Networks LAB

software testing (Anna University)

IMPLEMENTATION OF STOP AND WAIT PROTOCOL

EX NO: 01 A

DATE:

AIM:

To implement stop and wait protocol using c language.

ALGORITHM:

1. Start the program.
2. Import all the necessary packages.
3. Create 2 application sender and receiver.
4. Connect both applications using socket.
5. Sender port number and the frame is input as receiver.
6. Sender frame is send to the receiver and display to the receiver.
7. Frame is received and displays the acknowledgement and otherwise display the negative acknowledgement.
8. Receiver receives all the frames automatically and displays the message.
9. Close all the connections and terminate the program.

DESCRIPTION:

- **MODULE IN SENDER**

1. Establish connection to the server.
2. Send the frame to the receiver.
3. If it receive the acknowledgement and negative acknowledgement from the receiver.
4. If the receiver negative acknowledgement then once again send the frame to the receiver.
5. Close the sender side connection.

- **MODULE IN RECEIVER**

1. Establish connection to the server.
2. Receive the frame from the receiver.
3. Send acknowledgement and negative acknowledgement from the receiver.
4. Close the sender side connection.

PROGRAM FOR STOP RECEIVING:

//stoprecvhead.h

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<sys/socket.h>
#include<stdlib.h>
#include<unistd.h>
int receiver();
```

//IMPLEMENTATION FILE

```
int receiver()
{
int sd,con,port,i;
char content[30],ack[3];
struct sockaddr_in cli;
if((sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))==-1)
{
printf("\n socket problem");
return 0;
}
bzero((char*)&cli,sizeof(cli));
cli.sin_family=AF_INET;
printf("ENTER PORT NO");
scanf("%d",&port);
cli.sin_port=htons(port);
cli.sin_addr.s_addr=htonl(INADDR_ANY);
con=connect(sd,(struct sockaddr*)&cli,sizeof(cli));
if(con==-1)
{
printf("\n connection error");
return 0;
}
i=recv(sd,content,30,0);
while(strcmp(content,"EOF")!=0)
{
printf("received from sender:frame %s \n",content);
```

```
ph:
printf("acknowledgement(ACK/NAK):");
scanf("%s",ack);
if(!(strcmp(ack,"ack")==0||strcmp(ack,"nak")==0||strcmp(ack,"ACK")==0||
strcmp(ack,"NAK")==0))
{
printf("\n use ACK or NAK..\n");
goto ph;
}
send(sd,ack,5,0);
i=recv(sd,content,30,0);
}
printf("\n\n bye...");
close(sd);
return 0;
}
```

APPLICATION FILE FOR RECEIVER:

//RECEIVER.C

```
#include"stoprecvhead.h"
int main()
{
receiver();
}
```

PROGRAM FOR STOP SENDING:

//stopsendhead.h

//HEADER FILE

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
```

```
#include<sys/socket.h>
#include<stdlib.h>
#include<unistd.h>
int stopsender();

int stopsender()
{
int sd,i,r,bi,nsd,port,frame,prev_frame=0,count=0;;
char ack[5],buff[30];
struct sockaddr_in ser,cli;
if((sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))==-1)
{
printf("\n socket problem");
return 0;
}
printf("\nsocket created\n");
bzero((char*)&cli,sizeof(ser));
printf("ENTER PORT NUMBER:\n");
scanf("%d",&port);
printf("\n port address is  %d\n:",port);
ser.sin_family=AF_INET;
ser.sin_port=htons(port);
ser.sin_addr.s_addr=htonl(INADDR_ANY);
bi=bind(sd,(struct sockaddr*)&ser,sizeof(ser));
if(bi==-1)
{
printf("\nbind error,port busy,plz change port number");
return 0;
}
i=sizeof(cli);
listen(sd,5);
nsd=accept(sd,((struct sockaddr *)&cli),&i);
if(nsd==-1)
{
printf("\ncheck the description parameter\n");
return 0;
}
printf("\nconnection accepted.");
while(count<5)
{
```

```
ph:
printf("\n sendingFRAME %d to the receiver...\n",prev_frame);
snprintf(buff,sizeof(buff),"%d",prev_frame);
send(nsd,buff,30,0);
r=recv(nsd,ack,5,0);
if(strcmp(ack,"ack")==0 || strcmp(ack,"ACK")==0)
{
count++;
if(prev_frame==0)prev_frame=1;
else prev_frame=0;
}
else if(strcmp(ack,"nak")==0||strcmp(ack,"NAK")==0)
{
printf("\n NAK:so again sending the previous frame\n");
goto ph;
}
}
printf("\n bye");
send(nsd,"EOF",4,0);
close(sd);
close(nsd);
return 0;
}
```

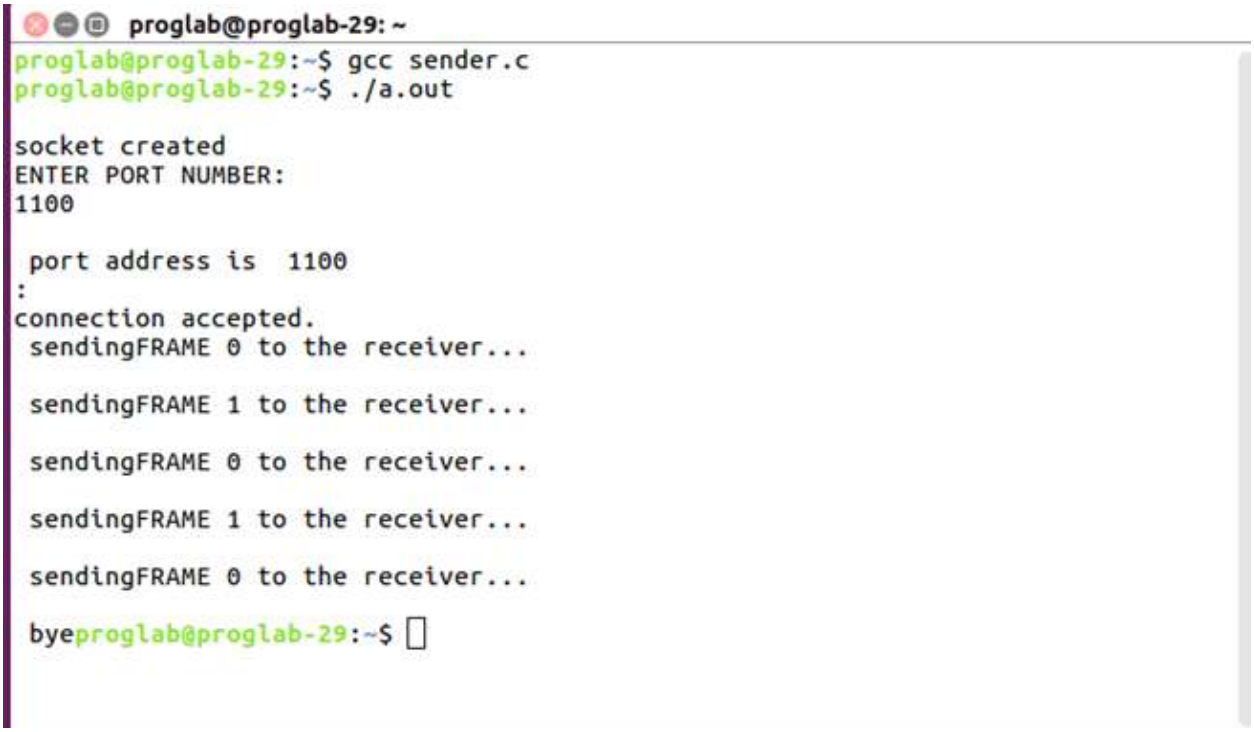
APPLICATION FILE FOR SENDER:

//sender.c

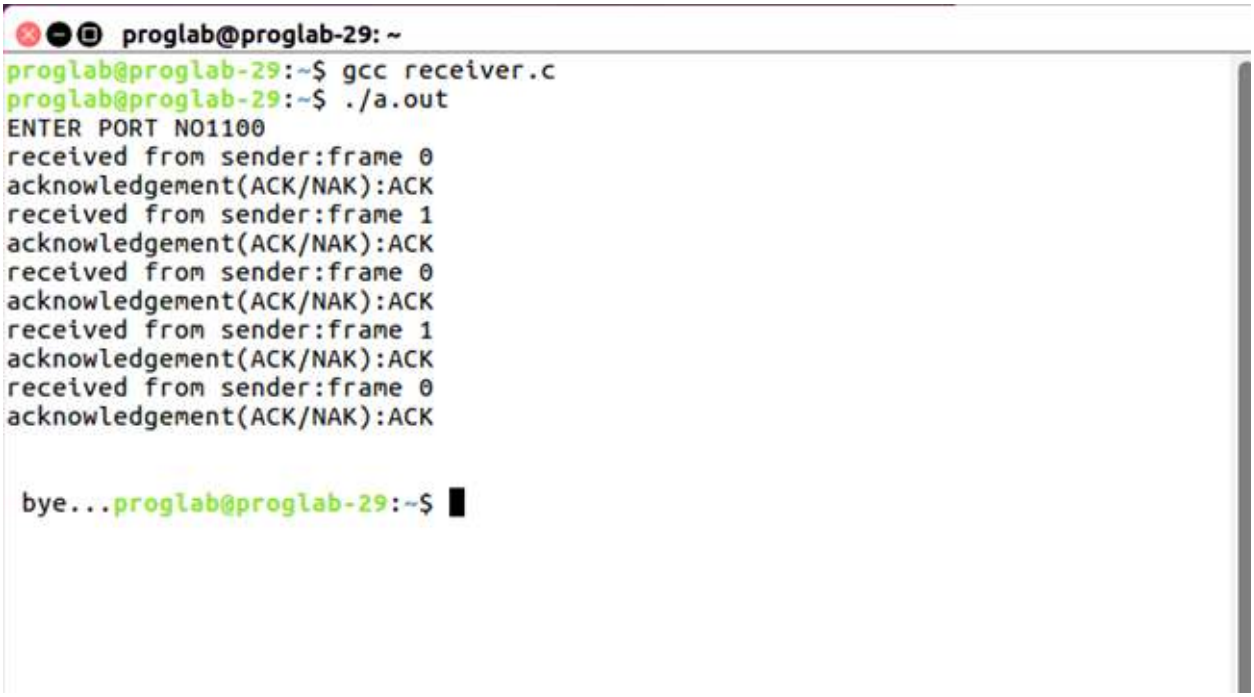
```
#include"stopsendhead.h"
int main()
{
stopsender();
}
```

OUTPUT:

SENDER:



RECEIVER:



RESULT:

Thus the stop and wait protocol program was done successfully and the output was verified.

IMPLEMENTATION OF SLIDING WINDOW PROTOCOL

EX NO: 01 B

DATE:

AIM:

To implement sliding window protocol using c program.

ALGORITHM:

1. Start the program.
2. Import the entire necessary package.
3. Create two applications sender and receiver.
4. Connect both applications using socket.
5. Sender data frame is send to the receiver.
6. Sender frame is send to receiver and display the frame to the receiver.
7. Frame is send to the receiver continuously when the end operation is enabled when the sender doesn't send any frame.
8. Then only the sender gets the acknowledgement and negative acknowledgement.
9. Close the program.

DESCRIPTION:

Module in sender:

1. Establish connection to the server.
2. Send the frame to the receiver.
3. Receive the acknowledgement and negative acknowledgement from the sender.
4. Close the connection.

Module in receiver:

1. Establish connection to the sender.
2. Receive the frame from sender.
3. Send the ACK and NAK from the receiver.
4. Close the receiver side connection.

PROGRAM FOR SENDER SIDE CONNECTION:

//**sender.c**

```
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<netdb.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
int main()

{
int sock,bytes_received,connected,true=1;i=1,s,f=0,sin_size,count;
char send_data[1024],data[1024],c,fr[30]="",ack[40];
struct sockaddr_in server_addr,client_addr;
if((sock=socket(AF_INET,SOCK_STREAM,0))==-1
{
perror("socket not created");
exit(1);
}
if(setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&true,sizeof(int))==-1)
{
perror("setsockopt");
exit(1);
}
server_addr.sin_family=AF_INET;
server_addr.sin_port=htons(17000);
server_addr.sin_addr.s_addr=INADDR_ANY;
if(bind(sock,(struct sockaddr*)&server_addr,sizeof(struct sockaddr))==-1)
{
perror("unable to bind");
exit(1);
}
if(listen(sock,5)==-1)
{
perror("listen");
```

```
exit(1);
}
fflush(stdout);
sin_size=sizeof(struct sockaddr_in);
connected=accept(sock,(struct sockaddr*)&client_addr,&sin_size);
while(strcmp(fr,"exit")!=0)
{
printf("enter data frame %d:(enter exit for end):",i);
scanf("%s",fr);
send(connected,fr,strlen(fr),0);
recv(sock,data,1024,0);
if(strlen(data)!=0)
//printf("i got an acknowledgement:%s\n",data);
fflush(stdout);
i++;
}
for(count=1;count<i+1;count++)
{
printf("enter the acknowledgement for frame %d:",count);
scanf("%s",ack);
if((strcmp(ack,"ack")))
{
printf("resending frame %d/n",count);
}
}
close(sock);
return (0);
}
```

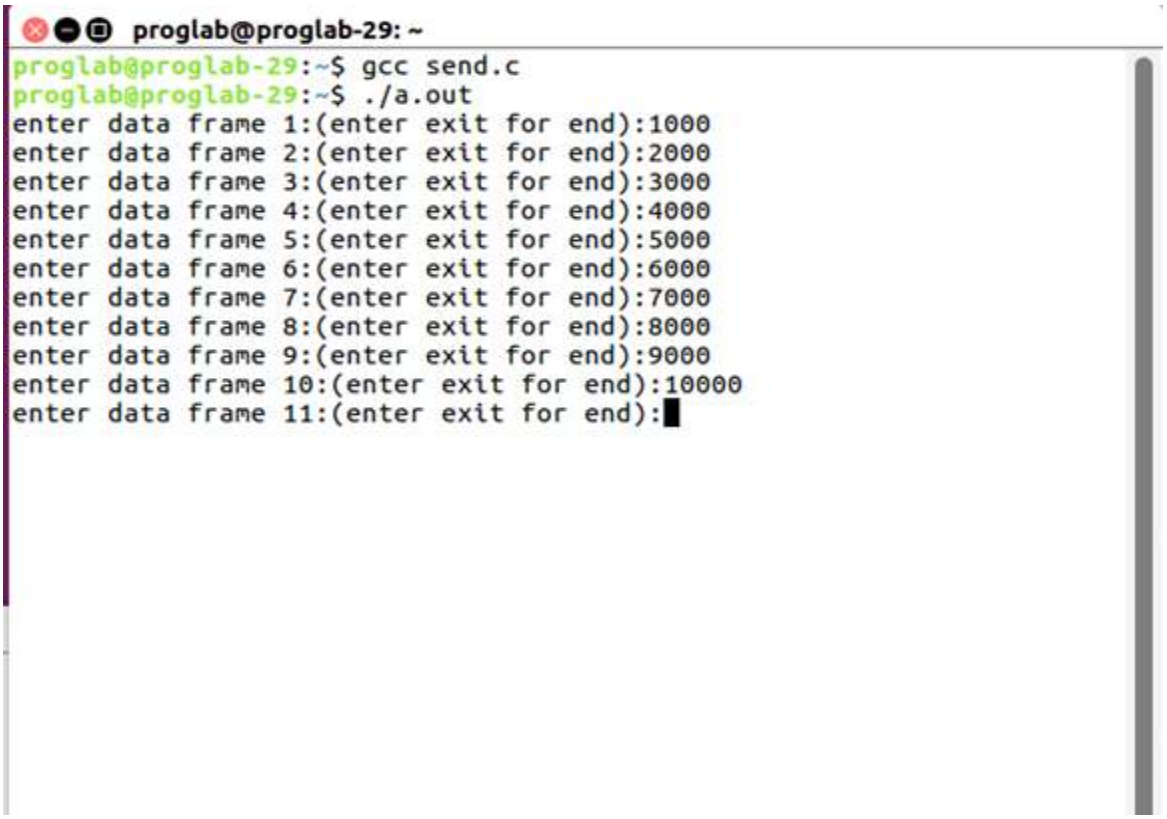
2.Program for receiver :

//**receiver.c**

```
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<netdb.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
int main()
{
int sock,bytes_received,i=1;
char receive[30];
struct hostent*host;
struct sockaddr_in server_addr;
host=gethostbyname("127.0.0.1");
if((sock=socket(AF_INET,SOCK_STREAM,0))==-1)
{
perror("socket not created");
exit(1);
}
printf("socket created");
server_addr.sin_family=AF_INET;
server_addr.sin_port=htons(17000);
server_addr.sin_addr=*((struct in_addr*)host->h_addr);
bzero(&(server_addr.sin_zero),8);
if(connect(sock,(struct sockaddr*)&server_addr,sizeof(struct sockaddr))==-1)
{
perror("connect");
exit(1);
}
while(1)
{
bytes_received=recv(sock,receive,20,0);
receive[bytes_received]='\0';
if(strcmp(receive,"exit")==0||strcmp(receive,"exit")==0)
```

```
{
close(sock);
break;
}
else
{
if(strlen(receive)<10)
{
printf("\n frame %d data %s received\n",i,receive);
send(0,"negative",10,0);
}
else
{
send(0,"negative",10,0);
}
i++;
}
}
close(sock);
return(0);
}
```

OUTPUT:



RESULT:

Thus the sliding window protocol was executed and the output was verified successfully.

STUDY OF SOCKET PROGRAMMING AND CLIENT_SERVER MODEL

EX NO: 02

DATE:

AIM:

To study about the basis of socket in network programming with example program.

NETWORK PROGRAMMING:

Network programming involves writing program that communicate with other programs across a computer network. One program is normally called the client and the other server. Common examples in TCP/IP are web clients (browsers) and web servers.

To facilitate communication between unrelated processors and to standardize network programming on API is needed.

There are to such APIs:-

1. Socket sometimes called “Berkeley Sockets”.
2. XIT(x/open transport interface).

SOCKET:

In TCP/IP an addressable point that consists of an IP address and a TCP or VDP port number that can provides application with access to TCP/IP protocol is called socket. A socket is an abstraction that represents an end point of communication. The complete set of operation that can be performed on a socket constitutes the socket API (Application Programming Interface).

STRUCTURE:

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

```
Struct sockaddr
 {
 Unsigned short sd_family;
 Char sa_data[14];
 }
```

IMPORTANT FUNCTIONS:

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);

i)Family:

Specify the protocol family and is one of the constant below.

Family	Description
AE_INET AE_INET AF_LOCAL AF_ROUTE AF_KEY	IPv4 protocols IPv6 protocols Unix domain protocols Routing sockets Key sockets

ii)type:

Indication communication semantics.

Sock_stream - stream socket
Sock_DGRM - data gram socket
Sock_raw – raw socket

iii)protocol:

set to 0 except of raw sockets.

Return value: on success: socket description

(a small non_negative integer)

Example: SOCKET(AF_INET,SOCK_STREAM.IPPROTO_TCP);

iii)bind():

The bind function assigns local protocol address to a socket. The protocol address is a combination of either 32 bit IPv4 address or a 128_ bit IPv6 address along with a 16 bit TCP or UDP port number.

#include<sys/socket.in>

int bind(int sockfd . const struct Sockaddr * my addr , socklen_t , addrlen);

Parameter	**Description**
Sockfd	Socket description return by the socket function
*my addr	A pointer to a protocol specific address.
Addrlen	The size of the socket address structure.
Return value	On success -0 On error -1
Example	Bind(sock fd,(struct Sockaddr *)& my addr,size of(struct Sockaddr));

SOCKET:

The connection 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*)

Serv_addr, socklen_addrlen

Parameter	**Description**
Sockfd	Socket description socket function
*servaddr	A pointer to a socket address structure.
Addrlen	The size of the socket address structure.
Return value	On success -0 On error -1
Example	Sd=socket(AF_INET,sock_STREAM,IPPROTO_TCK);

	Connect(sd,struct sockaddr *)&cli, Size of(cli));

LISTEN():

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

#include<sys/socket.in>

int listen(int sockfd,int backlog);

Parameter	**Description**
Sockfd	Socket description returned by a socket function.
Backlog	Maximum number of connections that the kernel should queue for the socket.
Return value	On success -0 On error -1
Example	Listen(sd,5)

ACCEPT:

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

#include<sys/socket.h>

int accept (int sockfd,struct sockaddr*(liaddr,sock len_t*addr)

Parameter	**Description**
Sockfd	Sockfd description returned by the socket function.
*cliaddr	Used to return the protocol address of the connection per process
*addrlen	Length of the address.
Return value	On success: a new (connected socket descriptor)

example	on error:-1 accept(:sizeof(cli):(sd,((struct sockaddr &cli,cli*)));

CLOSE:

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

#include<unistd.h>

int close(int Sockfd)

Parameter	Description
Sockfd	Socket descriptor returned by the socket.
Return value	On success:-0 On error:-1
Example	Sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP); Close(sd);

READ():

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

#include<unistd.h>

Size_t read(int sockfd,const void *buf,size_t n bytes);

Parameter	Descriptor
Sockfd	Socket descriptor returned by the socket function.
Buf	Buffer to store the data
n bytes	Size of the buffer.
return value	0 on EDF -1 on error
example	Read (sock fd,recv buff,size of(recvbuff)-1))

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 n bytes);

SEND TO():

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

#include<sys/socket.h>

Size_t send to(int sockfd,const void *buff,size_t nbytes,Int flag,const

struct sockaddr *to,socklen_t addrlen);

Parameter	**Description**
Sockfd	Socket descriptor
*buff	Pointer to buffer to write form
n bytes	Number of bytes to write
to	Socket addresses structure containing the protocol address of where the data is to be sent.
addrlen	Size of the socket addresses structure.
return value	Number of bytes read or written if oks -1 on error.

RECV FROM:

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

#include<sys socket.h>

Size_t recv from(int sockfd,void *buff,size_t n bytes,int flag,struct

Sockaddr *from,socklen_t *addrlen);

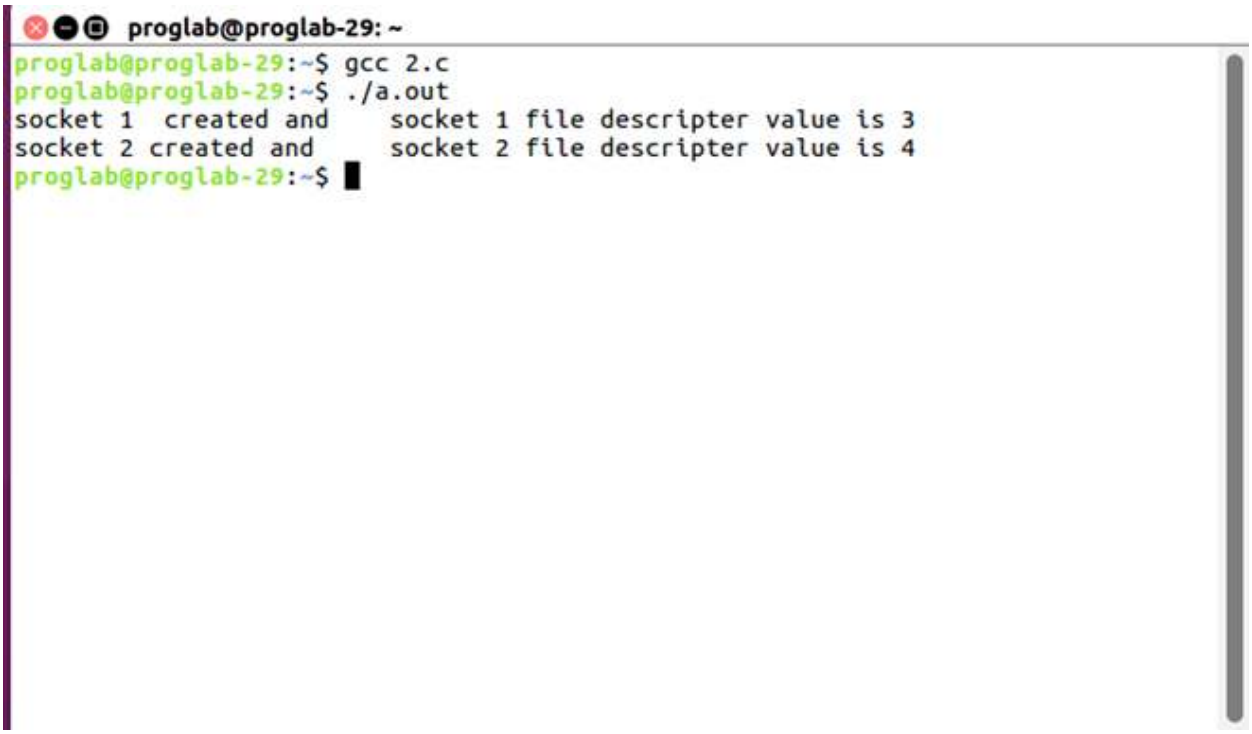
Parameter	Description
Sockfd	Socket descriptor
*buff	Pointer to buffer to read
n bytes	Number of bytes to read
from	Socket address structure who sent the datagram.
addrlen	Size of the socket data structure.
return value	Number of bytes read or written if ok, -1 an error.

PROGRAM TO CREATE A SOCKET:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#incluide<sys/socket.h>
#include<netinwet/in.h>
#include<arpa/inet.h>
main()
{
int sockfd1,sockfd2;
sockfd1=socket(AF_INET,SOCK_STREAM,0);
sockfd2=socket(PF_INET,SOCK_DGRAM,0);
if(sockfd11==-1)
{
printf("socket1 not created\n");
}
```

```
else
{
printf("socket 1  created and \t socket 1 file descripter value is %d \n",sockfd1);
}
if(sockfd2==-1)
{
printf("socket  2 creation error\n");
}
else
{
printf("socket 2 created and \t socket 2 file descripter value is %d \n",sockfd2);

}
}
```

OUTPUT:

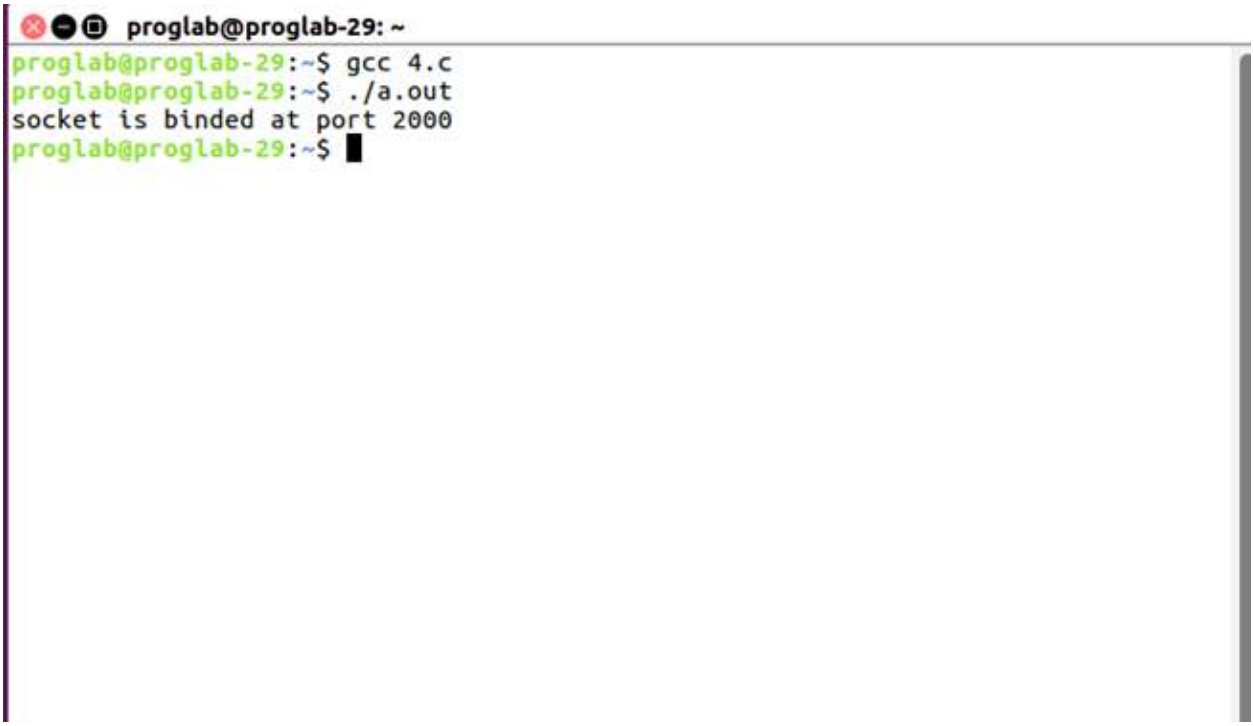


PROGRAM TO BIND A SOCKET:

```
#include<stdio.h>
#include<sys/socket.h>
#include<string.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("socket 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("socket is binded at port %d\n",i);
}
else
{
printf("binding error\n");
}
}
```

OUTPUT:

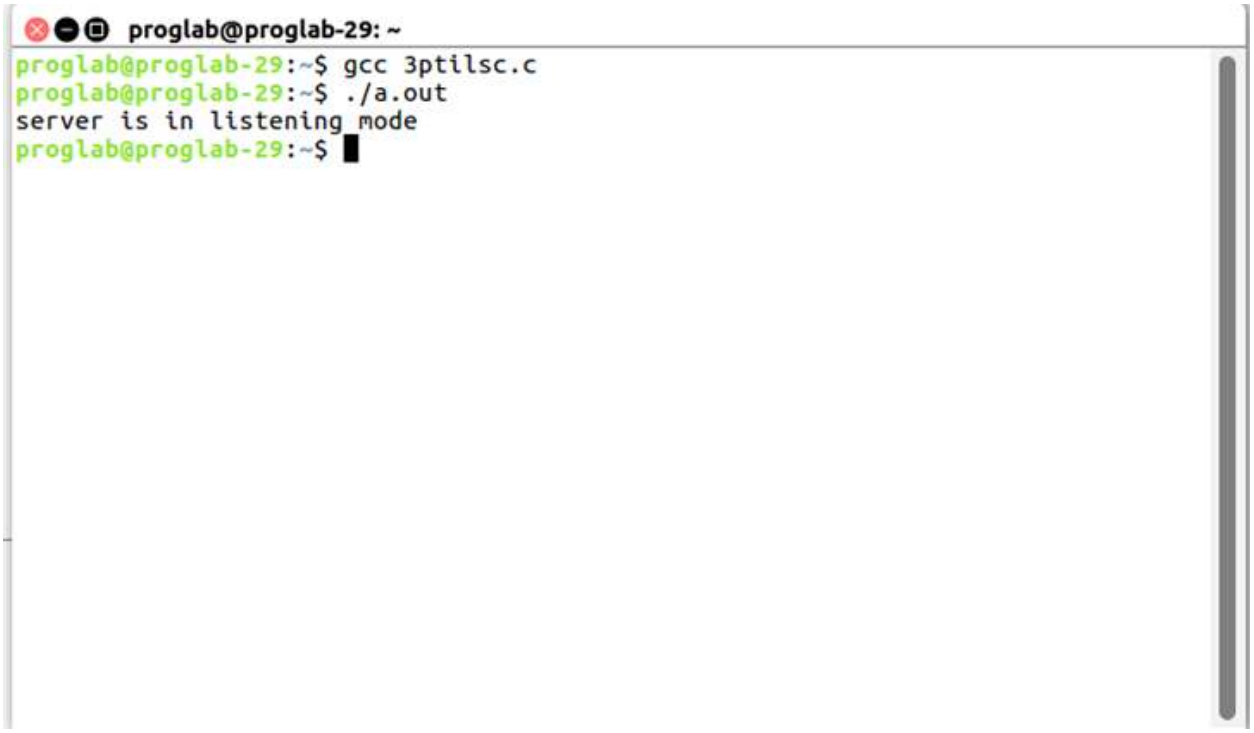


PROGRAM TO IMPLEMENT LISTEN() SYSTEM CALL:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
```

```
#include<string.h>
#include<stdlib.h>
#define PORT 3550
#define BACKLOG 12
int 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("socket() 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("bind() error\n");
exit(-1);
}
x=listen(fd,BACKLOG);
if(x==-1)
{
printf("listen() error\n");
exit(-1);
}
else
{
printf("server is in listening mode\n");
}}
```

OUTPUT:



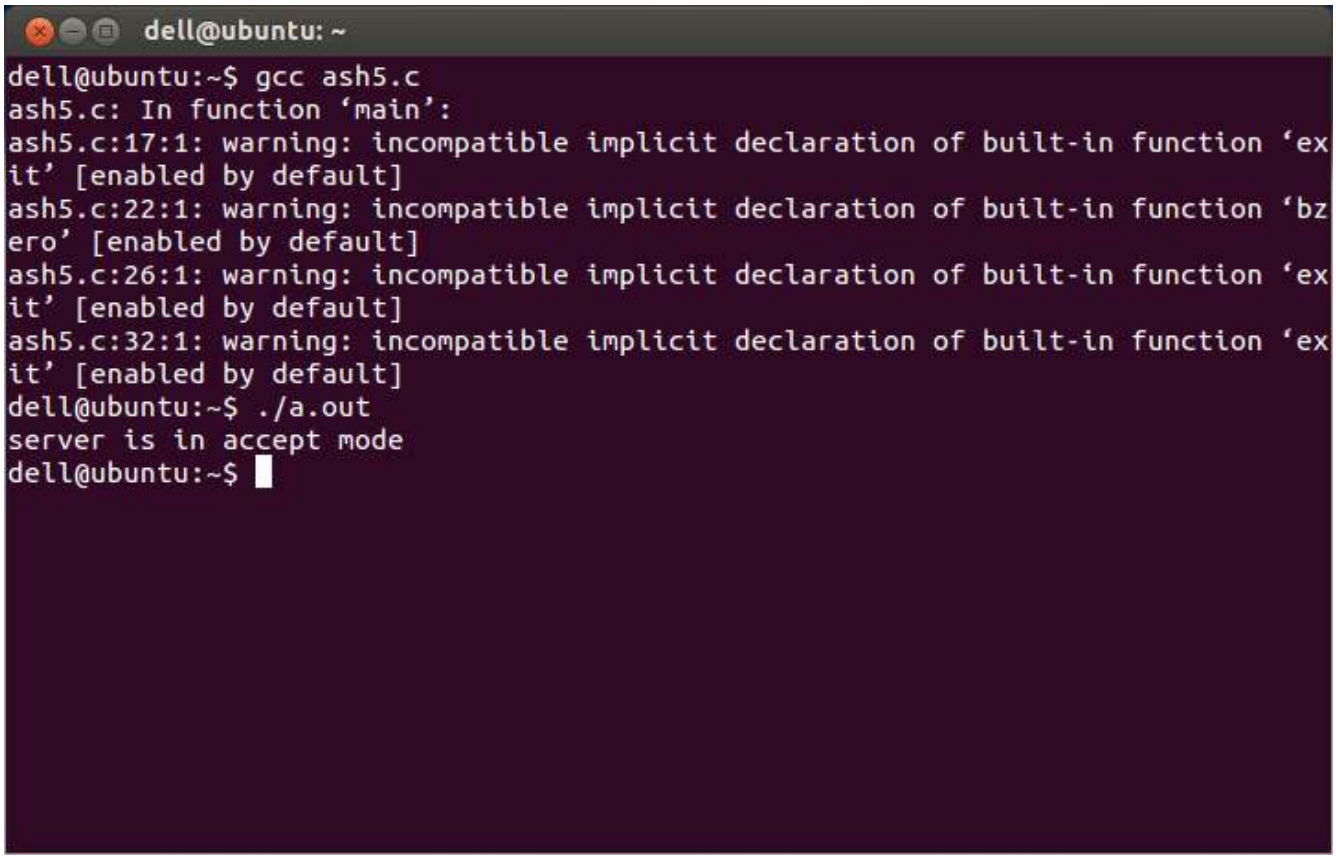
PROGRAM TO IMPLEMENT ACCEPT() SYSTEM CALL:

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

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
#define PORT 3550
#define BACKLOG 12
int main()
{
int fd,fd2;
struct sockaddr_in server;
struct sockaddr_in client;
int sin_size;
if((fd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
printf("socket()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("bind()error\n");
exit(-1);
}
if(listen(fd,BACKLOG)==-1)
{
printf("listen()error\n");
exit(-1);
}
printf("server is in accept mode\n");
while(1)
{
sin_size=sizeof(struct sockaddr_in);
if((fd2=accept(fd,(struct sockaddr*)&client,&sin_size))==-1)
{
printf("accept()error\n");
exit(-1);
}
```

```
else
printf("serve is in accept mode\n");
printf("you got a connection from %s\n",inet_ntoa(client.sin_addr));
}
}
```

OUTPUT:



RESULT:

Thus the program for socket programming and client server model was studied and the output was verified successfully.

IMPLEMENTATION OF ARP (or) RARP

EX NO: 03

DATE:

AIM:

To write a c program to implement ARP or RARP protocols.

ALGORITHM:

1. Start the program.
2. Import the entire necessary package.
3. Create two application server and client.
4. Connect both applications.
5. Send the data input to the client through the server.
6. Server frame is send to the client and display the frame to the client.
7. Send the frames to the client until the server enters exit.

DESCRIPTION:

Module in server:

i) Establish the connection to the server.
ii) Enter the mac address and IP in server until the connection exit.
iii) Display details

Module in client:

i) Establish the connection to client.
ii) Receive the frame from server.
iii) Enter the operation which is to be performed.
iv) If option 1 is selected, it gets the MAC address and display the IP address.
v) If option 2 is selected, it gets the address and display the MAC address.
vi) If option 3 is selected it exit from the client operation.

PROGRAM FOR SENDER SIDE CONNECTION:

//ARP SERVER:

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int shmid,a,i;
char *ptr,*shmptr;
shmid=shmget(3000,10,IPC_CREAT|0666);
shmptr=shmat(shmid,NULL,0);
ptr=shmptr;
for(i=0;i<3;i++)
{
printf("\n enter the %d mac address:",i+1);
scanf("%s",ptr);
a=strlen(ptr);
printf("\n string length:%d",a);
ptr[a]=' ';
puts("\n enter ip address:");
ptr=ptr+a+1;
scanf("%s",ptr);
ptr[a]='\n';
ptr=ptr+a+1;
}
ptr[strlen(ptr)]='\0';
printf("\n ARP table at serviceside is=\n%s",shmptr);
shmdt(shmptr);
}
```

PROGRAM FOR RECEIVER SIDE CONNECTION:

//ARP CLIENT:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/shm.h>
int main()
{
int shmid,a,i;
char *ptr,*shmptr;
char ptr2[51],ip[12],mac[26];
shmid=shmget(3000,10,0666);
shmptr=shmat(shmid,NULL,0);
puts("the arp table is");
printf("%s",shmptr);
while(1)
{
printf("\n.....");
printf("\n OPERATIONS");
printf("\n.....");
printf("\n 1.APR\n 2.RAR\n 3.EXIT\n");
printf("\n enter your choice:");
scanf("%d",&a);
switch(a)
{
case 1:
puts("enter ip address");
scanf("%s",ip);
ptr=strstr(shmptr,ip);
ptr-=8;
sscanf(ptr,"%s%*s",ptr2);
printf("\n mac addr is %s",ptr2);
break;
case 2:
puts("enter mac addr");
scanf("%s",mac);
ptr=strstr(shmptr,mac);
sscanf(ptr,"%*s%s",ptr2);
printf("\n IP address is:%s",ptr2);
break;
case 3:
```

```
exit(1);
}
}
}
```

OUTPUT:

//ARP SERVER:-

```
proglab@proglab-29: ~
proglab@proglab-29:~$ gcc pfssc.c
proglab@proglab-29:~$ ./a.out

 enter the 1 mac address:aswinth

 string length:7
 enter ip address:
1.3.5.7

 enter the 2 mac address:cseucen

 string length:7
 enter ip address:
2.4.6.8

 enter the 3 mac address:jerlin

 string length:6
 enter ip address:
1.2.5.8

 ARP table at serviceside is=
aswinth 1.3.5.7
cseucen 2.4.6.8
```

//ARP CLIENT:-

```
proglab@proglab-29: ~
proglab@proglab-29:~$ gcc pfrsc.c
proglab@proglab-29:~$ ./a.out
the arp table is

.........
 OPERATIONS
......
 1.APR
 2.RAR
 3.EXIT

 enter your choice:1
enter ip address
2.4.6.8

 mac addr is cseucen
.........
 OPERATIONS
......
 1.APR
 2.RAR
 3.EXIT

 enter your choice:1
```

RESULT:

Thus the implementation of ARP and RARP program was executed and the output was verified successfully.

TRACE ROUTE

EX NO: 04

DATE:

AIM:

To implement the trace route program using c program.

ALGORITHM:

1. Start the program
2. Import all necessary packages.
3. Create the bath between the client machine and remote path.
4. Enter the available trace route into it.
5. Print the path in which source should reach the destination.
6. Display the value of trace route from destination.
7. Stop the program.

1. PROGRAM TO IMPLEMENT TRACE ROUTE:-

//TRACE ROUTE:-

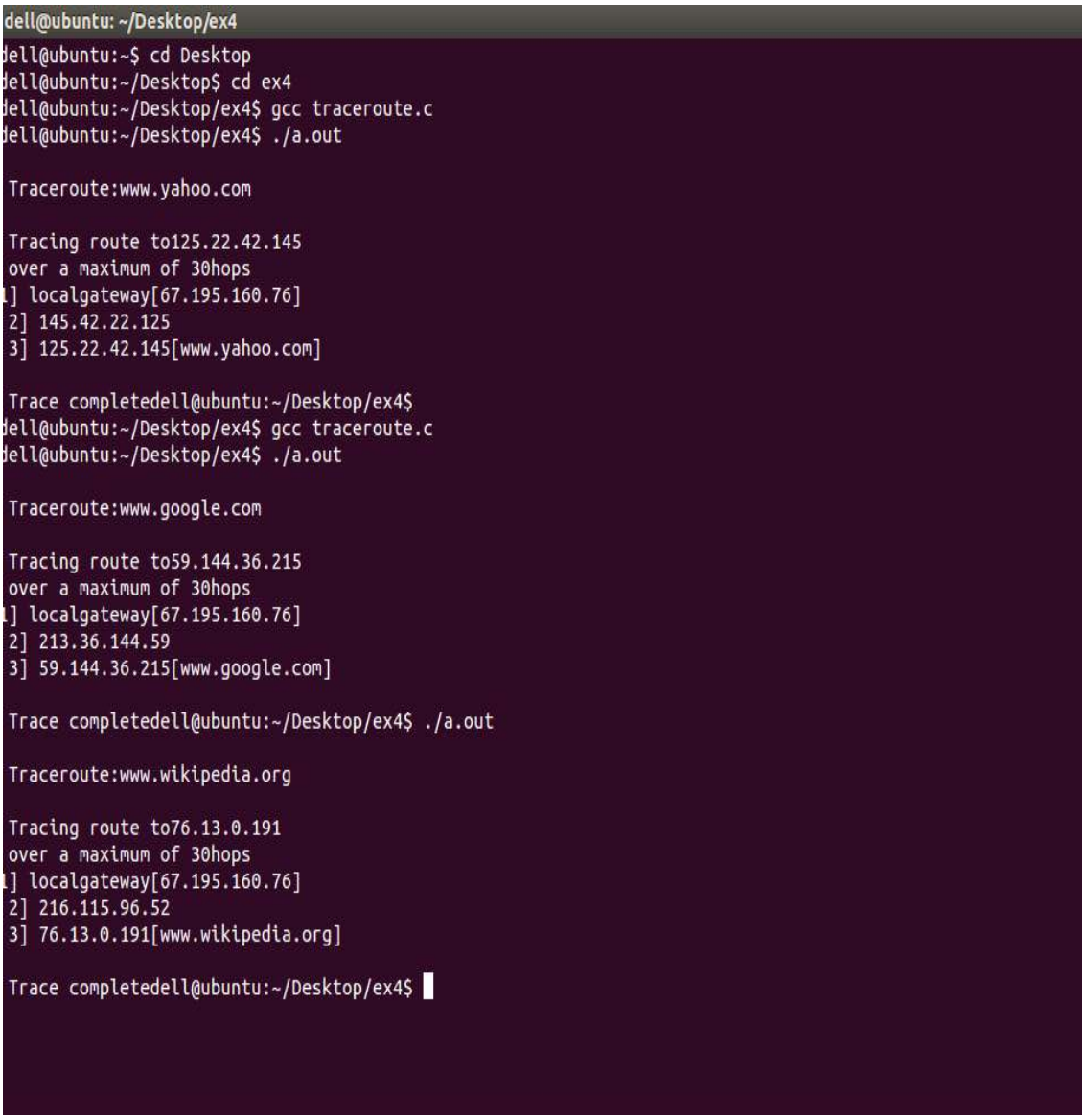
```
#include<stdio.h>
    #include<string.h>
    #include<stdlib.h>
    int main()
    {
char ip1[25],ip2[25],ip3[25],ip4[25],ip5[25];
char destn[25];
FILE *fp;
printf("\nTraceroute: ");
scanf("%s",&destn);
fp=fopen("path.txt","r");
while(!feof(fp))
{
fscanf(fp,"%s\t\t%s\t\t\t%s\t\t%s\t\t%s\n",&ip1,&ip2,&ip3,&ip4,&ip5);
if((strcmp(destn,ip4)==0)||(strcmp(destn,ip5)==0))
{
printf("\nTracing route to %s \n over a maximum of 30 hops",ip4);
printf("\n1] %s \n2] %s \n3] %s [ %s ]\n",ip2,ip3,ip4,ip5);
printf("\nTrace complete");
exit(0);
}
}
```

```
return 0;

}
```

//path.txt

```
3.21.191.19  LocalGateway[67.195.160.76]     145.42.22.125 125.22.42.145
www.yahoo.com

3.21.191.19  LocalGateway[67.195.160.76]     213.36.144.59 59.144.36.215
www.google.com

3.21.191.19  LocalGateway[67.195.160.76]     216.115.96.52 76.13.0.191
www.wikipedia.org
```

OUTPUT:-



RESULT:-

Thus the trace route program was executed successfully.

SOCKET CREATION FOR HTTP

EX NO: 05

DATE:

AIM:-

To implement socket creation of http using c program.

ALGORITHM:-

1. Start the program.
2. Import the necessary packages.
3. Create two new application server and client.
4. Connect both applications.
5. Send the input to client through server.
6. Get the output in client.

DESCRIPTION:-

Module in server:-

1. Establish the connection to server and client.
2. Enter the web address in server.

Module in client:-

1. Establish the connection to server and client.

2. After getting input from client display the particular client address http.doc
3. Stop the program.

PROGRAM:-

INPUT FILE:-

www.google.com

www.sify.com

www.rediffmail.com

www.gmail.com

SERVER:-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<string.h>
#include<netinet/in.h>
#include<stdlib.h>
int main()
{
struct sockaddr_in local;
int s,s1,rc,l=0;
FILE *fin,*fout;
char buf[2000],chec[2000];
local.sin_family=AF_INET;
local.sin_port=htons(14000);
```

```
local.sin_addr.s_addr=inet_addr("127.0.0.1");
s=socket(AF_INET,SOCK_STREAM,0);
if(s<0)
{
printf("SOCKET CALL FAILURE\n");
exit(1);
}
rc=bind(s,(struct sockaddr*)&local,sizeof(local));
if(rc<0)
{
printf("BIND CALL FAILURE\n");
exit(1);
}
rc=listen(s,5);
if(rc)
{
printf("LISTEN CALL FAILED\n");
exit(1);
}
s1=accept(s,NULL,NULL);
if(s1<0)
{
printf("ACCEPT CALLL FRIEND\n");
exit(1);
}
rc=recv(s1,buf,2000,0);
if(rc<0)
{
printf("RECEIVE CALL FAILED\n");
exit(1);
}
fin=fopen("http.doc","r");
while(!feof(fin))
{
fscanf(fin,"%s",chec);
if(!(strcmp(buf,chec)))
{
break;
}
}
```

```
if(!feof(fin))
{
fscanf(fin,"%s",chec);
rc=send(s1,chec,2000,0);
fclose(fin);
}
else
{
fclose(fin);
rc=send(s1,"failure",2000,0);
}
fout=fopen(chec,"r");
if(fout<0)
{
strcpy(buf,"failure");
}
else
{
while(!feof(fout))
{
fscanf(fout,"%s",chec);
printf("SUCCESS\n");
send(s1,chec,2000,0);
fclose(fout);
}
}
rc=send(s1,"over",2000,0);
if(rc<=0)
printf("SEND CALL FAILED\n");
exit(0);
}
```

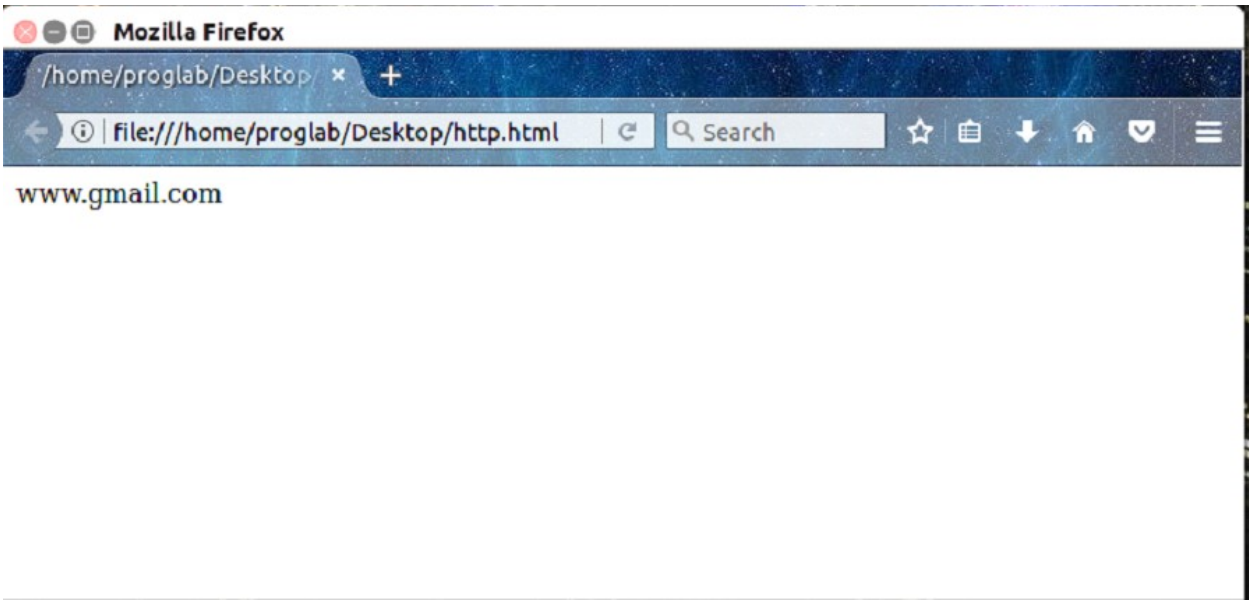
CLIENT:-

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdio.h>
```

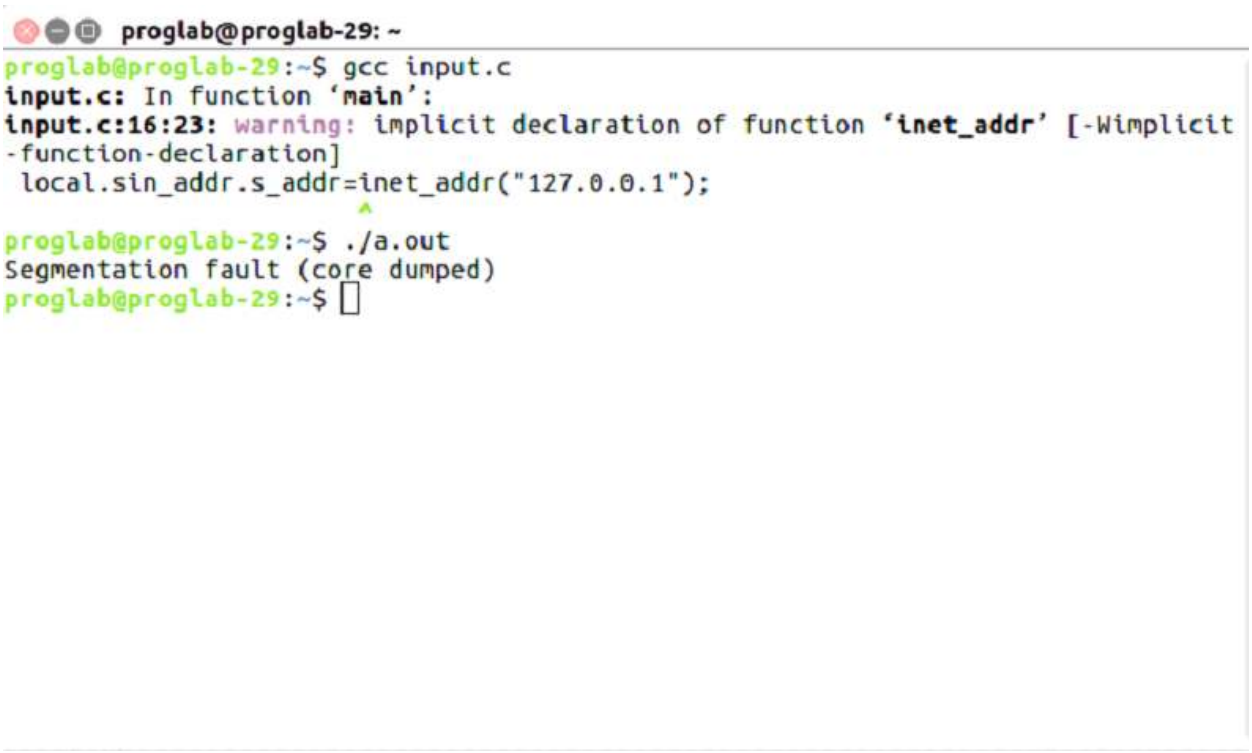
```
#include<stdlib.h>
int main()
{
struct sockaddr_in peer;
int s,rc,i=0;
FILE *fin;
char buf[2000],se[2000];
peer.sin_family=AF_INET;
peer.sin_port=htons(14000);
peer.sin_addr.s_addr=inet_addr("127.0.0.1");
s=socket(AF_INET,SOCK_STREAM,0);
if(s<0)
{
printf("SOCKET CALL FAILED\n");
exit(1);
}
rc=connect(s,(struct sockaddr*)&peer,sizeof(peer));
if(rc)
{
printf("CONNECTION CALL FAILED");
exit(1);
}
printf("ENTER THE WEB ADDRESS:");
scanf("%s",se);
rc=send(s,se,2000,0);
if(rc>=0)
{
printf("SEND CALL FAILED\n");
exit(1);
}
else
{
rc=recv(s,buf,2000,0);
if(rc<=0)
{
printf("RECEIVE CALL FAILED\n");
}
if(!strcmp(buf,"failure"))
{
printf("INVALID WEB\n");
```

```
}
else
{
fin=fopen("http.html","a");
rewind(fin);
fputs(buf,fin);
puts("FILE IS SUCCESSFULY OBTAINED\n");
}
}
exit(0);
}
```

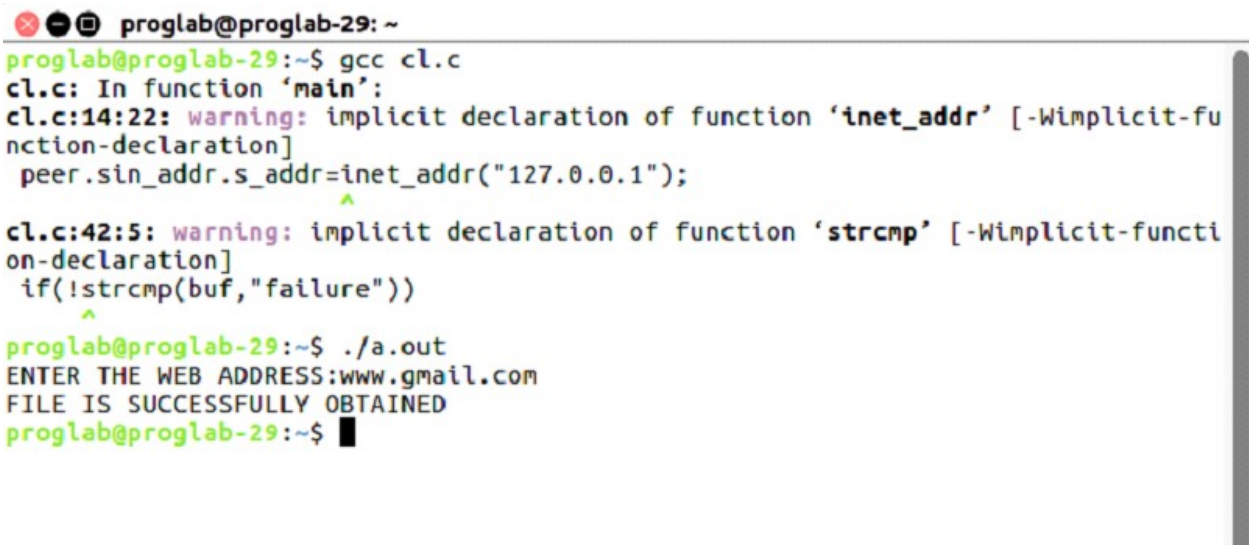
OUTPUT:-



SERVER:



CLIENT:



RESULT:-

Thus the socket creation for http has been executed successfully.

IMPLEMENTATION OF RPC

EX NO: 06

DATE:

AIM:

To write a c program to implement RPC.

ALGORITHM:

1. Start the program.
2. Import all necessary packages.
3. Create two application client and server.
4. Connect both applications.
5. Send the input data to client and server.
6. Cut the input in client.

DESCRIPTION:-

Module in server:-

1. Establish connection between client and server.
2. Enter the command in server.

Module in client:-

1. Establish connection to client and server.
2. After getting input in server display the output image.
3. Stop the program.

PROGRAM:-

SERVER:-

```
#include<netinet/in.h>
#include<string.h>
#include<sys/socket.h>
#include<stdio.h>
int main()
{
char buffer[10];
int i,sd,cd,size;
struct sockaddr_in serv;
```

```
struct sockaddr cli;
sd=socket(AF_INET,SOCK_STREAM,0);
serv.sin_addr.s_addr=INADDR_ANY;
serv.sin_port=htons(9059);
serv.sin_family=AF_INET;
bzero(&(serv.sin_zero),8);
if((connect(sd,(struct sockaddr*)&serv,sizeof(struct sockaddr)))>0)
   {
    printf("no connection to server...try after some time...");
    return 0;
   }
else                                                          6
  {
    printf("got command from server....");
    read(sd,buffer,10);
    puts(buffer);
   system(buffer);
   }
close(sd);
}
```

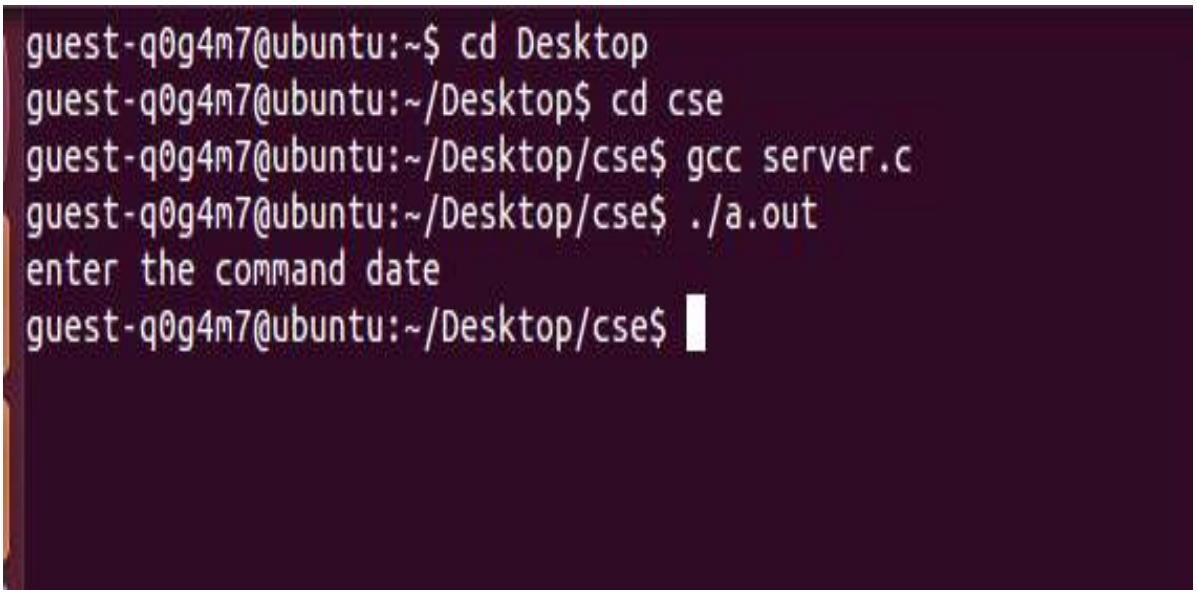
CLIENT:-

```
#include<netinet/in.h>
#include<string.h>
#include<sys/socket.h>
#include<stdio.h>
int main()
{
char a[10],b[10],buffer[10];
int i,sd,cd,size;
struct sockaddr_in serv;
struct sockaddr cli;
sd=socket(AF_INET,SOCK_STREAM,0);
serv.sin_addr.s_addr=INADDR_ANY;
serv.sin_port=htons(9059);
serv.sin_family=AF_INET;
bzero(&(serv.sin_zero),8);
bind(sd,(struct sockaddr*)&serv,sizeof(serv));
listen(sd,5);
size=sizeof(struct sockaddr_in);
cd=accept(sd,&cli,&size);
printf("enter the command");
fgets(buffer,10,stdin);
write(cd,buffer,strlen(buffer)+1);
```

```
close(sd);
close(cd);
}
```

OUTPUT:

SERVER:



CLIENT:

```
guest-q0g4m7@ubuntu:~$ cd Desktop
guest-q0g4m7@ubuntu:~/Desktop$ cd cse
guest-q0g4m7@ubuntu:~/Desktop/cse$ gcc client.c
guest-q0g4m7@ubuntu:~/Desktop/cse$ ./a.out
got command from server.... date

Thu Mar 12 01:43:47 PDT 2015
guest-q0g4m7@ubuntu:~/Desktop/cse$
```

RESULT:-

Thus the implementation of RPC has been executed.

APPLICATION USING TCP SOCKET FOR ECHO CLIENT AND ECHO SERVER.

EX NO: 07

DATE:

AIM:-

To write a c program application using TCP socket for echo client and eco server.

ALGORITHM:-

1. Start the program.
2. Import all necessary packages.
3. Create two new application client and server.
4. Connect both applications.

5. Send the input data into client through server.
6. Get the input in client.

DESCRIPTION:-

Module in server:-

1. Establish connection between client and server.
2. After getting an input from server display the particular data.
3. Stop the program.

PROGRAM:-

ECHOSERVER.C

```
#include<stdio.h>
#include<netinet/in.h>
#include<netdb.h>
#include<stdlib.h>
#include<unistd.h>
#define SERV_TCP_PORT 5035
int main(int argc,char**argv)
{
int sockfd,newsockfd,clength;
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("\n start");
bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
printf("\n listening");
printf("\n");
listen(sockfd,5);
clength=sizeof(cli_addr);
newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr,&clength);
printf("\n acxcepted");
printf("\n");
read(newsockfd,buffer,1096);
printf("\n client message:%s",buffer);
write(newsockfd,buffer,4096);
printf("\n");
close(sockfd);
return 0;
}
```

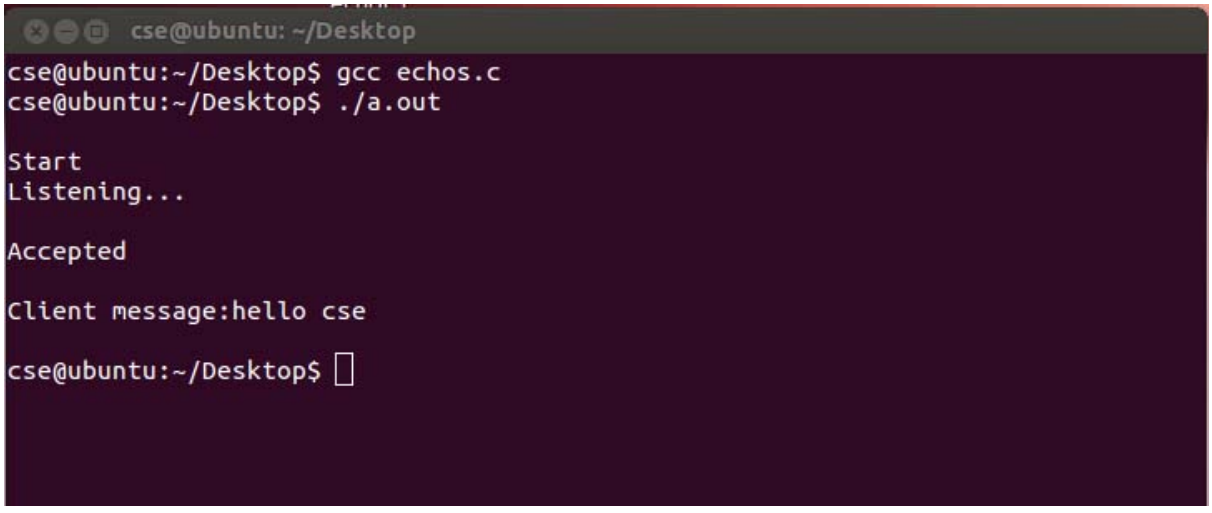
ECHOCLIENT.C

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<unistd.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");
```

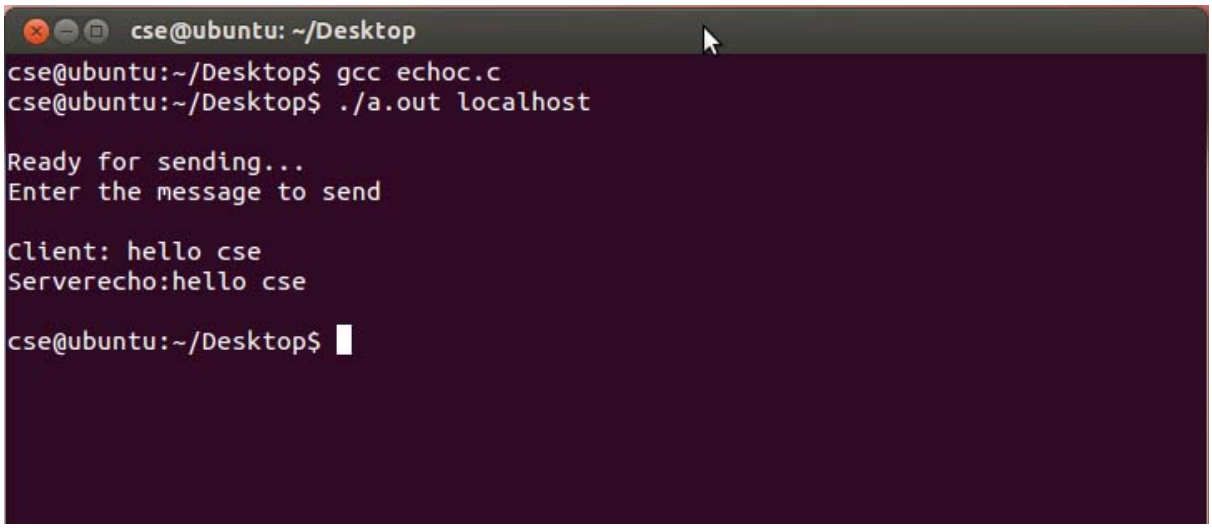
```
serv_addr.sin_port=htons(SERV_TCP_PORT);
printf("\n ready for sending..");
connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
printf("\n enter the message to send\n");
printf("\n client:");
fgets(buffer,4096,stdin);
write(sockfd,buffer,4096);
printf("serverecho:%s",buffer);
printf("\n");
close(sockfd);
return 0;
}
```

OUTPUT:-

ECHO SERVER:-



ECHOCLIENT:-



RESULT:-

Thus the application using TCP socket for echo client and echo server program was written and the output was verified successfully.

APPLICATION USING TCP SOCKET FOR SHARED CLIENT-SERVER CHART.

EX NO: 07 B

DATE:

AIM:-

To write a c program for client server application for chart.

ALGORITHM:-

1. Start the program.
2. Import all necessary packages.
3. Create two new application client and server.

4. Connect both applications.
5. Send the input data into client through server.
6. Get the input in client
7. Stop the program.

DESCRIPTION:-

Module in server:-

1. Establish connection between client and server.
2. Enter comment in server.

Module in client:-

1. Establish connection between client and server.
2. After giving the input in server display the output in it.
3. Stop the program.

PROGRAM:-

SERVER:-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include<sys/times.h>
#define SERV_IP 192.168.1.248
#define SERV_PORT 9059
void dg_ser(int sockfd);
int main(int argc,char *argv[])
{
int sockfd,connfd,bfd,listenfd;
```

```
struct sockaddr_in servaddr,cliaddr;
socklen_t clilen;
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
sockfd=socket(AF_INET,SOCK_STREAM,0);
printf("SOCKET DESCRIPTOR %d\n",sockfd);
bfd=bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
printf("BINDING %d\n",bfd);
listenfd=listen(sockfd,1);
printf("LISTENING %d\n",listenfd);
connfd=-1;
clilen=sizeof(cliaddr);
for(;;)
{
printf("BEFORE CONNECT %d\n",connfd);
connfd=accept(sockfd,(struct sockaddr*)&cliaddr,&clilen);
printf("AFTER CONNECT %d\n",connfd);
dg_ser(connfd);
exit(0);
}
}
void dg_ser(int sockfd)
{
int n,i;
socklen_t len;
char msg[500],msg1[500],msg2[]={'b','y','e','\0'};
for(;;)
{
n=recv(sockfd,msg,500,0);
printf("\n CLIENT:");
if((msg[0]=='b')&&(msg[1]=='y')&&(msg[2]=='e'))
{
printf("BYE\n");
printf("CLIENT TERMINATED\n");
return;
}
for(i=0;i<n;i++)
printf("%c",msg[i]);
```

```
printf("\n SERVER:");
fgets(msg1,500,stdin);
send(sockfd,msg1,strlen(msg1),0);
if((msg1[0]=='b')&&(msg1[1]=='y')&&(msg1[2]=='e'))
exit(0);
}
}
```

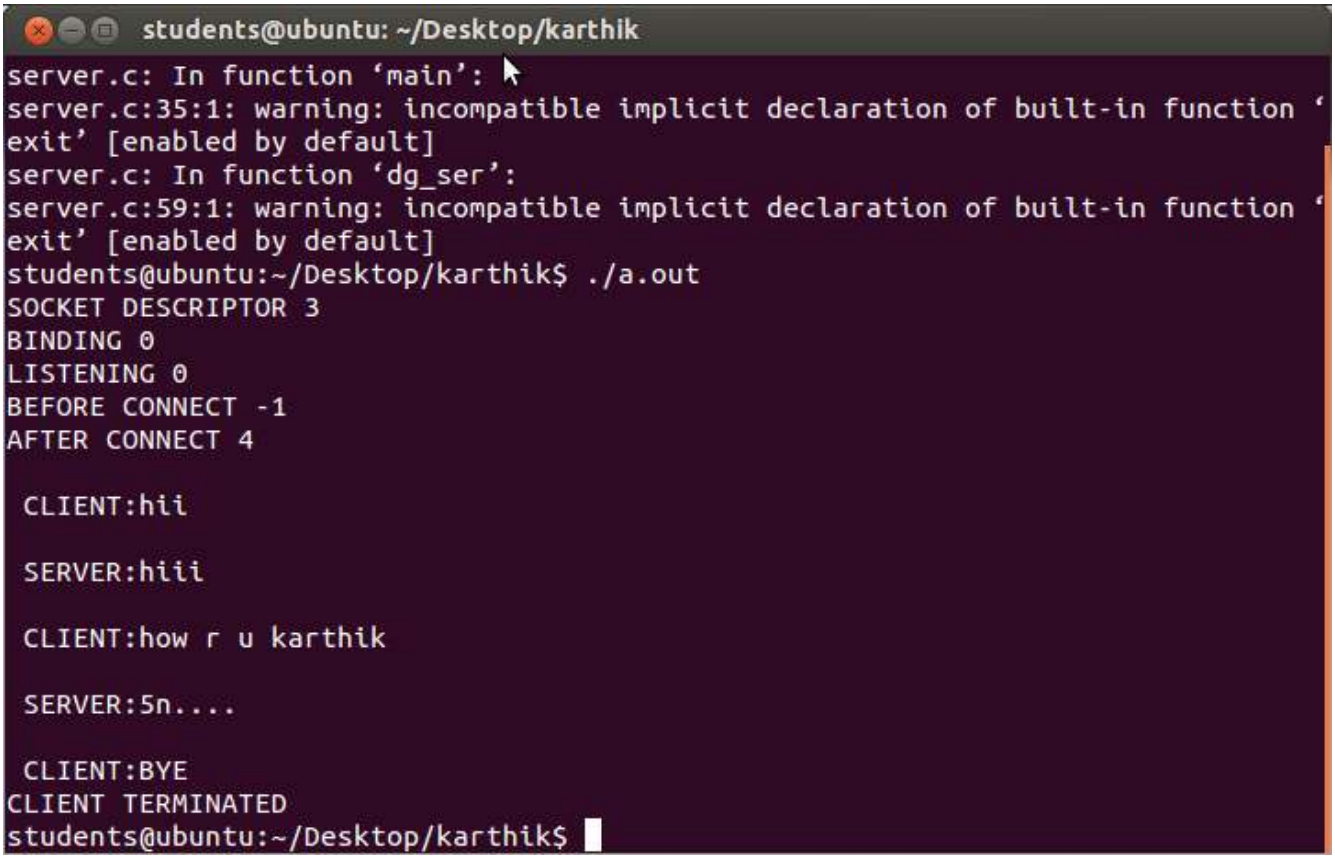
CLIENT:-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include<sys/times.h>
#define SERV_IP 192.168.1.248
#define SERV_PORT 9059
void dg_cli(FILE *fp,int sockfd);
int main(int argc,char *argv[])
{
int sockfd,connfd;
struct sockaddr_in servaddr;
```

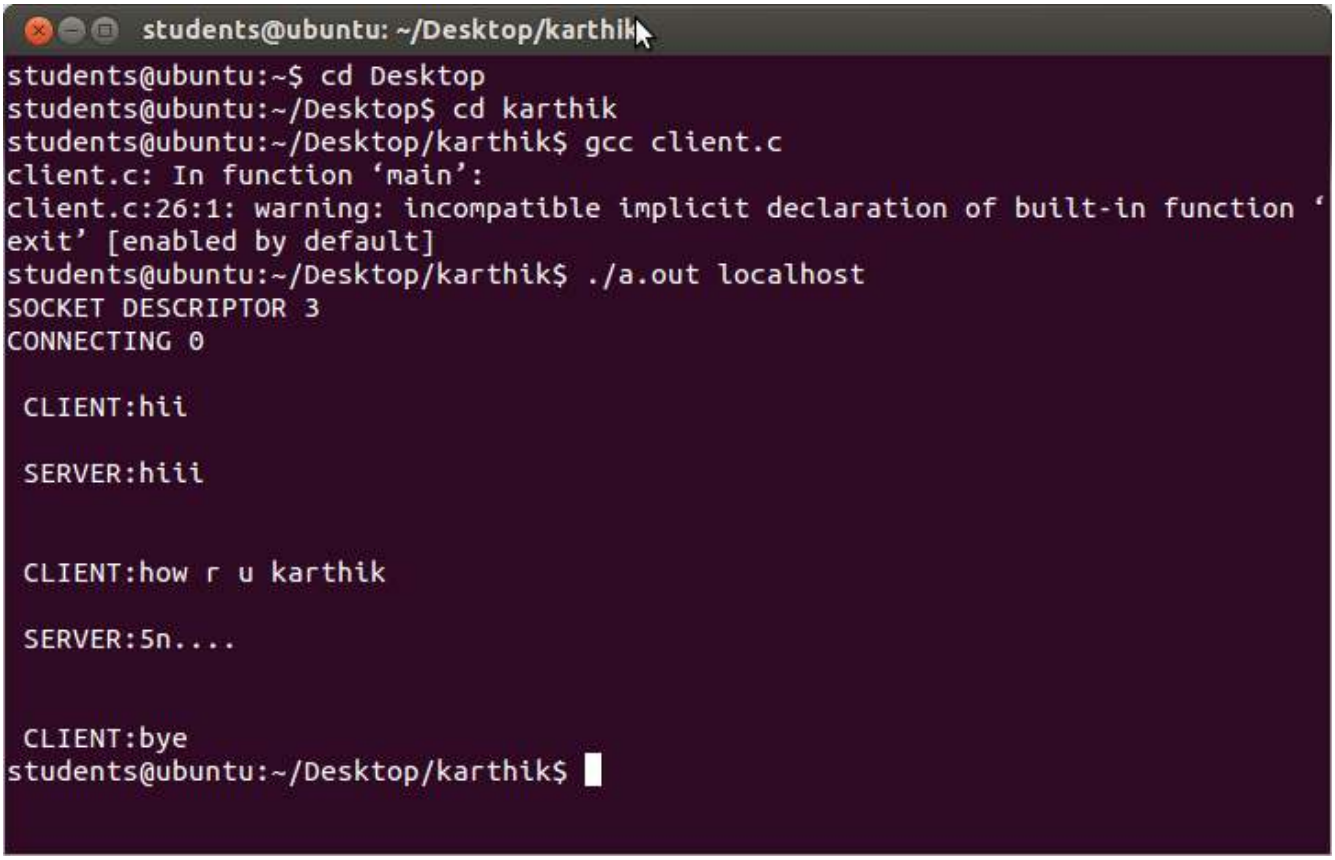
```
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
sockfd=socket(AF_INET,SOCK_STREAM,0);
printf("SOCKET DESCRIPTOR %d\n",sockfd);
connfd=connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
printf("CONNECTING %d\n",connfd);
dg_cli(stdin,sockfd);
close(sockfd);
exit(0);
}
void dg_cli(FILE *fp,int sockfd)
{
int n;
char sendline[500],recvline[500],msg2[]={'b','y','e','\0'};
for(;;)
{
printf("\n CLIENT:");
sendline[0]='\0';
fgets(sendline,500,stdin);
send(sockfd,sendline,strlen(sendline),0);
if((sendline[0]=='b')&&(sendline[1]=='y')&&(sendline[2]=='e'))
return;
recvline[0]='\0';
n=recv(sockfd,recvline,500,0);
recvline[n]=0;
if((recvline[0]=='b')&&(recvline[1]=='y')&&(recvline[2]=='e'))
return;
recvline[n]=0;
printf("\n SERVER:");
fputs(recvline,stdout);
printf("\n");
}
}
```

OUTPUT:-

SERVER:



CLIENT:



RESULT:-

Thus the application using TCP socket for shared client server application for chart program has been written and executed successfully.

APPLICATION TCP SOCKET LIKE FILE TRANSFER

EX NO: 07 C

DATE:

AIM:

To implement FTP by using C program.

ALGORITHM:-

1. Start the program.
2. Import all necessary packages to the program.
3. Create two application client and server.
4. Connect both applications.
5. Send the input data into client through server.
6. Get the input in client.

DESCRIPTION:-

Module in server:-

1. Establish connection between client and server.
2. Get the output acknowledgement from the client.

Module in client:-

1. Establish connection to client and server.
2. Get the file name.
3. Display the output.
4. Stop the program.

PROGRAM:-

INPUT FILE: (ftp.txt)

Welcome hi hello

SERVER:

```
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<error.h>
#include<string.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/wait.h>
#include<signal.h>
#include<sys/socket.h>
#define MYPORT 7014
#define BACKLOG 10
int main(void)
{
char buf[100],fname[30];
int n,nbytes;
int sockfd,new_fd,size,des;
struct sockaddr_in maddr;
struct sockaddr_in taddr;
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
perror("SOCKET");
exit(1);
}
maddr.sin_family=AF_INET;
maddr.sin_port=htons(MYPORT);
maddr.sin_addr.s_addr=INADDR_ANY;
memset(&(maddr.sin_zero),'\0',8);
if(bind(sockfd,(struct sockaddr*)&maddr,sizeof(struct sockaddr))==-1)
{
perror("BIND");
exit(1);
```

```
}
if(listen(sockfd,BACKLOG)==-1)
{
perror("LISTEN");
exit(1);
}
printf("FILE TRANSFER\n");
while(1)
{
size=sizeof(struct sockaddr_in);
if((new_fd=accept(sockfd,(struct sockaddr*)&taddr,&size))==-1)
{
perror("ACCEPT");
continue;
}
if((nbytes=recv(new_fd,fname,114,0))==-1)
{
perror("ERROR IN RECEIVING\n");
exit(1);
}
if((des=open(fname,0))==-1)
{
perror("ERROR");
exit(0);
}
while((n=read(des,buf,100))>0)
{
if(send(new_fd,buf,n,0)==-1)
perror("ERROR IN SENDING\n");
}
printf("FILE IS READ AND SENT SUCCESSFULLY\n");
fflush(stdout);
close(des);   }
return 0;
}
```

CLIENT:

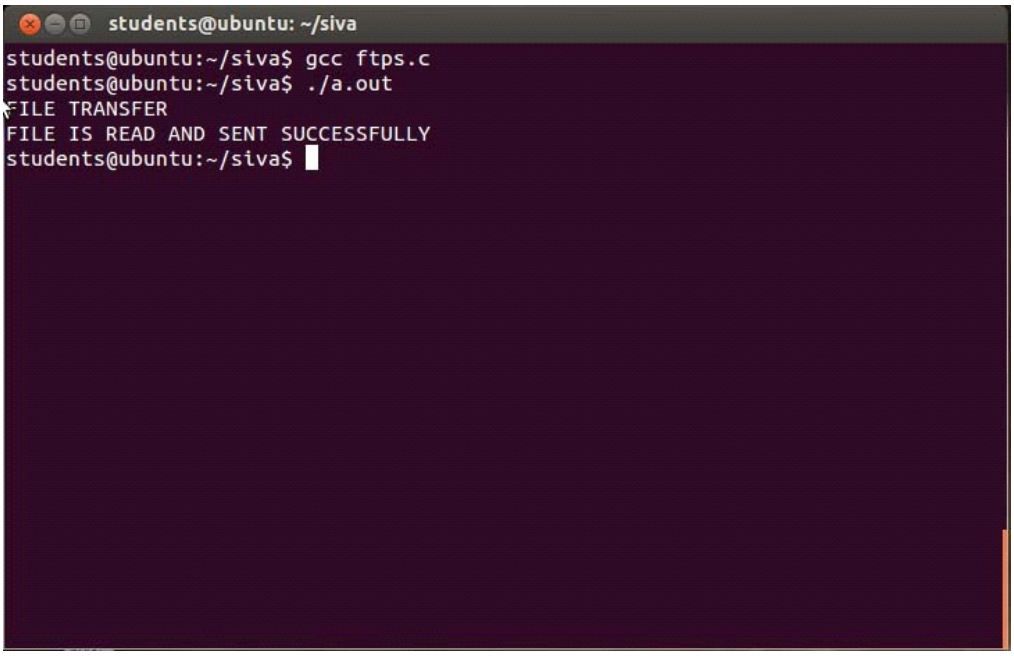
```
#include<stdio.h>
#include<stdlib.h>
```

```
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/wait.h>
#include<signal.h>
#define MYPORT 7014
int main(int argc,char *argv[])
{
char buf[100],fname[14];
int nbytes,sockfd;
struct sockaddr_in taddr;
if(argc!=2)
{
fprintf(stderr,"usage:ClientHost Name\n");
exit(1);
}
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
perror("SOCKET");
exit(1);
}
taddr.sin_family=AF_INET;
taddr.sin_port=htons(MYPORT);
taddr.sin_addr.s_addr=htonl(INADDR_ANY);
memset(&(taddr.sin_zero),'\0',8);
if(connect(sockfd,(struct sockaddr *)&taddr,sizeof(struct sockaddr))==-1)
{
perror("CONNECTING ERROR");
exit(1);
}
fflush(stdout);
printf("FILE TRANSFER\n");
fflush(stdout);
printf("INPUT\n");
printf("ENTER THE FILE NAME\n");
scanf("%s",fname);
if(send(sockfd,fname,14,0)==-1)
```

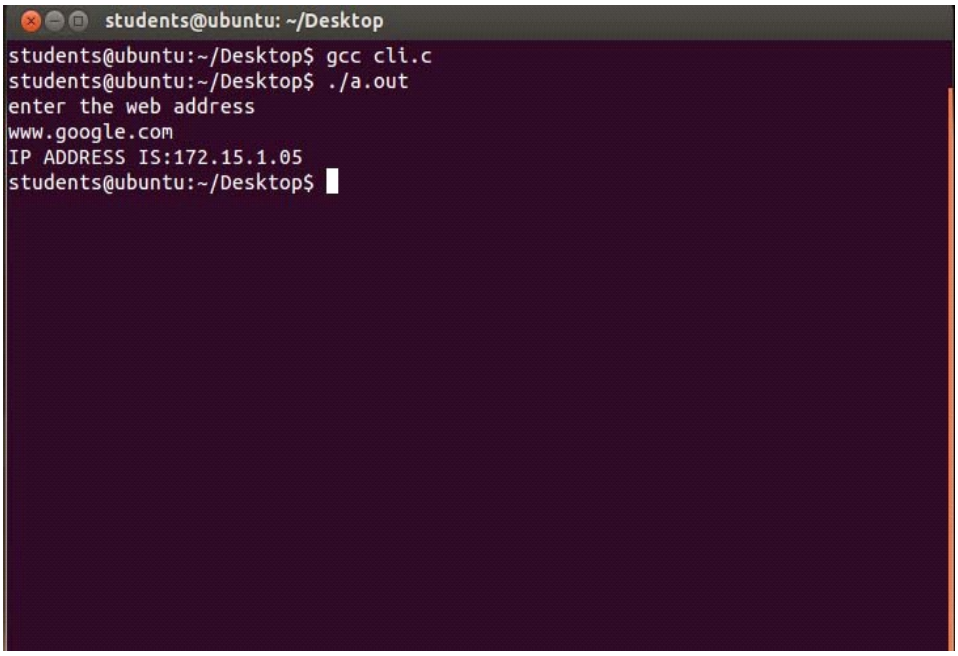
```
perror("SENDING ERROR");
printf("OUTPUT\n");
while(1)
{
if((nbytes=recv(sockfd,buf,100,0))!=0)
{
buf[nbytes]='\0';
printf("RECEIVED FROM CLIENT\n");
printf("THE FILE CONTENTS ARE %s",buf);
fflush(stdout);
}
else
break;
}
close(sockfd);
return 0;
}
```

OUTPUT:

FTP SERVER:



FTP CLIENT:



RESULT:-

Thus the implementation of ftp was written and the output was verified successfully.

APPLICATION USING TCP AND UDP SOCKET LINE DNS

EX NO: 8 A

DATE:

AIM:-

To write a C program for domain name server.

ALGORITHM:-

1. Start the program.
2. Import all necessary packages to the program.
3. Create two application client and server.
4. Connect both applications.
5. Send the input data into client through server.
6. Get the input in client.

DESCRIPTION:-

Module in server:-

1. Establish connection between client and server.
2. Enter web address in server.

Module in client:-

1. Establish connection to client and server.
2. After getting an input from server display the particular web address is given by server.
3. Stop the program.

PROGRAM:-

INPUT FILE (dns.doc)

www.yahoomail.com 172.15.1.04
www.google.com 172.15.1.05
www.rediffmail.com 172.15.1.06
www.gmail.com 172.15.1.08

SERVER:

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
struct sockaddr_in local;
int s,s1,rc,i=0;
FILE *fin,*fout;
char buf[100],chec[100];
local.sin_family=AF_INET;
local.sin_port=htons(9500);
local.sin_addr.s_addr=inet_addr("127.0.0.1");
s=socket(AF_INET,SOCK_STREAM,0);
if(s<0)
{
printf("SOCKET CALL FAILED\n");
exit(1);
}
rc=bind(s,(struct sockaddr*)&local,sizeof(local));
if(rc<0)
{
printf("BIND CALL FAILED\n");
exit(1);
}
rc=listen(s,5);
if(rc)
{
printf("LISTEN CALL FAILED\n");
exit(1);
}
s1=accept(s,NULL,NULL);
```

```
if(s1<0)
{
printf("ACCEPT CALL FAILED\n");
exit(1);
}
rc=recv(s1,buf,100,0);
if(rc<=0)
{
printf("RECEIVE CALL FAILED\n");
exit(1);
}
fin=fopen("dns.doc","r");
while(!feof(fin))
{
fscanf(fin,"%s",chec);
if(!(strcmp(buf,chec)))
{
break;
}
}
if(!feof(fin))
{
fscanf(fin,"%s",chec);
rc=send(s1,chec,100,0);
fclose(fin);
}
else
{
fclose(fin);
rc=send(s1,"failure",100,0);
}
if(rc<=0)
printf("SEND CALL FAILED\n");
exit(0);
}
```

CLIENT:

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

```
#include<netinet/in.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
struct sockaddr_in peer;
int s,rc,i=0;
char buf[100],sc[100];
peer.sin_family=AF_INET;
peer.sin_port=htons(9500);
peer.sin_addr.s_addr=inet_addr("127.0.0.1");
s=socket(AF_INET,SOCK_STREAM,0);
if(s<0)
{
printf("SOCKET CALL FAILED\n");
exit(1);
}
rc=connect(s,(struct sockaddr*)&peer,sizeof(peer));
if(rc)
{
printf("CONNECT CALL FAILED\n");
exit(1);
}
printf("ENTER THE WEB ADDRESS\n");
scanf("%s",sc);
rc=send(s,sc,100,0);
if(rc<=0)
{
printf("SEND CALL FAILED\n");
exit(1);
}
rc=recv(s,buf,100,0);
if(rc<=0)
{
printf("RECEIVE CALL FAILED\n");
}
if(!strcmp(buf,"failure"))
{
printf("INVALID WEB\n");
}
```

```
printf("IP ADDRESS IS: %s\n",buf);
exit(0);
}
```

OUTPUT:

DNS SERVER:

```
proglab19@proglab19-OptiPlex-9010:~$ cd Desktop
proglab19@proglab19-OptiPlex-9010:~/Desktop$ gcc ser7c.c
ser7c.c: In function 'main':
ser7c.c:54:9: warning: implicit declaration of function 'open' [-Wimplicit-function-declaration]
 if((des=open(fname,0))==-1)
         ^
proglab19@proglab19-OptiPlex-9010:~/Desktop$ ./a.out
FILE TRANSFER
FILE IS READ AND SENT SUCCESSFULLY
```

DNS CLIENT:

```
proglab19@proglab19-OptiPlex-9010:~$ cd Desktop
proglab19@proglab19-OptiPlex-9010:~/Desktop$ gcc cli7c.c
proglab19@proglab19-OptiPlex-9010:~/Desktop$ ./a.out localhost
FILE TRANSFER
INPUT
ENTER THE FILE NAME
ftp.txt
OUTPUT
RECEIVED FROM CLIENT
THE FILE CONTENTS ARE Welcome hi hello
```

RESULT:-

Thus the c program for domain name server was successfully implemented.

UDP CLIENT SERVER TO TRANSFER A FILE

EX NO: 8 B

DATE:

AIM:

To write a c program for UDP client and server to transfer a file.

ALGORITHM:-

1. Start the program.
2. Import all necessary packages to the program.
3. Create two applications, client and server.
4. Connect both applications.
5. Send the input data into client through server.
6. Get the input in client.

DESCRIPTION:-

Module in server:-

1. Establish connection between client and server.
2. Enter the data in client.

Module in client:-

1. Establish the connection between client and server.
2. After getting an input from server display the particular data given by the server.
3. Stop the program.

PROGRAM:

Hello.txt:

Hi

Hello

How r u

CLIENT:

```
#include<stdio.h>
 #include<string.h>
 #include<stdlib.h>
#include<sys/types.h>
 #include<sys/socket.h>
 #include<netinet/in.h>
 #include<unistd.h>
 #define SERV_PORT 6349
main(int argc,char **argv)
 {
char filename[80];
 int sockfd;
struct sockaddr_in servaddr;
sockfd=socket(AF_INET,SOCK_DGRAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
```

servaddr.sin_port=htons(SERV_PORT); 8B

inet_pton(AF_INET,argv[1],&servaddr.sin_addr);

printf("enter the file name");

scanf("%s",filename); sendto(sockfd,filename,strlen(filename),0,
(structsockaddr*)&servaddr,sizeof(servaddr))

}

SERVER:

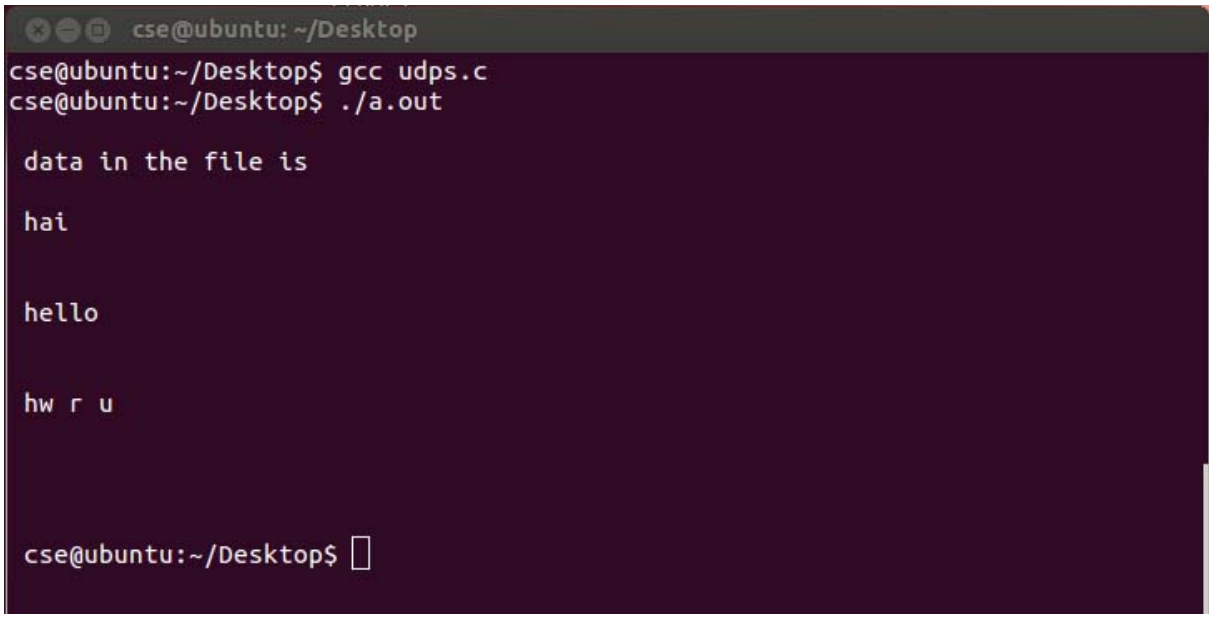
SERVER:

```
#include<stdio.h>
#include<string.h>
 #include<stdlib.h>
 #include<sys/socket.h>
#include<sys/types.h>
 #include<netinet/in.h>
#define SERV_PORT 6349
main(int argc,char **argv)
 {
char filename[80],recvline[80];
 FILE *fp;
struct sockaddr_in servaddr,cliaddr;
int clilen,sockfd;
```

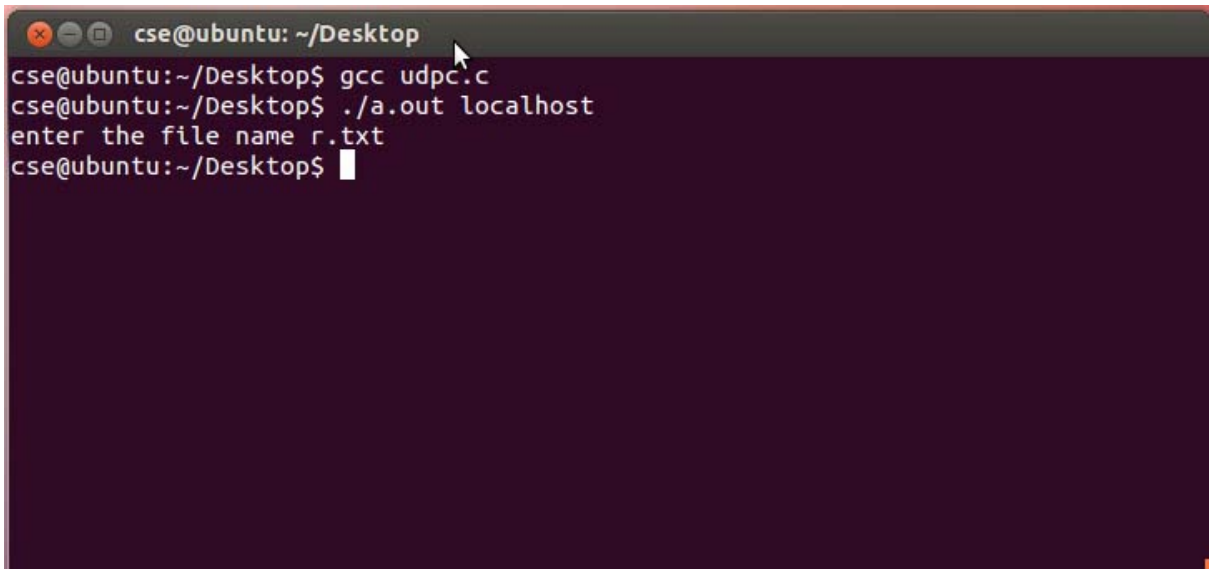
```
sockfd=socket(AF_INET,SOCK_DGRAM,0);
bzero(&servaddr,sizeof(servaddr));

servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(SERV_PORT);
 bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
clilen=sizeof(cliaddr);
recvfrom(sockfd,filename,80,0,(struct sockaddr*)&cliaddr,&clilen);
printf("\n data in the file is \n ");
fp=fopen(filename,"r");
while(fgets(recvline,80,fp)!=NULL)
{
 printf("\n %s\n ",recvline);
 }
fclose(fp);
 }
```

OUTPUT:

UDP SERVER:



UDP CLIENT:



RESULT:-

Thus the c program for UDP client server to transfer a file was executed successfully.

STUDY OF NETWORK SIMULATOR

EX NO: 09

DATE:

AIM:

To study about network simulator (NS).

NETWORK SIMULATORS (NS):

A **network simulator** is software that predicts the behavior of a computer network. Since communication Networks have become too complex for traditional analytical methods to provide an accurate understanding of system behavior, network simulators are used. In simulators, the computer network is modeled with devices, links, applications etc. and the performance is analyzed. Simulators come with support for the most popular technologies and networks in use today such as Wireless LANs, Mobile ADHOC Networks, Wireless Sensor Networks, Vehicular ADHOC Networks, Cognitive Radio networks, LTE / LTE-Advanced Networks, Internet of things (IOT) etc.

SIMULATIONS:

Most of the commercial simulators are GUI driven, while some network simulators are CLI driven. The network model / configuration describe the network (nodes, routers, switches, links) and the events (data transmissions, packet error etc.). Output results would include network level metrics, link metrics, device metrics etc. Further, drill down in terms of simulations trace files would also be available. Trace files log every packet, every event that occurred in the simulation and are used for analysis. Most network simulators use discrete event simulation, in which a list of pending "events" is stored, and those events are processed in order, with some events triggering future events—such as the event of the arrival of a packet at one node triggering the event of the arrival of that packet at a downstream node.

NETWORK EMULATION:

Network emulation allows users to introduce real devices and applications into a test network (simulated) that alters packet flow in such a way as

to mimic the behavior of a live network. Live traffic can pass through the simulator and be affected by objects within the simulation.

The typical methodology is that real packets from a live application are sent to the emulation server (where the virtual network is simulated). The real packet gets 'modulated' into a simulation packet. The Simulation packet gets demodulated into real packet after experiencing effects of loss, errors, delay, jitter etc., thereby transferring these network effects into the real packet. Thus it is as-if the real packet flowed through a real network but in reality it flowed through the simulated network.

Emulation is widely used in the design stage for validating communication networks prior to deployment.

EXAMPLES OF NETWORK SIMULATORS:

There are both free/open-source and proprietary network simulators available. Examples of notable network simulators / emulators include:

- ns (open source)
- OPNET (proprietary software)
- NetSim (proprietary software)

Both commercial tools are available at deeply discounted prices to universities

USES OF SIMULATORS/EMULATORS:

Network simulators provide a cost effective method for

a. Network design validation for enterprises / data centers /sensor networks etc.
b. Analyzing Utilities distribution communication, railway signaling / communication etc.
c. Network R & D (More than 70 % of all Network Research paper reference a network simulator)
d. Defense applications such as HF / UHF / VHF MANET networks, Tactical data links etc.
e. Education - Lab experimentation. Most universities use a network simulation to teach / experiment on networking since it's too expensive to buy hardware equipment covering all the various technologies.

There are a wide variety of network simulators, ranging from the very simple to the very complex. Minimally, a network simulator must enable a user to

1. Model the network topology specifying the nodes on the network and the links between those nodes
2. Model the application flow (traffic) between the nodes
3. Providing network performance metrics as output
4. Visualization of the packet flow
5. Technology / protocol evaluation and device designs
6. Logging of packet / events for drill down analyses / debugging

Basic features:

a). A simulator has to allow defining the characteristics of the system that one aims to model. The simulator must serve the general-purpose, enough for being able to provide the default behaviors of the system e.g a communication network simulator must have features to model a few transmission media (i.e. propagation media like air, fiber-optics etc).

b) The simulator must be flexible enough to allow the user to define objectives by utilizing what the simulator offers; e.g. often, in communications network research, users create newer protocols, or approaches to a particular challenge. They expect the simulator to provide "pluggable" features which shall enable them to quickly execute the components they have designed using the other available features. Features may be allowed in a particular format by a simulator. This format is important in the selection of a simulator. Input-output from the real system: The next important feature is the format of input and output data from the system. Input data are necessary for setting the main parameters of the model e.g. transmission characteristics, inter-arrival rates of traffic, channel loss etc. Output data are necessary to validate functionality, and or calculate the importance of the obtained results. Equally important is the format of data used to specify input or collect output, since creating input data files or rendering and analyzing output results should not become a task of its own. Design of the experiment and simulation runs: This involves designing the whole experiment. It includes various aspects such as details about each simulation, preferences for data to be collected, simulation runs, time duration of each simulation run, etc. It may also involve finding out what support the simulator has for synchronizing clocks, random number generators and such optional details.

Advantages and Disadvantages:

Main advantages of simulation include:

- Study the behavior of a system without building it.
- Results are accurate in general, compared to analytical model.
- Help to find un-expected phenomenon, behavior of the system.
- Easy to perform ``What-If'' analysis.

Main disadvantages of simulation include:

- Expensive to build a simulation model.
- Expensive to conduct simulation.
- Sometimes it is difficult to interpret the simulation results.

RESULT:-

Thus the case study about the different routing algorithms to select the network path with its optimum and economical during data transfer was completed.

CASE STUDY ABOUT DIFFERENT ROUTING ALGORITHMS

EX NO: 10 A

DATE:

AIM:

To study the concept of state routing, flooding and distance vector.

ROUTING:

- Routing is the process of selecting paths in a network along which to send network traffic.
- Goals of routing are correctness, simplicity, Robustness, Stability, Fairness and Optimality.
- Routing is performed for many kinds of network, including the telephone network, electronic data networks and transportation networks.
- Routing Algorithms can be classified based on the following:
    - Static or Dynamic Routing,
    - Distributed or Centralized,
    - Single path or Multi path,
    - Flat or Hierarchical,
    - Intra Domain or Inter Domain,
    - Link State or Distance Vector.
- Algorithms may be static; the routing decisions are made ahead of time, with information about the network topology and capacity, and then loaded into the routers.
- Algorithms may be dynamic, where the routers make decisions based on information they gather, and the routes change over time, adaptively.
- Routing can be grouped into two categories: Non-adaptive routing, and Adaptive routing.

NON-ADAPTIVE ROUTING

- Once the pathway to destination has been selected, the router sends all packets for that destination along that one route.
- The routing decisions are not made based on the condition or topology of the network.
- Examples: Centralized, Isolated, and Distributed Algorithms

ADAPTIVE ROUTING

- A router may select a new route for each packet (even packets belonging to the same transmission) in response to changes in condition and topology of the networks.
- Examples: Flooding, and Random Walk.

ROUTING ALGORITHMS

Shortest Path Routing:

- Links between routers have a cost associated with them. In general it could be a function of distance, bandwidth, average traffic, communication cost, mean queue length, measured delay, router processing speed, etc.
- The shortest path algorithm just finds the least expensive path through the network, based on the cost function.
- Examples: Dijkstra's algorithm

Distance Vector Routing:

- In this routing scheme, each router periodically shares its knowledge about the entire network with its neighbours.
- Each router has a table with information about network. These tables are updated by exchanging information with the immediate neighbours.
- It is also known as **Belman-Ford** or Ford-Fulkerson Algorithm.
- It is used in the original ARPANET, and in the Internet as RIP.
- Neighboring nodes in the subnet exchange their tables periodically to update each other on the state of the subnet (which makes this a dynamic algorithm). If a neighbor claims to have a path to a node which is shorter than your path, you start using that neighbor as the route to that node.
- Distance vector protocols (a vector contains both distance and direction), such as RIP, determine the path to remote networks using

hop count as the metric. A hop count is defined as the number of times a packet needs to pass through a router to reach a remote destination.

- For IP RIP, the maximum hop is 15. A hop count of 16 indicates an unreachable network. Two versions of RIP exist: version 1 and version 2.
- IGRP is another example of a distance vector protocol with a higher hop count of 255 hops.
- Periodic updates are sent at a set interval. For IP RIP, this interval is 30 seconds.
- Updates are sent to the broadcast address 255.255.255.255. Only devices running routing algorithms listen to these updates.
- When an update is sent, the entire routing table is sent.

Link State Routing:

- The following sequence of steps can be executed in the Link State Routing.
- The basis of this advertising is a short packed called a Link State Packet (LSP).
- OSPF (Open shortest path first) and IS-IS are examples of Link state routing.
- Link State Packet(LSP) contains the following information:
    1. The ID of the node that created the LSP;
    2. A list of directly connected neighbors of that node, with the cost of the link to each one;
    3. A sequence number;
    4. A time to live(TTL) for this packet.
- When a router floods the network with information about its neighbourhood, it is said to be advertising.
    1. Discover your neighbors
    2. Measure delay to your neighbors
    3. Bundle all the information about your neighbors together
    4. Send this information to all other routers in the subnet
    5. Compute the shortest path to every router with the information you receive
    6. Each router finds out its own shortest paths to the other routers by using **Dijkstra's algorithm**.
- In link state routing, each router shares its knowledge of its neighborhood with all routers in the network.

- Link-state protocols implement an algorithm called the shortest path first (SPF, also known as Dijkstra's Algorithm) to determine the path to a remote destination.
- There is no hop count limit. (For an IP datagram, the maximum time to live ensures that loops are avoided.)
- Only when changes occur, it sends all summary information every 30 minutes by default. Only devices running routing algorithms listen to these updates. Updates are sent to a multicast address.
- Updates are faster and convergence times are reduced. Higher CPU and memory requirements to maintain link-state databases.
- Link-state protocols maintain three separate tables:

    1. **Neighbor table:** It contains a list of all neighbors, and the interface each neighbor is connected off of. Neighbors are formed by sending Hello packets.
    2. **Topology table (Link- State table)** : It contains a map of all links within an area, including each link's status.
    3. **Routing table** : It contains the best routes to each particular destination

Flooding Algorithm:

- It is a non-adaptive algorithm or static algorithm.
- When a router receives a packet, it sends a copy of the packet out on each line (except the one on which it arrived).
- To prevent form looping forever, each router decrements a hop count contained in the packet header.
- As soon as the hop count decrements to zero, the router discards the packet.

Flow Based Routing Algorithm:

- It is a non-adaptive routing algorithm.
- It takes into account both the topology and the load in this routing algorithm;
- We can estimate the flow between all pairs of routers.
- From the known average amount of traffic and the average length of a packet you can compute the mean packet delays using queuing theory.
- Flow-based routing then seeks to find a routing table to minimize the average packet delay through the subnet.

- Given the line capacity and the flow, we can determine the delay. It needs to use the formula for delay time T.

$$T = \frac{1}{\mu c - \lambda}$$

- Where, μ = Mean number of arrivals in packet/sec, 1/μ = the mean packet size in the bits, and c = Line capacity (bits/s).

The Optimality Principal:

This simple states that if router J is on the optimal path form router I to router k, then the optimal path from J to K also falls along this same path.

RESULT:

Thus the case study for different routing algorithms to select the networks path with its optimum and economical during data transfer has been studied.

DISTANCE VECTOR ROUTING

EX NO: 10 B

DATE:

AIM:

To write a c program to find distance vector routing.

ALGORITHM:

Step1: start the program.

Step2: declare the size.

Step3: get the number of nodes

Step4: display the adjacency matrix.

Step5: get the adjacency matrix table is inserted.

Step6: display the state value of router under1, under2, under3.

Step7: stop the program.

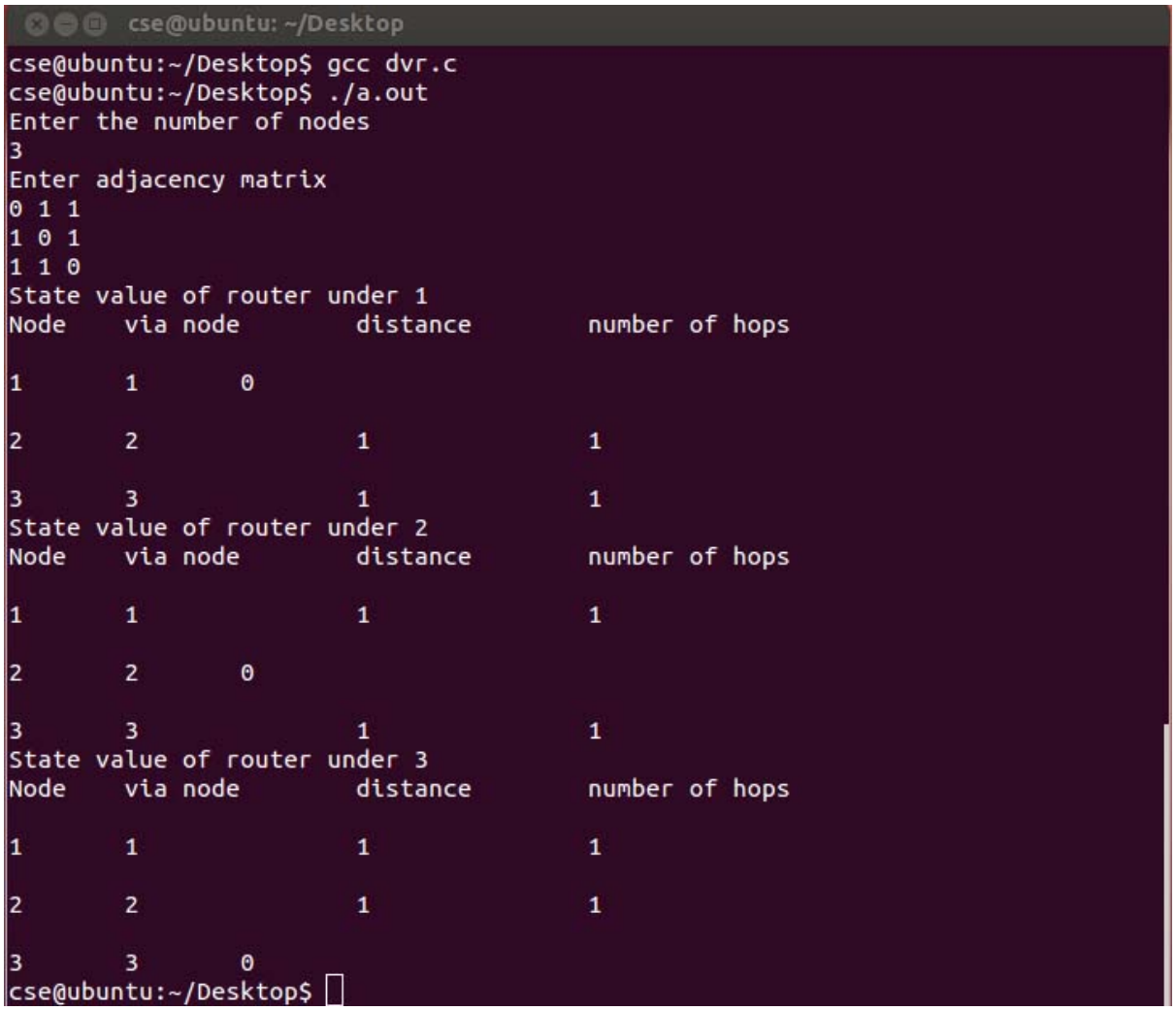
PROGRAM:

dvr.c

```
#include<stdio.h>
#define MAX 10
struct dist_vect
{
int dist[MAX];
int from[MAX];
};
int main()
{
int adj[MAX][MAX],n,i,j,hop[10][10]={{0}},k,count;
struct dist_vect arr[10];
printf("enter the number of nodes\n");
scanf("%d",&n);
printf("enter adjacency matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++);
scanf("%d",&adj[i][j]);
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
arr[i].dist[j]=adj[i][j];
arr[i].from[j]=j;
}
}
count=0;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
for(k=0;k<n;k++)
{
```

```
if(arr[i].dist[j]>adj[i][k]+arr[k].dist[j])
{
arr[i].dist[j]=adj[i][k]+arr[k].dist[j];
arr[i].from[j]=k;
count++;
if(count==0)
hop[i][j]=1;
else
hop[i][j]=count+hop[k][j];
}
}
count=0;
}
}
for(i=0;i<n;i++)
{
printf("state value of router under %d",i+1);
printf("\n node\tvia node\tdiatance\tnumber of hopes\n");
for(j=0;j<n;j++)
{
if(i==j)
printf("\n%d\t%d\t%d\n",j+1,arr[i].from[j]+1,arr[i].dist[j]);
else
printf("\n%d\t%d\t\t%d\t\t%d\n",j+1,arr[i].from[j]+1,arr[i].dist[j],hop[i][j]+1);
}
}
}
```

OUTPUT:



RESULT:

Thus the distance vector routing using c language was implemented and the output was verified successfully.