KONGUENGINEERINGCOLLEGE

(Autonomous)

PERUNDURAI, ERODE 638052

School of Communication and Computer Sciences

Department of Computer Science and Engineering

LABORATORY MANUAL

(As per KEC Autonomous Curricula and Syllabi – R2014)

14ITL61

Network Programming Management

Laboratory

VI SEMESTER BE-CSE

KONGU ENGINEERING COLLEGE
(Autonomous)
PERUNDURAI, ERODE 638052

School of Communication and Computer Sciences

Department of Computer Science and Engineering

LABORATORY MANUAL

(As per KEC Autonomous Curricula and Syllabi – R2014)

14TLL61

Network Programming Management

Laboratory

Prepared by

Dr.K.Sangeetha
AP(SRG)/CSE

Approved by

Dr.R.R.Rajalaxmi
HOD/CSE

14ITL61 NETWORK PROGRAMMING LABORATORY

(COMMON TO CSE & IT Branches)

EXPERIMENT NAME
1. Create socket and display socket ID
2.Implementation of Address Conversion Routines
3. Develop a Client – Server application for chat using TCP
4. Implementation of UDP Client Server Communication using Bind, SendTo, and RecvFrom System Calls
5. Design TCP Client and Server application to transfer file
6. Demonstration to generate SIGPIPE Error with Socket
7. Demonstration to restart server by capturing SIGHUP signal
8. Develop a code for ping and trace route programs
9. Performance analysis of TCP/UDP using simulation tool
10. Simulation of LAN(CSMA/CD) using simulation tool
11. Simulation of Distance vector / Link State routing protocol
12. Simulation and Performance analysis of 802.11 with AODV,DSR, DSDV routing protocols using simulation tool
TOTAL 45

REFERENCES / MANUALS / SOFTWARE:

1. Linux Operating System
2. C Compiler, NS2, awk, Xgraph

Course Outcomes:

On completion of the course the students will be able to

- develop socket API based program and implement client-server application using TCP and UDP sockets
- identify different signal handling functions and use them
- use simulation tools to understand wired and wireless functionalities

LIST OF EXPERIMENTS

EXPERIMENT NAME	COURSE OUTCOME
1. Create socket and display socket ID	CO1
2.Implementation of Address Conversion Routines	CO1
3. Develop a Client – Server application for chat using TCP	CO1
4. Implementation of UDP Client Server Communication using Bind, SendTo, and RecvFrom System Calls	CO1
5. Design TCP Client and Server application to transfer file	CO1
6. Demonstration to generate SIGPIPE Error with Socket	CO2
7. Demonstration to restart server by capturing SIGHUP signal	CO2
8. Develop a code for ping and trace route programs	CO2
9. Performance analysis of TCP/UDP using simulation tool	CO3
10. Simulation of LAN(CSMA/CD) using simulation tool	CO3
11. Simulation of Distance vector / Link State routing protocol	CO3
12. Simulation and Performance analysis of 802.11 with AODV,DSR, DSDV routing protocols using simulation tool	CO3

Ex.no.1 CREATE SOCKET AND DISPLAY SOCKET ID

AIM:

To write a C program to create socket and display the socket id.

ALGORITHM:

1. To create a socket using socket function and pass 3 arguments to the function. The function are family,type and protocol.
2. The argument family values are AF_INET, AF_INET6, AF_LOCAL, AF_ROUTE and AF_KEY.
3. The type values are SOCK_STREAM (TCP), SOCK_DGRAM (UDP), SOCK_SEQPACKET (SCTP) and SOCK_RAW.
4. The protocol values are IPPROTO_TCP, IPPROTO_UDP and IPPROTO_SCTP.
5. The return value of the socket function is stored in the variable id. If id value is greater than 0 then socket is created.
6. Check id value is greater than 0 then print “socket is created”.
7. If id value is less than 0 then print “socket is not created”.
8. Repeat the step for all possible combination.
9. Stop.

PROGRAM:

```
#include<stdio.h>

#include<sys/socket.h>

void main()

{

int id,ch;

printf("enter chioce\n \t \tSOCK_STREAM \n1.AF_INET \n2.AF_INET6 \n3.AF_LOCAL
\n4.AF_ROUTE \n5.AF_KEY \n \t \tSOCK_DGRAM \n6.AF_INET \n7.AF_INET6\n8.AF_LOCAL
\n9.AF_ROUTE \n10.AF_KEY \n \t \t SOCK_SEQPACKET \n11.AF_INET \n12.AF-INET6
\n13.AF_LOCAL \n14.AF_ROUTE \n15.AF_KEY\n \t \t SOCK_RAW \n16.AF_INET
\n17.AF_INET6\n18.AF_LOCAL \n19.AF_ROUTE \n20.AF_KEY");

do
```

```
{
scanf("%d",&ch);
switch(ch)
{
case 1:
id=socket(AF_INET,SOCK_STREAM,0);
break;
case 2:
id=socket(AF_INET6,SOCK_STREAM,0);
break;
case 3:
id=socket(AF_LOCAL,SOCK_STREAM,0);
break;
case 4:
id=socket(AF_ROUTE,SOCK_STREAM,0);
break;
case 5:
id=socket(AF_KEY,SOCK_STREAM,0);
break;
case 6:
id=socket(AF_INET,SOCK_DGRAM,0);
break;
case 7:
id=socket(AF_INET6,SOCK_DGRAM,0);
break;
case 8:
```

```
id=socket(AF_LOCAL,SOCK_DGRAM,0);
break;
case 9:
id=socket(AF_ROUTE,SOCK_DGRAM,0);
break;
case 10:
id=socket(AF_KEY,SOCK_DGRAM,0);
break;
case 11:
id=socket(AF_INET,SOCK_SEQPACKET,0);
break;
case 12:
id=socket(AF_INET6,SOCK_SEQPACKET,0);
break;
case 13:
id=socket(AF_LOCAL,SOCK_SEQPACKET,0);
break;
case 14:
id=socket(AF_ROUTE,SOCK_SEQPACKET,0);
break;
case 15:
id=socket(AF_KEY,SOCK_SEQPACKET,0);
break;
case 16:
id=socket(AF_INET,SOCK_RAW,0);
break;
```

```
case 17:
id=socket(AF_INET6,SOCK_RAW,0);
break;
case 18:
id=socket(AF_LOCAL,SOCK_RAW,0);
break;
case 19:
id=socket(AF_ROUTE,SOCK_RAW,0);
break;
case 20:
id=socket(AF_KEY,SOCK_RAW,0);
break;
}
if(id>0)
{
printf("socket is created and socket id=%d  \n",id);
}
else
printf("socket not created and id=%d  \n",id);
}while(ch!=0);
}
```

OUTPUT:

```
The choices are:
   SOCK_STREAM
1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
```

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:1

SOCKET IS CREATED AND SOCKET ID=3

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE
10.AF_KEY
SOCK_SEQPACKET
11.AF_INET
12.AF_INET6
13.AF_LOCAL
14.AF_ROUTE
15.AF_KEY
SOCK_RAW
16.AF_INET
17.AF_INET6
18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT
ENTER THE CHOICE:2

SOCKET IS CREATED AND SOCKET ID=4
The choices are:
SOCK_STREAM
1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY
SOCK_DGRAM
6.AF_INET
7.AF_INET6
8.AF_LOCAL
9.AF_ROUTE
10.AF_KEY
SOCK_SEQPACKET
11.AF_INET
12.AF_INET6

13.AF_LOCAL
14.AF_ROUTE
15.AF_KEY
SOCK_RAW
16.AF_INET
17.AF_INET6
18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT
ENTER THE CHOICE:3

SOCKET IS CREATED AND SOCKET ID=5
The choices are:
SOCK_STREAM
1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY
SOCK_DGRAM
6.AF_INET
7.AF_INET6
8.AF_LOCAL
9.AF_ROUTE
10.AF_KEY
SOCK_SEQPACKET
11.AF_INET
12.AF_INET6
13.AF_LOCAL
14.AF_ROUTE
15.AF_KEY
SOCK_RAW
16.AF_INET

17.AF_INET6
18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT
ENTER THE CHOICE:4

SOCKET IS NOT CREATED AND SOCKET ID=-1
The choices are:
SOCK_STREAM
1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY
SOCK_DGRAM
6.AF_INET
7.AF_INET6
8.AF_LOCAL
9.AF_ROUTE
10.AF_KEY
SOCK_SEQPACKET
11.AF_INET
12.AF_INET6
13.AF_LOCAL
14.AF_ROUTE
15.AF_KEY
SOCK_RAW
16.AF_INET
17.AF_INET6
18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT

ENTER THE CHOICE:5

SOCKET IS NOT CREATED AND SOCKET ID=-1

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:6

SOCKET IS CREATED AND SOCKET ID=6

The choices are:

SOCK_STREAM

1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY
SOCK_DGRAM
6.AF_INET
7.AF_INET6
8.AF_LOCAL
9.AF_ROUTE
10.AF_KEY
SOCK_SEQPACKET
11.AF_INET
12.AF_INET6
13.AF_LOCAL
14.AF_ROUTE
15.AF_KEY
SOCK_RAW
16.AF_INET
17.AF_INET6
18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT
ENTER THE CHOICE:7

SOCKET IS CREATED AND SOCKET ID=7
The choices are:
SOCK_STREAM
1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:8

SOCKET IS CREATED AND SOCKET ID=8

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:9

SOCKET IS CREATED AND SOCKET ID=9

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:10

SOCKET IS NOT CREATED AND SOCKET ID=-1

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT
ENTER THE CHOICE:11

SOCKET IS NOT CREATED AND SOCKET ID=-1
The choices are:
SOCK_STREAM
1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY
SOCK_DGRAM
6.AF_INET
7.AF_INET6
8.AF_LOCAL
9.AF_ROUTE
10.AF_KEY
SOCK_SEQPACKET
11.AF_INET
12.AF_INET6
13.AF_LOCAL
14.AF_ROUTE
15.AF_KEY
SOCK_RAW
16.AF_INET
17.AF_INET6
18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT
ENTER THE CHOICE:12

SOCKET IS NOT CREATED AND SOCKET ID=-1

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:13

SOCKET IS CREATED AND SOCKET ID=10

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY
SOCK_DGRAM
6.AF_INET
7.AF_INET6
8.AF_LOCAL
9.AF_ROUTE
10.AF_KEY
SOCK_SEQPACKET
11.AF_INET
12.AF_INET6
13.AF_LOCAL
14.AF_ROUTE
15.AF_KEY
SOCK_RAW
16.AF_INET
17.AF_INET6
18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT
ENTER THE CHOICE:14

SOCKET IS NOT CREATED AND SOCKET ID=-1
The choices are:
SOCK_STREAM
1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY
SOCK_DGRAM

6.AF_INET
7.AF_INET6
8.AF_LOCAL
9.AF_ROUTE
10.AF_KEY
SOCK_SEQPACKET
11.AF_INET
12.AF_INET6
13.AF_LOCAL
14.AF_ROUTE
15.AF_KEY
SOCK_RAW
16.AF_INET
17.AF_INET6
18.AF_LOCAL
19.AF_ROUTE
20.AF_KEY
0.EXIT
ENTER THE CHOICE:15

SOCKET IS NOT CREATED AND SOCKET ID=-1
The choices are:
SOCK_STREAM
1.AF_INET
2.AF_INET6
3.AF_LOCAL
4.AF_ROUTE
5.AF_KEY
SOCK_DGRAM
6.AF_INET
7.AF_INET6
8.AF_LOCAL
9.AF_ROUTE
10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:16

SOCKET IS NOT CREATED AND SOCKET ID=-1

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:17

SOCKET IS NOT CREATED AND SOCKET ID=-1

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:18

SOCKET IS CREATED AND SOCKET ID=11

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:19

SOCKET IS CREATED AND SOCKET ID=12

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:20

SOCKET IS NOT CREATED AND SOCKET ID=-1

The choices are:

SOCK_STREAM

1.AF_INET

2.AF_INET6

3.AF_LOCAL

4.AF_ROUTE

5.AF_KEY

SOCK_DGRAM

6.AF_INET

7.AF_INET6

8.AF_LOCAL

9.AF_ROUTE

10.AF_KEY

SOCK_SEQPACKET

11.AF_INET

12.AF_INET6

13.AF_LOCAL

14.AF_ROUTE

15.AF_KEY

SOCK_RAW

16.AF_INET

17.AF_INET6

18.AF_LOCAL

19.AF_ROUTE

20.AF_KEY

0.EXIT

ENTER THE CHOICE:0

SOCKET IS NOT CREATED AND SOCKET ID=-1

RESULT:

Thus the creation of socket id was executed successfully**.**

Ex.no.2 IMPLEMENTATION OF ADDRESS CONVERSION ROUTINES

AIM:

To write C program to implement address conversion routines.

ALGORITHM:

1. Start
2. Get the host address in dotted decimal format.
3. The host address is converted to network address by using inet_address function. Store the result in address variable.
4. Display the network address in address variable.
5. Network address is get from the user using netaddr bin.s_addr which is a datatype declared under in_addr structure.
6. The network address is converted into host address store the result in pointer variable ptr by using inet_nota function.
7. Display the host address in pointer variable ptr.
8. Stop

PROGRAM:

```
#include<stdio.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<arpa/inet.h>
void main()
{
struct in_addr netaddrbin;
long int addr;
char host[20],*ptr;
printf("ENTER THE HOSTID:");
gets(host);
addr=inet_addr(host);
printf("BINARY ADDRESS IS:%d\n",addr);
printf("ENTER THE NETWORK ADDRESS:\n");
scanf("%d",&netaddrbin.s_addr);
ptr=inet_ntoa(netaddrbin);
printf("%s\n",ptr);
}
```

OUTPUT:

```
Enter network byte order value:
255
dotted decimal value
255.0.0.0

Enter the ip address

12.23.4.5

32-bit address=84154124
```

RESULT:

Thus the implementation of address convertion routines was executed successfully.

Ex.no.3 CHAT APPLICATION USING TCP CLIENT AND SERVER

AIM:

To write a C program to create client server application for sorting the numbers in the array.

ALGORITHM:

CLIENT:

1. Start.
2. Create a socket using socket function and the return value is stored sockfd.
3. Create the connect function and check if the return value is equal to -1.
4. If it is true, display that it is failed, else connection is made by calling connect function.
5. If the return value is equal to -1 display that connection with server is failed. If not, server and client connection is successful.
6. Then call the function func() with the sockfd. In func() get the elements for sorting.
7. Stop.

SERVER:

1. Start
2. Create a socket using socket function and check if the returned value is equal to -1.
3. If it is true, display that the socket creation failed. If it is executed to failure, then call the bind function.
4. If bind function returned value is not equal to -1, call listen function.
5. If the listen function returned value is not equal to -1, then call accept function.
6. If it not equal to -1, then call user defined func(). Get message from user and write it using write function.
7. Then read the data sent from server and display it.
8. Stop.

PROGRAM:

SERVER:

```
#include<sys/socket.h>
#include<stdio.h>
#include<netinet/in.h>
#include<arpa/inet.h>
main()
{
int sockfd,len,confd;
struct sockaddr_in servaddr,cliaddr;
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd==-1)
{
printf("socket creation is failed\n");
}
else
{
printf("socket creation is successful\n");
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(43454);
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
if(bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr))==-1)
{
printf("bind failed\n");
```

```
}
else
{
printf("bind is successful\n");
if(listen(sockfd,5)==-1)
{
printf("listen is failed\n");
}
else
{
printf("listen is successful\n");
len=sizeof(cliaddr);
confd=accept(sockfd,(struct sockaddr*)&cliaddr,&len);
if(confd==-1)
{
printf("accept is failed\n");
}
else
{
printf("server accepts client\n");
func(confd);
}
}
}
}
}
```

```
void func(int confd)
{
char buff[50];
printf("Message from client is \n");
read(confd,buff,sizeof(buff));
printf("%s",buff);
printf("\n enter a message to client \n");
gets(buff);
write(confd,buff,sizeof(buff));
}
```

CLIENT:

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdio.h>
#include<arpa/inet.h>
main()
{
int sockfd;
struct sockaddr_in servaddr;
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd==-1)
{
printf("socket creation is failed\n");
}
else
```

```
{
printf("socket creation is successful\n");
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(43454);
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
if(connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr))==-1)
{
printf("connection with server failed\n");
}
else
{
printf("connected with server\n");
func(sockfd);
}
}}
void func(int sockfd)
{
char buff[50];
printf("enter message to server \n");
gets(buff);
write(sockfd,buff,sizeof(buff));
printf("Message from server is \n");
read(sockfd,buff,sizeof(buff));
printf("%s",buff);
}
```

OUTPUT:

SERVER SIDE:
socket creation is successful
bind is successful
listen is successful
server accepts client
Message from client is
hai
enter a message to client
hello

CLIENT SIDE:
socket creation is successful
connected with server
enter message to server
hai
Message from server is
hello

RESULT:

Thus the client server application for sorting the numbers in the array was executed successfully.

Ex.no.4 DEVELOP A UDP CLIENT AND SERVER APPLICATION

AIM:

To implement a UDP client and server application for string reverse.

ALGORITHM:

CLIENT:

1. Start
2. Create UDP client socket using socket function and return value of that function is used to check socket created or not.
3. Assign values to socket address structure that the address of the server and using memset function and set the structure member sin zero value as null.
4. Enter value from the message and pass it to the server as a message using sendti function.
5. Get the request from the server using recvfrom function and display to the client.
6. Stop.

SERVER:

1. Start
2. Create UDP socket using socket function and return value of that function is used to check socket created or not.
3. Assign values to bind with that address using bind function and return value of then function is used to check that the bind is successful or not.
4. Using memset set the structure member sin zero value as null and pass the required argument to the function.
5. Store the size of server storage and pass as an last argument to recvfrom function that is used to receive the message from client and print the message.
6. Reverse the message and send that message to client using sendto function.
7. If client want to exit the process then it enter 0 it will display on serverside.
8. **Stop.**

PROGRAM:

SERVER:

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
int main()
{
int clientSocket, portNum, nBytes;
int ch=1;
char buffer[100];
struct sockaddr_in serverAddr;
socklen_t addr_size;
/*Create UDP socket*/
clientSocket = socket(PF_INET, SOCK_DGRAM, 0);
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(7891);
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
addr_size = sizeof serverAddr;
do
{
printf("Type a sentence to send to server:\n");
fgets(buffer,100,stdin);
printf("You typed: %s",buffer);
nBytes = strlen(buffer);
sendto(clientSocket,buffer,nBytes,0,(struct sockaddr *)&serverAddr,addr_size);
nBytes = recvfrom(clientSocket,buffer,100,0,NULL, NULL);
printf("Received from server: %s\n",buffer);
printf("\n do u want to continue?");
scanf("%d",&ch);
}while(ch!=0);
return 0;
}
```

CLIENT:

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdlib.h>
int main(){
int udpSocket, nBytes;
```

```
int ch=1;
char buffer[100];
struct sockaddr_in serverAddr, clientAddr;
struct sockaddr_storage serverStorage;
socklen_t addr_size, client_addr_size;
int i;
/*Create UDP socket*/
udpSocket = socket(PF_INET, SOCK_DGRAM, 0);
/*Configure*/
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(7891);
serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
/*Bind socket*/
bind(udpSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
addr_size = sizeof serverStorage;
nBytes = recvfrom(udpSocket,buffer,100,0,(struct sockaddr *)&serverStorage, &addr_size);
printf("\n message from client is : %s\n",buffer);
for(i=0;i<nBytes;i++)
buffer[i] = toupper(buffer[i]);
sendto(udpSocket,buffer,nBytes,0,(struct sockaddr *)&serverStorage,addr_size);
return 0;
}
```

OUTPUT:

Server

Bind success

Msg from client is hello

Client

Message to server :

hello

Message from server:

olleh

exit the process enter 0

RESULT:

Thus the implementation UDP client and server application for string reverse was executed successfully.

Ex.no.5 DESIGN TCP CLIENT AND SERVER APPLICATION TO TRANSFER FILE

AIM:

To write a C program to implement file transfer application using TCP client and server.

ALGORITHM:

CLIENT:

1. Start.
2. Create a socket using socket function and the return value is stored sockfd.
3. Create the connect function and check if the return value is equal to -1.
4. If it is true, display that it is failed, else connection is made by calling connect function.
5. If the return value is equal to -1 display that connection with server is failed. If not, server and client connection is successful.
6. Then call the function func() with the sockfd for transfer the file.
7. Now from this func() sends the filename to server and also reads the message from server and checks the filename is created or not.
8. If filename exists, it displays message given from server else it displays file not opened.
9. Stop.

SERVER:

1. Start.
2. Create socket function using TCP protocol of SOCK_STREAM type.
3. Using bind () function, check specific address and port number matches.
4. Using listen () function, specify the maximum client requests the server will listen.
5. To check whether the server accept client request use accept () function.
6. Read the clients filename and print the filename.
7. Open the file by using fd () function. If file exists it returns to client amd display.
8. Stop.

PROGRAM:

SERVER:

```
#include<sys/socket.h>
#include<stdio.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<string.h>
main()
{
int sockfd,len,confd;
struct sockaddr_in servaddr,cliaddr;
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd==-1)
{
printf("socket creation is failed\n");
}
else
{
printf("socket creation is successful\n");
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(43454);
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
if(bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr))==-1)
{
printf("bind failed\n");
}
else
{
printf("bind is successful\n");
if(listen(sockfd,5)==-1)
{
printf("listen is failed\n");
}
else
{
printf("listen is successful\n");
len=sizeof(cliaddr);
confd=accept(sockfd,(struct sockaddr*)&cliaddr,&len);
if(confd==-1)
{
printf("accept is failed\n");
}
else
{
```

```
printf("server accepts client\n");
func(confd);
}
}
}
}
}
void func(int confd)
{
char buff[50];
int fd;
read(confd,buff,sizeof(buff));
printf("Filename by client is %s",buff);
fd=open(buff,O_RDONLY);
if(fd<=0)
{
printf("file is not opened\n");
strcpy(buff,"file not opened");
}
else
{
printf("file is opened\n");
read(fd,buff,sizeof(buff));
}
write(confd,buff,sizeof(buff));
}
```

CLIENT:

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

#include<netinet/in.h>

#include<stdio.h>

#include<arpa/inet.h>

main()

{

int sockfd;

struct sockaddr_in servaddr;

sockfd=socket(AF_INET,SOCK_STREAM,0);

if(sockfd==-1)
```

```
{
printf("socket creation is failed\n");
}
else
{
printf("socket creation is successful\n");
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(43454);
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
if(connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr))==-1)
{
printf("connection with server failed\n");
}
else
{
printf("connected with server\n");
func(sockfd);
}
}
}
void func(int sockfd)
{
char buff[50];
printf("enter filename to server \n");
gets(buff);
write(sockfd,buff,sizeof(buff));
```

```
printf("Message from server is \n");

read(sockfd,buff,sizeof(buff));

printf("%s",buff);

}
```

file.txt

Network Programming Laboratory

3rd year CSE-A

OUTPUT:

File exists

SERVER SIDE:
socket creation is successful
bind is successful
listen is successful
server accepts client
Filename by client is file.txtfile is opened

CLIENT SIDE:
socket creation is successful
connected with server
enter filename to server
file.txtsocket creation is successful
Message from server is
Network Programming Laboratory
3rd year CSE-A

File not exists

SERVER SIDE:
socket creation is successful
bind is successful
listen is successful

server accepts client
Filename by client is x.cfile is not opened

CLIENT SIDE:
socket creation is successful

connected with server
enter filename to server
X.C
Message from server is
file not opened

RESULT:

Thus the implementation of file transfer application using TCP client and server was executed successfully.

Ex.no.6 GENERATION OF SIGPIPE ERROR WITH SOCKET

AIM:

To generate SIGPIPE error with socket.

ALGORITHM:

CLIENT:

1. Start.
2. Create socket using socket function and return value of that function is used to check socket is created or not.
3. The client can be connected with server using connect function and return value of the function is used to check the client connected with server or not.
4. If the client is connected then it give the message to the server using write function.
5. After server terminate, the client again write the socket that is not present. So sigpipe function is called and handled by server.
6. Stop.

SERVER:

1. Start.
2. Create socket using socket function and return value of that function is used to check socket is created or not.
3. Assign values to socket address structure that is server bind with that address using bind function and return value of that function is used to check the bind is success or not.
4. Now server is listening for a client request with backlog5 using listen function and return value of that function is used to check that the listen is successful or not.
5. The server accept client using accept function and using the return value we determine that the server accept client or not.
6. The server get message from client and echoed back to client.
7. Stop.

PROGRAM:

SERVER:

```
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<stdlib.h>
#include<string.h>
#define SA struct sockaddr
void func(int sockfd)
{
char buff[100];
int n;
bzero(buff,100);
read(sockfd,buff,sizeof(buff));
printf("message from client: %s\t",buff);
bzero(buff,100);
n=0;
printf("\nConnection closed");
close(sockfd);
}
int main()
{
int sockfd,connfd,len;
struct sockaddr_in servaddr,cli;
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd==-1)
{
printf("socket creation failed\n");
exit(0);
}
```

```
else
printf("Socket successfully created\n");
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
servaddr.sin_port=htons(43454);
if((bind(sockfd,(SA*)&servaddr, sizeof(servaddr)))!=0)
{
printf("bind failed\n");
exit(0);
}
else
printf("bind success\n");
if((listen(sockfd,5))!=0)
{
printf("listen failed\n");
exit(0);
}
else
printf("listen success\n");
len=sizeof(cli);
connfd=accept(sockfd,(SA *)&cli,&len);
if(connfd<0)
{
printf("accept failed\n");
exit(0);
}
else
printf("accept success\n");
func(connfd);
close(sockfd);
}
```

CLIENT:

```
#include<stdio.h>
```

```
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<signal.h>
#include<string.h>
#include<stdlib.h>
#define SA struct sockaddr
void sig_pipe(int_signum)
{
printf("SERVER TERMINATED PREMATURELY\n");
signal(SIGPIPE,sig_pipe);
}
void func(int sockfd)
{
char buff[100];
int n;
for(;;)
{
bzero(buff,sizeof(buff));
n=0;
while((buff[n++]=getchar())!='\n');
write(sockfd,buff,sizeof(buff));
bzero(buff,sizeof(buff));
}
}
int main()
{
int sockfd,connfd;
struct sockaddr_in servaddr,cli;
signal(SIGPIPE,sig_pipe);
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd==-1)
{
```

```
printf("socket creation failed\n");
exit(0);
}
else
printf("Socket successfully created\n");
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
servaddr.sin_port=htons(43454);
if(connect(sockfd,(SA *)&servaddr,sizeof(servaddr))!=0)
{
printf("connection failed\n");
exit(0);
}
else
printf("connection success\n");
func(sockfd);
close(sockfd);
}
```

OUTPUT:

server

bind success

listen success

accept success

enter message to client

hello

client

connection success

enter message to server

hai

message from server

hello

sigpipe caught

RESULT:

Thus the generation of SIGPIPE error with socket was executed successfully.

EX NO:7 RESTARTING SERVER BY CAPTURING SIGUP SIGNAL

AIM:

To implement restarting server by capturing sigup signal.

ALGORITHM:

1.Start

2.Include the required header files

3.If fork() return 0,then call the child function

4.Initialize the socket and set its attribute assign any port number as desired .

5.Call the sigaction() function and check return value.

6.If it is not equal to zero display the message.

7.Repeat the process upto system becomes off.

8.Stop.

PROGRAM:

```
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <unistd.h>
int isfirst;
char **command_args;
void sig_hangup(int signum)
{
        if ( isfirst )
        {
                isfirst = 0;
                fprintf(stderr, "Parent died\n");
        }
        else                                    /* Restart! */
        {
                fprintf(stderr, "Restarting...\n");
                /*** Kill all existing child processes ***/
                execv(command_args[0], command_args);
                fprintf(stderr, "Could not restart!!!\n");
                abort();
        }
}
void child(void)
```

```
{       struct sigaction act;
        bzero(&act, sizeof(act));
        act.sa_handler = sig_hangup;
        act.sa_flags = SA_RESTART;
        if ( sigaction(SIGHUP, &act, 0) != 0 )
                perror("Can't capture SIGHUP");
        for (;;)
        {
                fprintf(stderr, "[pid=%d] I'm still here\n", getpid());
                sleep(1);
        }
        exit(0);
}
int main(int count, char *strings[])
{
        isfirst = 1;
        command_args = strings;
        if ( fork() == 0 )
                child();
        sleep(1);
        return 0;
}
```

OUTPUT:

```
Cannot handle SIGKILL: Invalid argument
You can never handle SIGKILL anyway
Sleeping for 3 seconds
Cough SIGUP sleeping for 3 seconds
```

Try Sending another SIGUP/SIGINT/SIGALAM

Suspended() returned

SIGUSR is waiting

Sleeping for 3 seconds

Killed

KILL -HUP 4476

KILL –USRI 4476

KILL –PLRM 4476

KILL –SIGKILL 4476

RESULT

Thus the implementation of restarting server by capturing sigup signal executed successfully.

EX NO:8 PING AND TRACE ROUTE PROGRAM

AIM:

To develop a code for ping and trace route program.

ALGORITHM:

1.Start

2.Include the required header files

3.Declare the global variables which is necessary for the program.

4.Initialize the socket and set its attribute assign any port number as desired .

5.Call the send_packet() and receive_packet() function and check the return value.

6.If it is not equal to zero display the message.

7.Stop.

PROGRAM

```
#include <stdio.h>
#include <signal.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netdb.h>
#include <setjmp.h>
#include <errno.h>
#include<stdlib.h>
#include<string.h>
#define PACKET_SIZE     4096
#define MAX_WAIT_TIME   5
#define MAX_NO_PACKETS  3
char sendpacket[PACKET_SIZE];
char recvpacket[PACKET_SIZE];
int sockfd, datalen = 56;
int nsend = 0, nreceived = 0;
struct sockaddr_in dest_addr;
pid_t pid;
struct sockaddr_in from;
struct timeval tvrecv;
void statistics(int signo);
unsigned short cal_chksum(unsigned short *addr, int len);
```

```
int pack(int pack_no);

void send_packet(void);

void recv_packet(void);

int unpack(char *buf, int len);

void tv_sub(struct timeval *out, struct timeval *in);

void statistics(int signo)

{ printf("\n--------------------PING statistics-------------------\n");

printf("%d packets transmitted, %d received , %%%d lost\n", nsend,  nreceived, (nsend - nreceived) / nsend *
100);

close(sockfd);

exit(1); }

unsigned short cal_chksum(unsigned short *addr, int len)

{  int nleft = len;

    int sum = 0;

    unsigned short *w = addr;

    unsigned short answer = 0;

   while (nleft > 1)

{  sum +=  *w++;

       nleft -= 2 }

    if (nleft == 1)

    {*(unsigned char*)(&answer) = *(unsigned char*)w;

       sum += answer; }

   sum = (sum >> 16) + (sum &0xffff);

   sum += (sum >> 16);

   answer = ~sum;

   return answer; }

int pack(int pack_no)

{
```

```
    int i, packsize;
    struct icmp *icmp;
    struct timeval *tval;
    icmp = (struct icmp*)sendpacket;
    icmp->icmp_type = ICMP_ECHO;
    icmp->icmp_code = 0;
    icmp->icmp_cksum = 0;
    icmp->icmp_seq = pack_no;
    icmp->icmp_id = pid;
    packsize = 8+datalen;
    tval = (struct timeval*)icmp->icmp_data;
gettimeofday(tval, NULL);
    icmp->icmp_cksum = cal_chksum((unsigned short*)icmp, packsize);
    return packsize;
}
void send_packet()
{
    int packetsize;
    while (nsend < MAX_NO_PACKETS)
    {
        nsend++;
        packetsize = pack(nsend);
        if (sendto(sockfd, sendpacket, packetsize, 0, (struct sockaddr*)
&dest_addr, sizeof(dest_addr)) < 0)
        {
perror("sendto error");
            continue;
        } sleep(1);
```

```
    }
}
void recv_packet()
{
    int n, fromlen;
    extern int errno;
signal(SIGALRM, statistics);
    fromlen = sizeof(from);
    while (nreceived <nsend)
    {
        alarm(MAX_WAIT_TIME);
        if ((n = recvfrom(sockfd, recvpacket, sizeof(recvpacket), 0, (struct sockaddr*) &from, &fromlen)) < 0)
        {
            if (errno == EINTR)
                continue;
perror("recvfrom error");
            continue;
        } gettimeofday(&tvrecv, NULL);
        if (unpack(recvpacket, n) ==  - 1)
            continue;
        nreceived++;
    }
}
int unpack(char *buf, int len)
{
    int i, iphdrlen;
    struct ip *ip;
    struct icmp *icmp;
```

```
    struct timeval *tvsend;
    double rtt;
    ip = (struct ip*)buf;
    iphdrlen = ip->ip_hl << 2;
    icmp = (struct icmp*)(buf + iphdrlen);
    len -= iphdrlen;
    if (len < 8)
   {
printf("ICMP packets\'s length is less than 8\n");
return  - 1;
    }
    if ((icmp->icmp_type == ICMP_ECHOREPLY) && (icmp->icmp_id == pid))
    {
        tvsend = (struct timeval*)icmp->icmp_data;
        tv_sub(&tvrecv, tvsend);
        rtt = tvrecv.tv_sec * 1000+tvrecv.tv_usec / 1000;
printf("%d byte from %s: icmp_seq=%u ttl=%d rtt=%.3f ms\n", len,inet_ntoa(from.sin_addr), icmp->icmp_seq,
ip->ip_ttl, rtt);
    }
    else
return  - 1;
}
main(int argc, char *argv[])
{
    struct hostent *host;
    struct protoent *protocol;
    unsigned long inaddr = 0l;
    int waittime = MAX_WAIT_TIME;
```

```
    int size = 50 * 1024;
    if (argc < 2)
    {
printf("usage:%s hostname/IP address\n", argv[0]);
exit(1);
    } if ((protocol = getprotobyname("icmp")) == NULL)
    {
       perror("getprotobyname");
exit(1);
    }
    if ((sockfd = socket(AF_INET, SOCK_RAW, protocol->p_proto)) < 0)
    {
perror("socket error");
exit(1);
    }
   setuid(getuid());
setsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &size, sizeof(size));
bzero(&dest_addr, sizeof(dest_addr));
    dest_addr.sin_family = AF_INET;
    if (inaddr = inet_addr(argv[1]) == INADDR_NONE)
    {
       if ((host = gethostbyname(argv[1])) == NULL)
{   perror("gethostbyname error");
exit(1);
        }
       memcpy((char*) &dest_addr.sin_addr, host->h_addr, host->h_length);
    }
   else
```

```
        dest_addr.sin_addr.s_addr = inet_addr(argv[1]);
    pid = getpid();
printf("PING %s(%s): %d bytes data in ICMP packets.\n", argv[1], inet_ntoa
        (dest_addr.sin_addr), datalen);
    send_packet();
    recv_packet();
    statistics(SIGALRM);
    return 0;
}
void tv_sub(struct timeval *out, struct timeval *in)
{
    if ((out->tv_usec -= in->tv_usec) < 0)
{  --out->tv_sec;
        out->tv_usec += 1000000;
    } out->tv_sec -= in->tv_sec;
}
```

OUTPUT:

```
gcc ping.c
./a.out google.com
Ping google.com(216.58.197.98) 56 bytes in ICMP packets
64 Bytes from 216.58.197.98 ICMP-seq 1 ttl=55  rtt=3000 ms
64 Bytes from 216.58.197.98 ICMP-seq 2 ttl=55  rtt=2000 ms
64 Bytes from 216.58.197.98 ICMP-seq 3 ttl=55  rtt=1000 ms
----------------------PING statistics-----------------------
3 packets transmitted, 3 received, %0 loss
```

RESULT:

Thus the development of code for ping and trace execute program is executed successfully.

EX.NO:9 PERFORMANCE ANALYSIS OF TCP /UDP USING SIMULATION TOOL

AIM

To write TCP program to create a simulator demonstrating the connection between nodes and analysis performance for UDP/TCP connection.

ALGORITHM

1.Start

2.Create a simulator object and define different color for dataflows

3.Open the nam file using namtrace_all statement trace all event in the out1.nam file

4.Open the trace file in write only mode using namtrace_all statement trace all event in the out1.tr file

5.Define a finish procedure ,inside the finish procedure close the nam trace file and execute nam on the trace file

6.Create a topology that consists of four nodes and create links between the node and specify link characteristics

7.Set queue size of link to 10 and give node position for nam

8.Setup a UDP connection for that create udp agent and attach to n1 and create null agent attach that agent to the destination node and connect the agent

9.Setup a CBR over UDP connection for that create cbr application attach that to udp set characteristic of application

10.Specify the finish to call the finish procedure

11.Execute the events using run command

12.Stop

PROGRAM:

```
set ns [new Simulator]
$ns color 1 Green
$ns color 2 Violet
set nf [open out1.nam w]
$ns namtrace-all $nf
set nd [open out1.tr w]
$ns trace-all $nd
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out1.nam &
        exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
$ns queue-limit $n2 $n3 10
$ns duplex-link-op $n2 $n3 queuePos 0.5
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
```

```
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false
$ns at 0.1 "$cbr start"
$ns at 4.5 "$cbr stop"
$ns at 5.0 "finish"
$ns run
```

AWK SCRIPT:

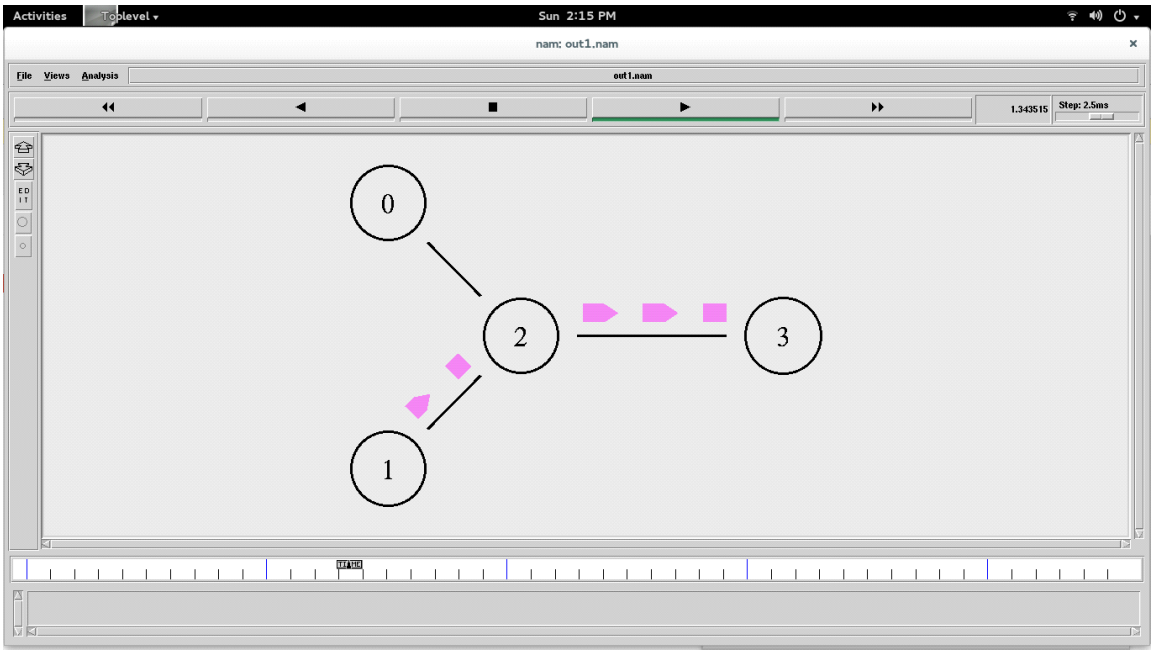
```
BEGIN{
 count= 0;
 countd= 0;
 counte= 0;
}

{
if($1=="r" && $5=="cbr")
count=count+1;
if($1=="-" )
countd=countd+1;
if($1=="+" )
counte=counte+1;
}
```

END{

printf("number of packets received in udp:%d\n packets enqued:%d \n packet dequeued:%d",count,countd,counte);

}

OUTPUT:



RESULT:

Thus the TCP program to create a simulator demonstrating the connection between nodes and analysis performance for UDP/TCP connection was executed successfully.

EX NO:10 SIMULATION OF LAN USING SIMILATION TOOLS

AIM

To write a TCL program to create LAN in TCP connection.

ALGORITHM

1.Start

2.Create an object for simulator and define a routing protocol

3.Create files out.nam and out.tr files and namtrace_all dump the traced format at respective files

4.Create a procedure finish with closing the opened file and execution of the command nam out.nam

5.Create the required nodes and create link between the nodes and define the orientation of the nodes

6.Create agent tcp and attach it at n0 and Create agent sink and attach it at n3

7.Set fid as 1 for this connection

8.Create application ftp attach it over tcp

9.After the application starts specify which link is down some specific amount of time and again up the link

10.Schedule the events and execute run

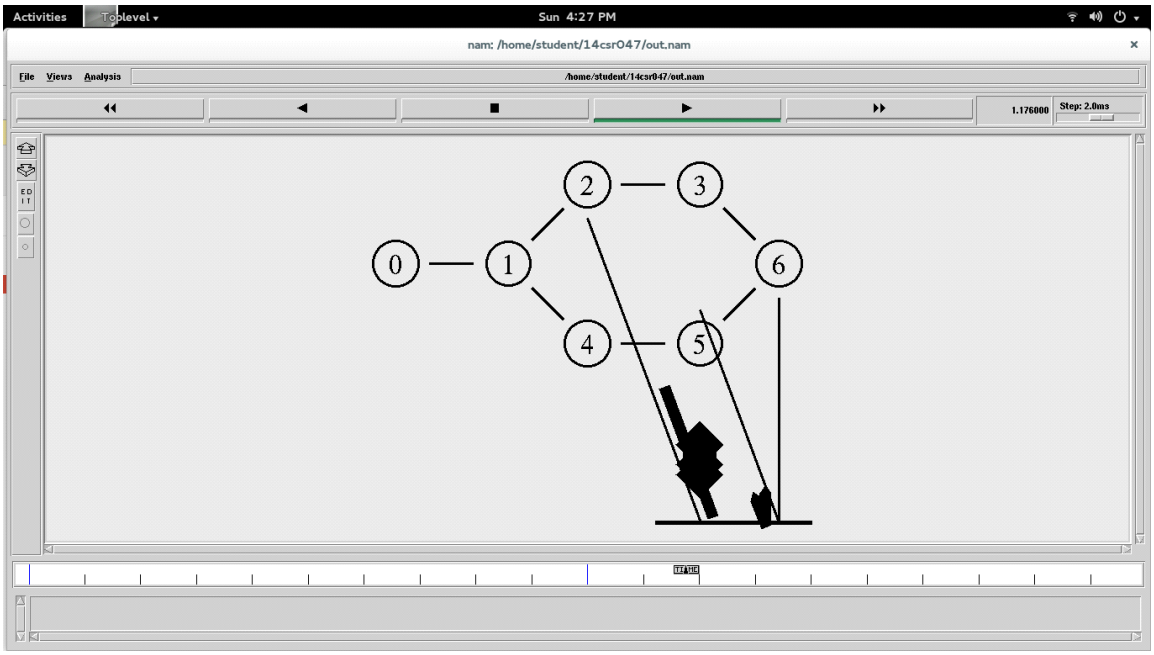
11.Stop

PROGRAM:

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set nd [open out.tr w]
$ns trace-all $nd
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
$ns duplex-link $n0 $n1 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 2Mb 10ms DropTail
$ns duplex-link $n1 $n4 2Mb 10ms DropTail
$ns duplex-link $n4 $n5 2Mb 10ms DropTail
$ns duplex-link $n3 $n6 2Mb 10ms DropTail
$ns duplex-link $n5 $n6 2Mb 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right
```

```
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n1 $n4 orient right-down
$ns duplex-link-op $n4 $n5 orient right
$ns duplex-link-op $n3 $n6 orient right-down
$ns duplex-link-op $n5 $n6 orient right-up
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n6 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
set lan [$ns newLan "$n2 $n5 $n6" 0.5Mb 100ms LL Queue/DropTail MAC/Csma/Cd Channel]
$ns at 0.1 "$ftp start"
$ns at 1.5 "$ftp stop"
$ns at 2.0 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n6 $sink"
$ns at 2.5 "finish"
$ns run
```

OUTPUT:



RESULT

Thus the LAN is created in TCP connection successfully.

EX NO: 11 SIMULATION OF DISTANCE VECTOR ROUTING PROTOCOL

AIM

To simulate distance vector or link state routing protocol.

ALGORITHM

1.Start

2.Create an object for simulator and define a routing protocol

3.Create files out.nam and out.tr files and namtrace_all dump the traced format at respective files

4.Create a procedure finish with closing the opened file and execution of the command nam out.nam

5.Create the required nodes and create link between the nodes and define the orientation of the nodes

6.Create agent tcp and attach it at n0 and Create agent sink and attach it at n3

7.Set fid as 1 for this connection

8.Create application ftp attach it over tcp

9.After the application starts specify which link is down some specific amount of time and again up the link

10.Schedule the events and execute run

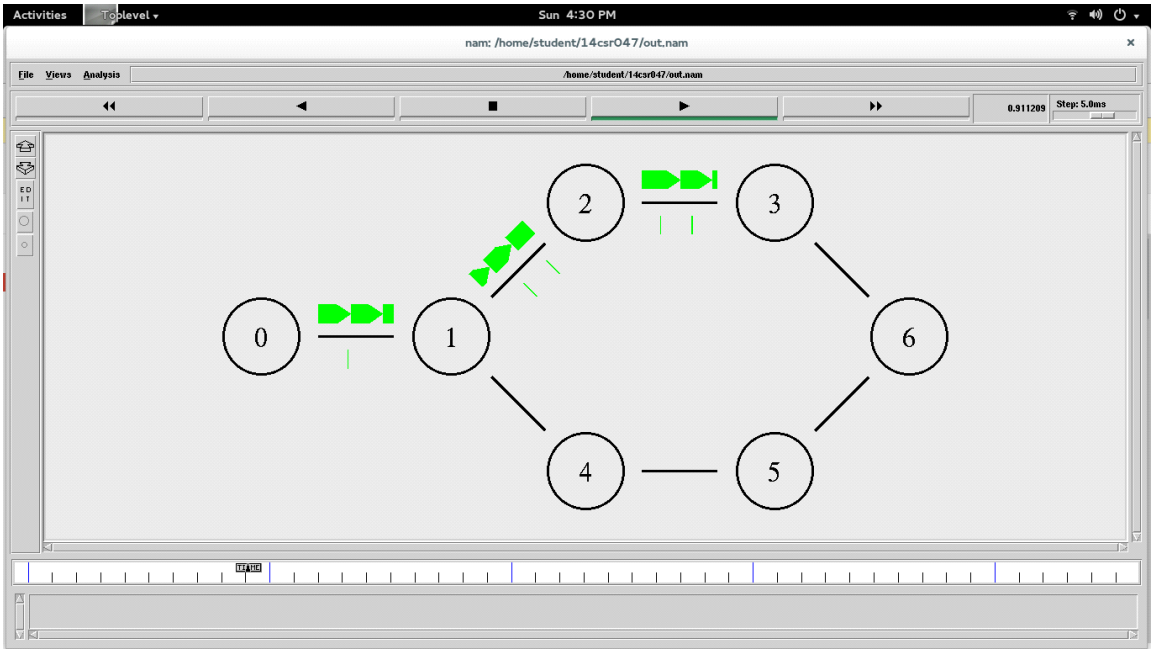
11.Stop.

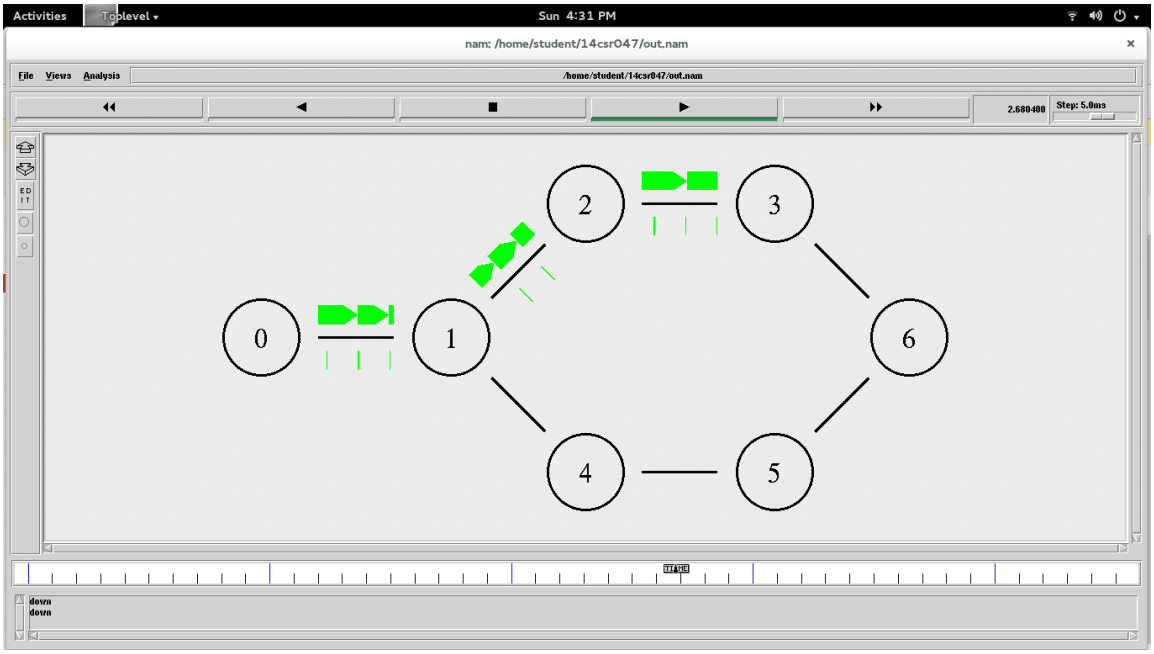
PROGRAM:

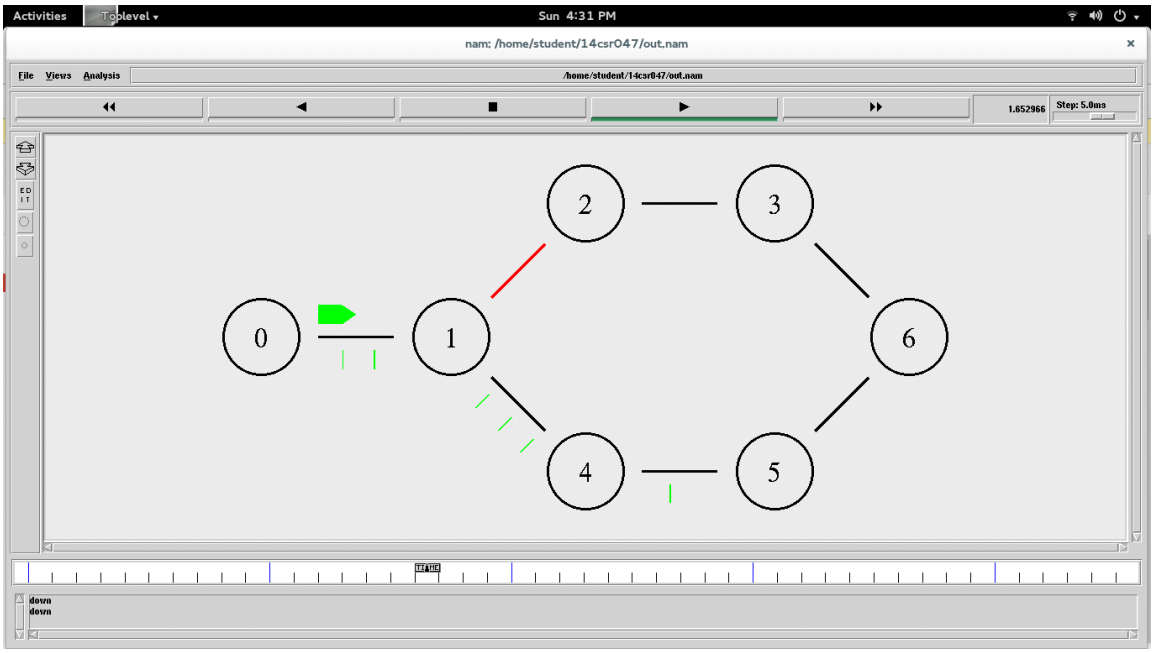
```
#Create a simulator object
set ns [new Simulator]

#Tell the simulator to use dynamic routing
$ns rtproto DV

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf


#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
           #Close the trace file
        close $nf
           #Execute nam on the trace file
        exec nam out.nam &
        exit 0
}

#Create seven nodes
for {set i 0} {$i < 7} {incr i} {
        set n($i) [$ns node]
}


#Create links between the nodes
for {set i 0} {$i < 7} {incr i} {
        $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}

#Create a UDP agent and attach it to node n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.025
$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node n(3)
set null0 [new Agent/Null]
```

```
$ns attach-agent $n(3) $null0

#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0

#Schedule events for the CBR agent and the network dynamics
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

OUTPUT:




Activities
Toplevel
Sun 4:31 PM
nam: /home/student/14csr047/out.nam
File
Views
Analysis
/home/student/14csr047/out.nam
2.680400
Step: 5.0ms
0
1
2
3
4
5
6



RESULT:

Thus the simulation of the distance vector routing protocol is executed successfully.

EX NO:12 PERFORMANCE ANALUSIS OF AODV,DSR,DSDV ROUTING PROTOCOL USING SIMULATION TOOL

AIM

To implement performance analysis of AODV,DSR,DSDV routing protocol using simulation tool.

ALGORITHM

1.Start

2.Create an object for simulator and define a routing protocol

3.Create respective files

4.Create a procedure finish with closing the opened file and execution of the command nam out.nam

5.Create the required nodes and create link between the nodes and define the orientation of the nodes

6.Create agent TCP and attach it at n0 and Create agent sink and attach it at n3

7.Set fid as 1 for this connection

8.Create application ftp attach it over TCP

9.After the application starts specify which link is down some specific amount of time and again up the link

10.Run the event

11.Repeat the steps for DSR,DSDV routing

12.Stop

PROGRAM:

```
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 50 ;# max packet in ifq
set val(nn) 4 ;# number of mobilenodes
set val(rp) AODV ;# routing protocol
set val(x) 1800 ;# X dimension of topography
set val(y) 840 ;# Y dimension of topography

    set ns_ [new Simulator]

    set tracefd [open wless.tr w]
    $ns_ trace-all $tracefd
$ns_ use-newtrace
    set namtrace [open wless.nam w]
    $ns_ namtrace-all-wireless $namtrace  $val(x) $val(y)

    set topo [new Topography]
    $topo load_flatgrid $val(x) $val(y)
    create-god $val(nn)
    set chan_1_ [new $val(chan)]
```

```
set dist(5m)  7.69113e-06
set dist(9m)  2.37381e-06
set dist(10m) 1.92278e-06
set dist(11m) 1.58908e-06
set dist(12m) 1.33527e-06
set dist(13m) 1.13774e-06
set dist(14m) 9.81011e-07
set dist(15m) 8.54570e-07
set dist(16m) 7.51087e-07
set dist(20m) 4.80696e-07
set dist(25m) 3.07645e-07
set dist(30m) 2.13643e-07
set dist(35m) 1.56962e-07
set dist(40m) 1.56962e-10
set dist(45m) 1.56962e-11
set dist(50m) 1.20174e-13
Phy/WirelessPhy set CSThresh_ $dist(50m)
Phy/WirelessPhy set RXThresh_ $dist(50m)


        $ns_ node-config -adhocRouting $val(rp) \
         -llType $val(ll) \
         -macType $val(mac) \
         -ifqType $val(ifq) \
         -ifqLen $val(ifqlen) \
         -antType $val(ant) \
```

```
            -propType $val(prop) \
            -phyType $val(netif) \
            -topoInstance $topo \
        -agentTrace ON \
            -routerTrace ON \
            -macTrace ON \
            -movementTrace ON \
            -channel $chan_1_ \
            -energyModel EnergyModel \
            -initialEnergy 50 \
            -rxPower 35.28e-3 \
            -txPower 31.32e-3 \
            -idlePower 712e-6 \
            -sleepPower 144e-9

    set Monitor [$ns_ node]

    set n2 [$ns_ node]
    set n3 [$ns_ node]
    set n4 [$ns_ node]
    set n5 [$ns_ node]



    set opt(seed) 0.1
    set a [ns-random $opt(seed)]
    set i 0
```

```
        while {$i < 5} {
        incr i
        }

        $Monitor set X_ 513.0
        $Monitor set Y_ 517.0
    $Monitor set Z_ 0.0



        $n2 set X_ 36.0
        $n2 set Y_ 529.0
        $n2 set Z_ 0.0

        $n3 set X_ 143.0
        $n3 set Y_ 666.0
        $n3 set Z_ 0.0

        $n4 set X_ 201.0
        $n4 set Y_ 552.0
        $n4 set Z_ 0.0

        $n5 set X_ 147.0
        $n5 set Y_ 403.0
        $n5 set Z_ 0.0
```

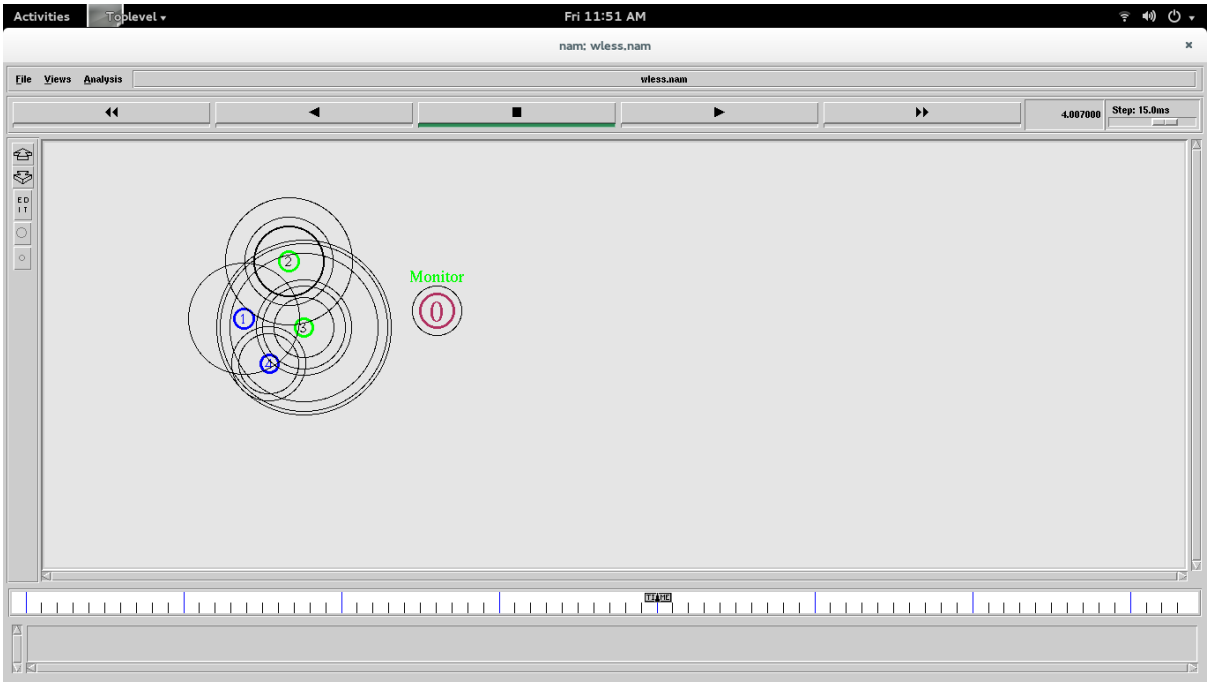
```
$ns_ at 0.75 "$n2 setdest 379.0 349.0 20.0"
$ns_ at 0.75 "$n3 setdest 556.0 302.0 20.0"
$ns_ at 0.20 "$n4 setdest 309.0 211.0 20.0"
$ns_ at 1.25 "$n5 setdest 179.0 333.0 20.0"

$ns_ initial_node_pos $Monitor 75
$ns_ initial_node_pos $n2 40
$ns_ initial_node_pos $n3 40
$ns_ initial_node_pos $n4 40
$ns_ initial_node_pos $n5 40

$ns_ at 0.0 "$Monitor label Monitor"

$Monitor color maroon
$ns_ at 0.0 "$Monitor color maroon"

$ns_ at 0.0 "$ns_ set-animation-rate 15.0ms"

$n5 color blue
$ns_ at 7.0 "$n5 color blue"
$n2 color blue
$ns_ at 7.29 "$n2 color blue"
```

```
        set udp7 [$ns_ create-connection UDP $n4 LossMonitor $n4 0]
        $udp7 set fid_ 1
        set cbr7 [$udp7 attach-app Traffic/CBR]
        $cbr7 set packetSize_ 1000
        $cbr7 set interval_ 5
        $ns_ at 4.0 "$cbr7 start"
        $ns_ at 4.1 "$cbr7 stop"
        set udp9 [$ns_ create-connection UDP $n4 LossMonitor $n3 0]
        $udp9 set fid_ 1
        set cbr9 [$udp9 attach-app Traffic/CBR]
        $cbr9 set packetSize_ 1000
        $cbr9 set interval_ 5
        $ns_ at 4.0 "$cbr9 start"
        $ns_ at 4.1 "$cbr9 stop"

        set udp10 [$ns_ create-connection UDP $n4 LossMonitor $n2 0]
        $udp10 set fid_ 1
        set cbr10 [$udp10 attach-app Traffic/CBR]
        $cbr10 set packetSize_ 1000
        $cbr10 set interval_ 5
        $ns_ at 4.0 "$cbr10 start"
        $ns_ at 4.1 "$cbr10 stop"
    proc stop {} {
                global ns_ tracefd
                $ns_ flush-trace
```

```
        close $tracefd
        exec nam wless.nam &
        exit 0
    }

puts "Starting Simulation........"
$ns_ at 25.0 "stop"
$ns_ run
```

AWK script for performance Analysis

OUTPUT :



AWK OUTPUT:

DSR protocol

No of pkts send	151
No of pkts recv	151
Pkt delivery ratio	100
Control overhead:	96

Normalized routing overheads	0.635762
Delay:	0.0243218
Throughput	295355
Jitter	0.029431
No of Pkts Dropped	0

DSDV protocol

No of pkts send	151
No of pkts recv 121	
Pkt delivery ratio	80.1325
Control overhead:	90
Normalized routing overheads	0.743802
Delay:	0.0170977
Throughput	241487
Jitter	0.0317323
No of Pkts Dropped	30

AODV protocol

No of pkts send	151
No of pkts recv 151	
Pkt delivery ratio	100
Control overhead:	255
Normalized routing overheads	1.68874
Delay:	0.0437258
Throughput	301262
Jitter	0.0399836
No of Pkts Dropped	0

RESULT Thus the simulation of the distance vector routing protocol is executed successfully.