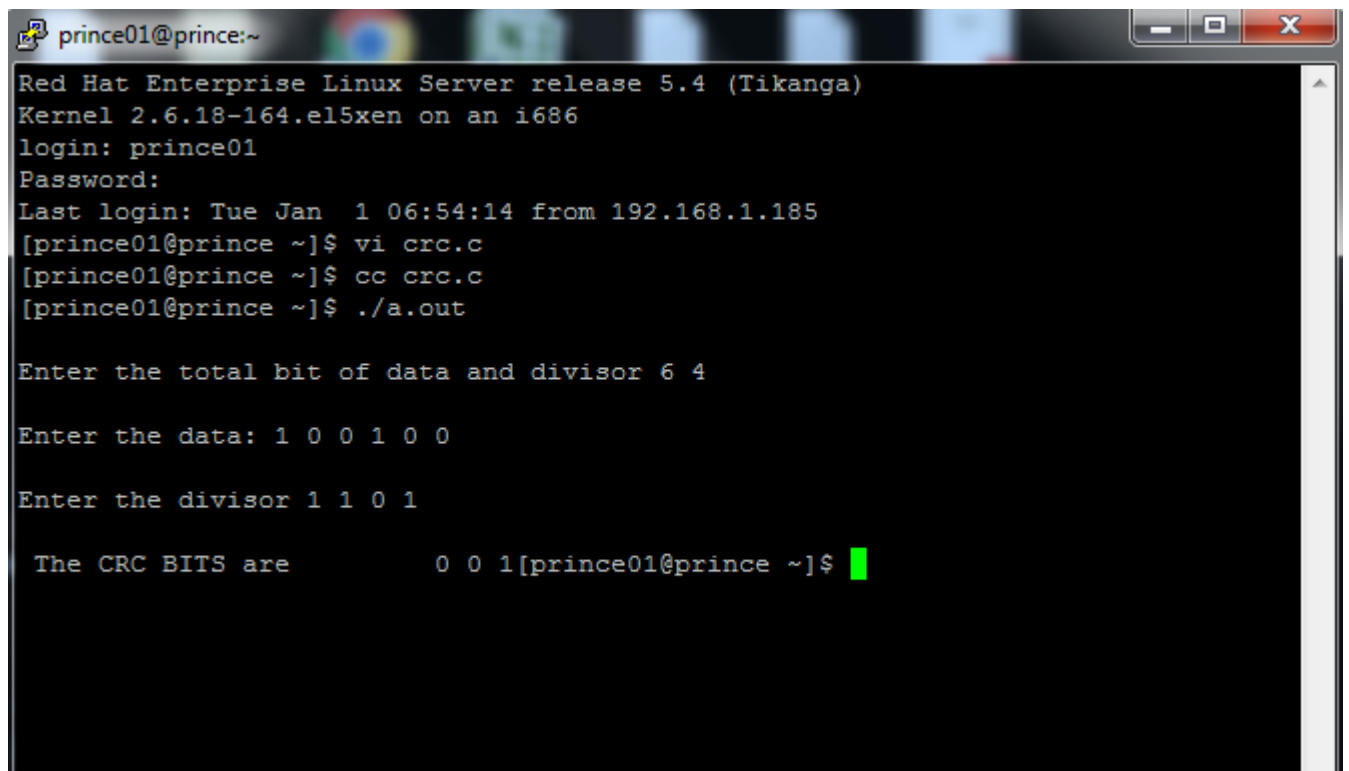
```
#include<stdio.h>
main()
{
    int da[20],di[20],te[20],tem[20],l;
    int i,j,m,n,data,div,t,k,e;
    printf("\nEnter the total bit of data and divisor");
    scanf("%d %d",&data,&div);
    m=data+div-1;
    printf("\nEnter the data:");
    for(i=0;i<data;i++)
        { scanf("%d",&da[i]);te[i]=da[i]; }
    for(i=data;i<m;i++)
        { te[i]=0; }
    printf("\nEnter the divisor");
    for(i=0;i<div;i++)
        { scanf("%d",&di[i]); }
    l=div;t=0;
    k=0;
    for(i=0;i<data;i++)
    {
        e=0;t=0;
        for(j=1;j<div;j++)
        {
            if(((da[j]==1)&&(di[j]==1))||((da[j]==0)&&(di[j]==0)))
            {
                tem[j-1]=0;
                if(e!=1)
                {
                    k=k+1;
                    t=t+1;
                    i=i+1;
                }
            }
            else
            {
```

```

        tem[j-1]=1;
        e=1;
    }
}
j=0;
for(e=t;e<div-1;e++)
{
    da[j]=tem[e];
    j++;
}
for(j=j;j<div;j++)
{
    if(l>=data+1)
    {
        da[j]=0;
    }
    else
    {
        da[j]=te[l];
        l=l+1;
    }
}
}
printf("\n The CRC BITS are\t ");
for(i=0;i<div-1;i++)
{
    printf(" %d",tem[i]);
}
}

```

OUTPUT:



```

prince01@prince:~
Red Hat Enterprise Linux Server release 5.4 (Tikanga)
Kernel 2.6.18-164.el5xen on an i686
login: prince01
Password:
Last login: Tue Jan  1 06:54:14 from 192.168.1.185
[prince01@prince ~]$ vi crc.c
[prince01@prince ~]$ cc crc.c
[prince01@prince ~]$ ./a.out

Enter the total bit of data and divisor 6 4

Enter the data: 1 0 0 1 0 0

Enter the divisor 1 1 0 1

The CRC BITS are      0 0 1[prince01@prince ~]$

```

RESULT:

Thus the program for CRC has been executed successfully.