Experment-1

Implement the data link layer framing methods such as character stuffing.(byte Stuffing)

Introduction to character stuffing method of framing for Data Link Layer: The framing method gets around the problem of resynchronization after an error by having each frame start with the ASCII character sequence DLE STX and the sequence DLE ETX. If the destination ever losses the track of the frame boundaries all it has to do is look for DLE STX or DLE ETX characters to figure out. The data link layer on the receiving end removes the DLE before the data are given to the network layer. This technique is called character stuffing.

Program Algorithm:

```
Begin
Step 1: Initialize I and j as 0
Step 2: Declare n and pos as integer and a[20],b[50],ch as character
Step 3: read the string a
Step 4: find the length of the string n, i.e n-strlen(a)
Step 5: read the position, pos
Step 6: if pos > n then
Step 7: print invalid position and read again the position, pos
Step 8: endif
Step 9: read the character, ch
Step 10: Initialize the array b, b[0…5] as ’d’, ’l’, ’e’, ’s’, ’t’,’x’ respectively
Step 11: j=6;
Step 12: Repeat step[(13to22) until i<n
Step 13: if i==pos-1 then
Step 14: initialize b array,b[j],b[j+1]…b[j+6] as‘d’, ‘l’, ‘e’ ,’ch, ’d’, ‘l’,‘e’ respectively
Step 15: increment j by 7, i.e j=j+7
Step 16: endif
Step 17: if a[i]==’d’ and a[i+1]==’l’ and a[i+2]==’e’ then
Step 18: initialize array b, b[13…15]=’d’, ‘l’, ‘e’ respectively
Step 19: increment j by 3, i.e j=j+3
Step 20: endif
Step 21: b[j]=a[i]
Step 22: increment I and j;
Step 23: initialize b array,b[j],b[j+1]…b[j+6] as‘d’, ‘l’,‘e’,’e’,‘t’, ‘x’,‘\0’ respectively
Step 24: print frame after stuffing
Step 25: print b
End
```

Program Code: //Program for Character Stuffing

```
#include<stdio.h>
#include<string.h>
#include<process.h>
#include<conio.h>
void main()
{
int i=0,j=0,n,pos;
char a[20],b[50],ch;
```

```
printf("Enter string\n");
scanf("%s",&a);
n=strlen(a);
printf("Enter position\n");
scanf("%d",&pos);
if(pos>n)
{
printf("invalid position, Enter again :");
scanf("%d",&pos);}
printf("Enter the character\n");
ch=getche();
b[0]='d';
b[1]='l';
b[2]='e';
b[3]='s';
b[4]='t';
b[5]='x';
j=6;
while(i<n)
{
if(i==pos-1)
{
b[j]='d';
b[j+1]='l';
b[j+2]='e';
b[j+3]=ch;
b[j+4]='d';
b[j+5]='l';
b[j+6]='e';
j=j+7;
}
if(a[i]=='d' && a[i+1]=='l' && a[i+2]=='e')
{
b[j]='d';
b[j+1]='l';
b[j+2]='e';
j=j+3;
}
b[j]=a[i];
i++;
j++;
}
b[j]='d';
b[j+1]='l';
b[j+2]='e';
b[j+3]='e';
b[j+4]='t';
b[j+5]='x';
```

```
b[j+6]='\0';
printf("\nframe after stuffing:\n");
printf("%s",b);
}
```

Program Output:

```
Enter string
KITSCAI
Enter position
2
Enter the character
frame after stuffing:
dlestxMdldleLRITMdleetx
------------------
(program exited with code: 0) Press return to continue
```

Implement the data link layer framing methods such as and bit stuffing.

Introduction to bit stuffing framing method used in Data Link layer:
The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with the special bit pattern, 01111110, called a flag byte. Whenever the sender's data link layer encounters five consecutive ones in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to character stuffing, in which a DLE is stuffed into the outgoing character stream before DLE in the data.

Program Algorithm:

```
Begin
Step 1: Read frame length n
Step 2: Repeat step (3 to 4) until i<n(: Read values into the input frame (0’s and
1’s) i.e.
Step 3: initialize I i=0;
Step 4: read a[i] and increment i
Step 5: Initialize i=0, j=0,count =0
Step 6: repeat step (7 to 22) until i<n
Step 7: If a[i] == 1 then
Step 8: b[j] = a[i]
Step 9: Repeat step (10 to 18) until (a[k] =1 and k<n and count <5)
Step 10: Initialize k=i+1;
Step 11: Increment j and b[j]= a[k];
Step 12: Increment count ;
Step 13: if count =5 then
Step 14: increment j,
Step 15: b[j] =0
Step 16: end if
Step 17: i=k;
Step 18: increment k
Step 19: else
Step 20: b[j] = a[i]
Step 21: end if
Step 22: increment I and j
Step 23: print the frame after bit stuffing
Step 24: repeat step (25 to 26) until i< j
Step 25: print b[i]
Step 26: increment i
End
```

Program Code: // BIT Stuffing program

```
#include<stdio.h>
#include<string.h>
void main()
{
int a[20],b[30],i,j,k,count,n;
printf("Enter frame length:");
scanf("%d",&n);
printf("Enter input frame (0's & 1's only):");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
i=0; count=1; j=0;
while(i<n)
{
if(a[i]==1)
{
b[j]=a[i];
for(k=i+1;a[k]==1 && k<n && count<5;k++)
{
j++;
b[j]=a[k];
count++;
if(count==5)
{
j++;
b[j]=0;
}
i=k;
}}
else
{
b[j]=a[i];
}
i++;
j++;
}
printf("After stuffing the frame is:");
for(i=0;i<j;i++)
printf("%d",b[i]);
}
```

Program Output:

```
Enter frame length:5
Enter input frame (0's & 1's only):
1
1
1
1
1
After stuffing the frame is:111110
------------------
(program exited with code: 6)
Press return to continue
```

Experment-2

Write a C program to develop a DNS client server to resolve the given hostname.

ALGORITHM:

1. Create a new file. Enter the domain name and address in that file.
2. To establish the connection between client and server.
3. Compile and execute the program.
4. Enter the domain name as input.
5. The IP address corresponding to the domain name is display on the screen
6. Enter the IP address on the screen.
7. The domain name corresponding to the IP address is display on the screen.
8. Stop the program.

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<netdb.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
int main(int argc,char *argv[1])
{
struct hostent *hen;if(argc!=2)
{
fprintf(stderr,"Enter the hostname \n");
exit(1);
}
hen=gethostbyname(argv[1]);if(
hen==NULL)
{
fprintf(stderr,"Hostnotfound\n");
}
printf("Hostnameis%s\n",hen->h_name);
printf("IPaddressis%s\n",inet_ntoa(*((structin_addr*)hen->h_addr)));
}
```

OUTPUT

Thus the above program udp performance using domain name server was executed and successfully.

Outcome:

To understand the DNS client server for resolving the hostname.

Experment-3

Implement on a data set of characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCIP.

Introduction to Cyclic Redundancy Check:

CRC method can detect a single burst of length n, since only one bit per column will be changed, a burst of length n+1 will pass undetected, if the first bit is inverted, the last bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the n columns will have the correct parity that is 0.5. so the probability of a bad block being expected when it should not be 2 power(-n). This scheme sometimes is known as Cyclic Redundancy Code.

Program Algorithm/Flowchart:

```
Begin
Step 1:Declare I, j , fr[8], dupfr[11], recfr[11], tlen, flag, gen[4], genl, frl, rem[4] as integer
Step 2: initialize frl=8 and genl=4
Step 3: initialize i=0
Step 4: Repeat step(5to7) until i<frl
Step 5: read fr[i]
Step 6: dupfr[i]=fr[i]
Step 7: increment i
Step 8: initialize i=0
Step 9: repeat step(10to11) until i<genl
Step 10: read gen[i]
Step 11: increment i
Step 12: tlen=frl+genl-1
Step 13: initialize i=frl
Step 14: Repeat step(15to16) until i<tlen
Step 15: dupfr[i]=0
Step 16: increment i
Step 17: call the function remainder(dupfr)
Step 18: initialize i=0
Step 19: repeat step(20 to 21) until j<genl
Step 20: recfr[i]=rem[j]
Step 21: increment I and j
Step 22: call the function remainder(dupfr)
Step 23: initialize flag=0 and i=0
Step 24: Repeat step(25to28) until i<4
Step 25: if rem[i]!=0 then
Step 26: increment flag
Step 27: end if
Step 28: increment i
Step 29: if flag=0 then
Step 25: print frame received correctly
Step 25: else
Step 25: print the received frame is wrong
```

```
End
```

Function: Remainder(int fr[])

```
Begin
Step 1: Declare k,k1,I,j as integer
Step 2: initialize k=0;
Step 3: repeat step(4 to 14) until k< frl
Step 4: if ((fr[k] == 1) then
Step 5: k1=k
Step 6: initialize i=0, j=k
Step 7: repeat step(8 to 9) until i< genl
Step 8: rem[i] =fr[j] exponential gen[i]
Step 9: increment I and j
Step 10: initialize I = 0
Step 11: repeat step(12to13) until I <genl
Step 12: fr[k1] = rem[i]
Step 13: increment k1 and i
Step 14: end if
End
```

Program Code: // Program for Cyclic Redundency Check

```c
#include<stdio.h>
int gen[4],genl,frl,rem[4];
void main()
{
int i,j,fr[8],dupfr[11],recfr[11],tlen,flag;
frl=8; genl=4;
printf("Enter frame:");
for(i=0;i<frl;i++)
{
scanf("%d",&fr[i]);
dupfr[i]=fr[i];
}
printf("Enter generator:");
for(i=0;i<genl;i++)
scanf("%d",&gen[i]);
tlen=frl+genl-1;
for(i=frl;i<tlen;i++)
{
dupfr[i]=0;
}
remainder(dupfr);
for(i=0;i<frl;i++)
{
recfr[i]=fr[i];
}
for(i=frl,j=1;j<genl;i++,j++)
{
```

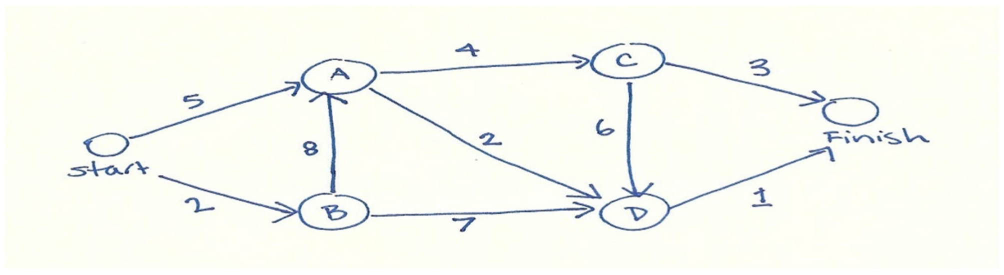
```
recfr[i]=rem[j];
}
remainder(recfr);
flag=0;
for(i=0;i<4;i++)
{
if(rem[i]!=0)
flag++;
}
if(flag==0)
{
printf("frame received correctly");
}
else
{
printf("the received frame is wrong");
}
}
remainder(int fr[])
{
int k,k1,i,j;
for(k=0;k<frl;k++)
{
if(fr[k]==1)
{
k1=k;
for(i=0,j=k;i<genl;i++,j++)
{
rem[i]=fr[j]^gen[i];
}
for(i=0;i<genl;i++)
{
fr[k1]=rem[i];
k1++;
}
}
}
}
```

Program Output:

```
Enter frame: smcecse
Enter generator: frame received correctly
------------------
(program exited with code: 24)
Press return to continue
```

Experment-4

Implement Dijkstra's algorithm to compute the Shortest path thru a given graph.



Program Algorithm/Flowchart:

```
Begin
Step1: Declare array path [5] [5], min, a [5][5], index, t[5];
Step2: Declare and initialize st=1,ed=5
Step 3: Declare variables i, j, stp, p, edp
Step 4: print "enter the cost "
Step 5: i=1
Step 6: Repeat step (7 to 11) until (i<=5)
Step 7: j=1
Step 8: repeat step (9 to 10) until (j<=5)
Step 9: Read a[i] [j]
Step 10: increment j
Step 11: increment i
Step 12: print "Enter the path"
Step 13: read p
Step 14: print "Enter possible paths"
Step 15: i=1
Step 16: repeat step(17 to 21) until (i<=p)
Step 17: j=1
Step 18: repeat step(19 to 20) until (i<=5)
Step 19: read path[i][j]
Step 20: increment j
Step 21: increment i
Step 22: j=1
Step 23: repeat step(24 to 34) until(i<=p)
Step 24: t[i]=0
Step 25: stp=st
Step 26: j=1
Step 27: repeat step(26 to 34) until(j<=5)
Step 28: edp=path[i][j+1]
Step 29: t[i]= [ti]+a[stp][edp]
Step 30: if (edp==ed) then
Step 31: break;
Step 32: else
Step 33: stp=edp
Step 34: end if
Step 35: min=t[st]
Step 36: index=st
Step 37: repeat step( 38 to 41) until (i<=p)
Step 38: min>t[i]
Step 39: min=t[i]
Step 40: index=i
Step 41: end if
Step 42: print" minimum cost" min
Step 43: print" minimum cost pth"
Step 44: repeat step(45 to 48) until (i<=5)
Step 45: print path[index][i]
Step 46: if(path[idex][i]==ed) then
Step 47: break
Step 48: end if
End
```

Program Code: Shortest Path for a given graph

```
#include<stdio.h>
void main()
{
int path[5][5],i,j,min,a[5][5],p,st=1,ed=5,stp,edp,t[5],index;
printf("Enter the cost matrix\n");
for(i=1;i<=5;i++)
for(j=1;j<=5;j++)
scanf("%d",&a[i][j]);
printf("Enter the paths\n");
scanf("%d",&p);
printf("Enter possible paths\n");
for(i=1;i<=p;i++)
for(j=1;j<=5;j++)
scanf("%d",&path[i][j]);
for(i=1;i<=p;i++)
{
t[i]=0;
stp=st;
for(j=1;j<=5;j++)
{
edp=path[i][j+1];
t[i]=t[i]+a[stp][edp];
if(edp==ed)
break;
else
stp=edp;
}
}
min=t[st];index=st;
for(i=1;i<=p;i++)
{
if(min>t[i])
{
min=t[i];
index=i;
}
}
printf("Minimum cost %d",min);
printf("\n Minimum cost path ");
for(i=1;i<=5;i++)
{
printf("--> %d",path[index][i]);
if(path[index][i]==ed)
break;
}
}
```

Program Output:

```
Enter the cost matrix
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
Enter the paths
2
Enter possible paths
1 2 3 4 5
1 2 3 4 5
Minimum cost 14
```

Experment-5

Take an example subnet graph with weights indicating delay between nodes. Now obtain Routing table art each node using distance vector routing algorithm

DESCRIPTION

Routing algorithms are a part of the network layer software whose responsibility is to decide what destination output line should be selected for successful journey completion of the packet from the source machine to destination machine. The network layer of ISO-OSI reference model is responsible for getting packets from the source to the destination. It uses routing algorithms to effectively use all communication lines and routers present in the communication subnet and decide which lines to use for forwarding incoming packets.

Classification of Routing Algorithms

Routing algorithms are of classified on the basis of routing decisions made by them.They are either non adaptive or adaptive. Non adaptive algorithms do not change routing decisions on the basis of measurements of current traffic and topology, where as adaptive routing algorithms do change routing decisions on basis of measurements of current traffic and topology.Nonadaptive algorithms are also called static algorithms and adaptive algorithms are also called dynamic algorithms.

Types of Routing Algorithms

Shortest Path Routing

Shortest path routing is a static (non adaptive) routing algorithm. In this algorithm a graph of the communication subnet is created with each node representing a router and each edge representing a communication line(link).Finding the shortest path between two nodes is done on the basis of some type of measurements. The se measurements are called metrics.Some of the metrics used are number of hops, physical distance, mean queuing and transmission delays. Metrics like transmission delays and queue length are measured hourly for standard test packets sent out on each line of the communication subnet. Shortest path can be calculated on the basis of any one of the criteria or a combination of criteria. The most simple and widely used algorithm for finding the shortest path is the Dijkstra's algorithm.

Distance Vector Routing

Most computer networks in operation today use dynamic routing algorithms rather than static routing algorithms. One of the widely used dynamic algorithms is Distance Vector routing, which is also known as Bellman Ford algorithm. At able maintained by a router is called a vector. Vectors contain information on how to get to the destination using the best possible path to get to it. It also contains an entry for each router in the subnet. Each of the routing tables is updated continuously. This can be done by exchanging information with the neighboring routers. For measuring the optimal distance between each node, metrics are used, similar to ones described above in the classification section of the article. The optimal path is followed till the packet reaches the destination machine.

Distance vector algorithms use the Bellman-Ford algorithm. This approach assigns a number, the cost, to each of the links between each nod either network. Nodes will send information from point

A to point B via the path that results in the lowest total cost (i.e. the sum of the costs of the links between the nodes used).The algorithm operates in a very simple manner. When a node first starts, it only knows of its immediate neighbors, and the direct cost involved in reaching them. Each node, on a regular basis, sends to each neighbor its own current idea of the total cost to get to all the destinations it knows of. The neighboring node(s) examine this information, and compare it to what they already 'know'; anything which represents an improvement on what they already have, they insert in their own routing table(s).Overtime, all the nodes in the network will discover the best next hop for all destinations, and the best total cost.

When one of the nodes involved goes down, those nodes which used it as their next hop for certain destinations discard those entries, and create new routing-table information. They then pass this information to all adjacent nodes, which then repeat the process. Eventually all the nodes in the network receive the updated information, and will then discover new paths to all the destinations which they can still "reach".

```
/*
Distance Vector Routing in this program is implemented using Bellman Ford Algorithm:-
*/
#include<stdio.h>
struct node
{
unsigned dist[20];
unsigned from[20];
}rt[10];
int main()
{
int costmat[20][20];
int nodes,i,j,k,count=0;
printf("\nEnter the number of nodes : ");
scanf("%d",&nodes);//Enter the nodes
printf("\nEnter the cost matrix :\n");
for(i=0;i<nodes;i++)
{
for(j=0;j<nodes;j++)
{
scanf("%d",&costmat[i][j]);
costmat[i][i]=0;
rt[i].dist[j]=costmat[i][j];//initialise the distance equal to cost matrix
rt[i].from[j]=j;
}
}
do
{
count=0;
for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the direct distance from the
node i to k using the cost matrix
//and add the distance from k to node j
for(j=0;j<nodes;j++)
for(k=0;k<nodes;k++)
if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
{                                                    //We calculate the minimum distance
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
rt[i].from[j]=k;
count++;
}
}while(count!=0);
for(i=0;i<nodes;i++)
{
printf("\n\n For router %d\n",i+1);
for(j=0;j<nodes;j++)
{
printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);
}
}
printf("\n\n");
getch();
}
```

Output:

```
/*
A sample run of the program works as:-
Enter the number of nodes :
3
Enter the cost matrix :
0 2 7
2 0 1
7 1 0
For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 3 Distance 3
For router 2
node 1 via 1 Distance 2
node 2 via 2 Distance 0
node 3 via 3 Distance 1
For router 3
node 1 via 1 Distance 3
node 2 via 2 Distance 1
node 3 via 3 Distance 0
```

Experment-6

Aim:
Take an example subnet of hosts. Obtain broadcast tree for it.

Theory

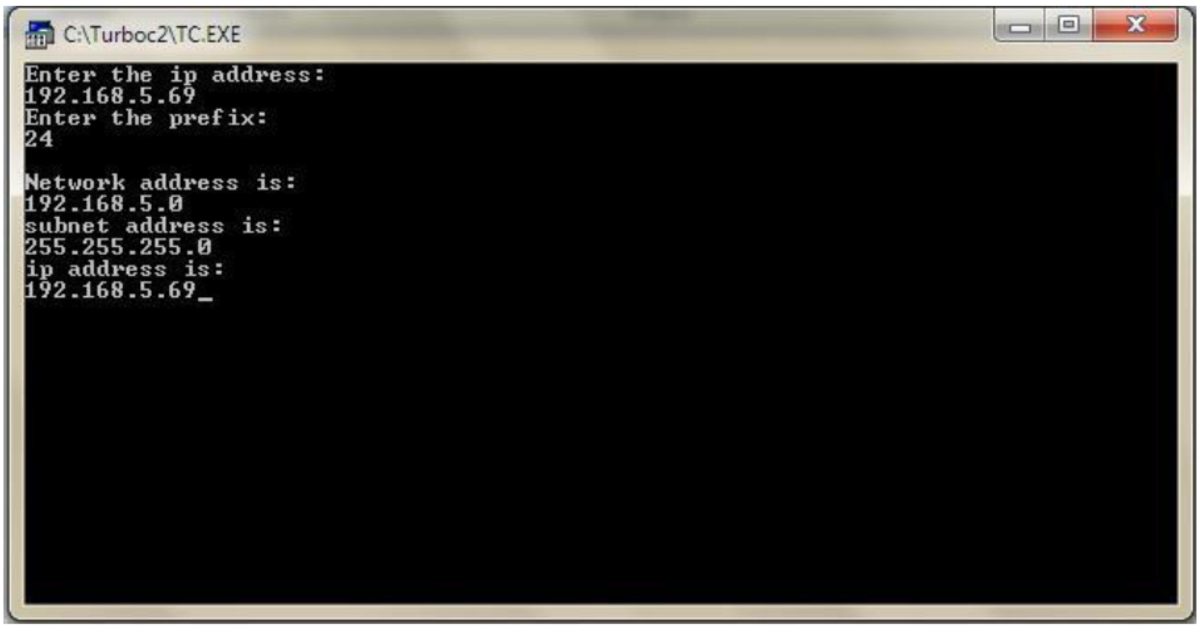
IP addressing is the allocation of unique ID to each and every system connected in a network to maintain communication among them throughout the affixed network. There are 5 classes of IP Addresses namely A through E with the range varying from one class to the other class.
A subnet is a network allocation to similar systems or same hierarchical systems present in a allocated network like an organization. Each and every system can be reached through a client-server computing environment where the server acts as the Master and the clients acts as the Slaves to form a Master-Slave computing environment. Below programs show the calculation of network addresses with subnet predefinition and subnet generation.

Program:

a)Network Address:

```
#include<stdio.h>
#include<conio.h>
void main()
{
unsigned int compad[4];
unsigned int mask[4];
unsigned int netadr[4];
int i;
clrscr();
printf("Enter the ip address:\n");
scanf("%u%*c%u%*c%u%*c%u%*c",&compad[3],&compad[2],&compad[1],&compad[0]);
printf("Enter the subnet address:\n");
scanf("%u%*c%u%*c%u%*c%u%*c",&mask[3],&mask[2],&mask[1],&mask[0]);
for(i=0;i<4;i++)
{
netadr[i]= compad[i]&mask[i];
}
printf("\nNetwork address is:\n");
printf("%u.%u.%u.%u",netadr[3],netadr[2],netadr[1],netadr[0]);
printf("\nsubnet address is:\n");
printf("%u.%u.%u.%u",mask[3],mask[2],mask[1],mask[0]);
printf("\nip address is:\n");
printf("%u.%u.%u.%u",compad[3],compad[2],compad[1],compad[0]);
getch();
}
```

Output:

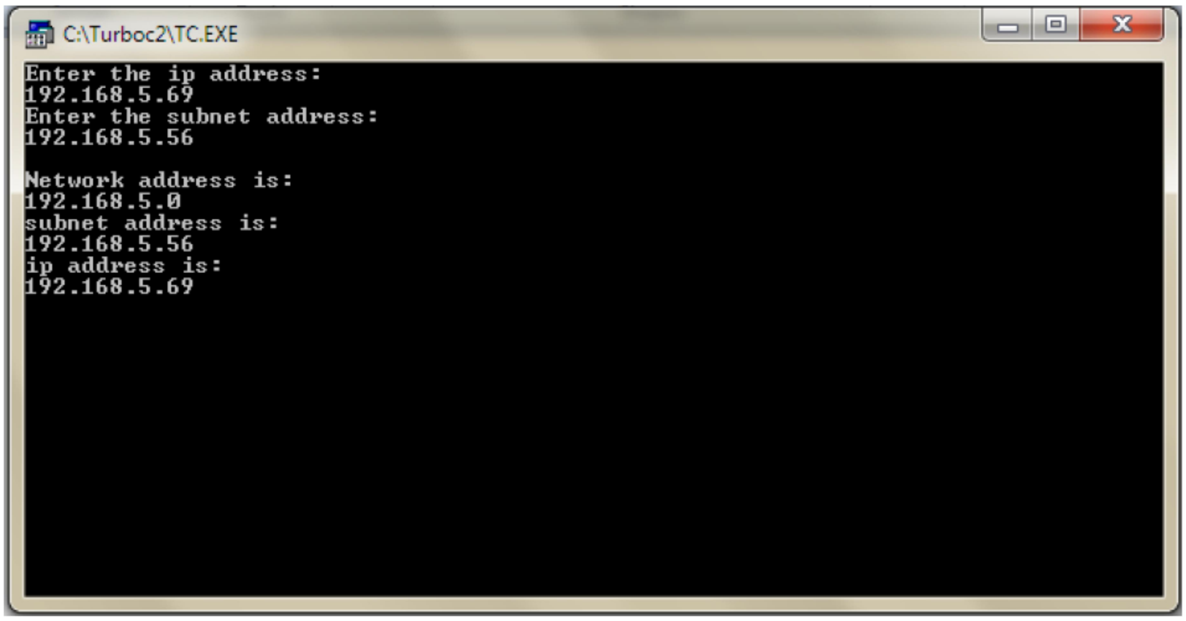


```
Enter the ip address:
192.168.5.69
Enter the prefix:
24

Network address is:
192.168.5.0
subnet address is:
255.255.255.0
ip address is:
192.168.5.69_
```

b)Network address with automatic subnet address generation:

```
#include<stdio.h>
#include<conio.h>
void main()
{
unsigned int compad[4];
unsigned int mask[4];
unsigned int netadr[4];
unsigned long int ma=0;
int i,pre;
clrscr();
printf("Enter the ip address:\n");
scanf("%u%*c%u%*c%u%*c%u%*c",&compad[3],&compad[2],&compad[1],&compad[0]);
printf("Enter the prefix:\n");
scanf("%ul",&pre);
for(i=(32-pre);i<32;i++)
ma=ma|(1<<i);
for(i=0;i<4;i++)
{
mask[i]=ma%256;
ma=ma/256;
}
for(i=0;i<4;i++)
{
netadr[i]= compad[i]&mask[i];
}
printf("\nNetwork address is:\n");
printf("%u.%u.%u.%u",netadr[3],netadr[2],netadr[1],netadr[0]);
printf("\nsubnet address is:\n");
printf("%u.%u.%u.%u",mask[3],mask[2],mask[1],mask[0]);
printf("\nip address is:\n");
printf("%u.%u.%u.%u",compad[3],compad[2],compad[1],compad[0]);
getch();
}
```

Output:



C:\Turboc2\TC.EXE

```
Enter the ip address:
192.168.5.69
Enter the subnet address:
192.168.5.56

Network address is:
192.168.5.0
subnet address is:
192.168.5.56
ip address is:
192.168.5.69
```

Experment-7

AIM:- Write a client-server application for chat using UDP

Procedure:

UDP - User Datagram Protocol sockets

UDP is an alternative protocol to the more commonly used TCP protocol. It is a connection-less protocol where you directly send packets without have to establish a proper connection.

A socket based on UDP is connectionless and is based on datagrams.
The chunk of data that is sent using UDP is called a datagram or a UDP packet. Each UDP packet has the data, the destination IP address, and the destination port number.
The connectionless socket do not establish a connection before they communicate.
UDP is an unreliable protocol because it does not guarantee the delivery and the order of arrived packets.
In a connectionless protocol, UDP, there will not be a server socket.
In UDP connection, client and server sends or receives a chunk of data without any prior knowledge of communication between them.
Each chunk of data sent to the same destination is independent of the previously sent data.
The following two classes are used when coding a UDP connection.
DatagramPacket class represents a UDP datagram.
DatagramSocket class represents a UDP socket that is used to send or receive a datagram packet.
The following code shows how to create a UDP Socket bound to a port number 12345 at localhost.

DatagramSocket udpSocket = new DatagramSocket(12345, "localhost");
The DatagramSocket class provides a bind() method, which lets you bind the socket to a local IP address and a local port number.

DatagramPacket
A DatagramPacket contains three things:

- a destination IP address
- a destination port number
- and the data.

Constructors of the DatagramPacket class to create a packet to receive data are as follows:

DatagramPacket(byte[] buf, int length)
DatagramPacket(byte[] buf, int offset, int length)

Constructors of the DatagramPacket class to create a packet to send data are as follows:

DatagramPacket(byte[] buf, int length, InetAddress address, int port)
DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)
DatagramPacket(byte[] buf, int length, SocketAddress address)
DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)

The following code demonstrates how to create a datagram packet:

The following code creates a packet to receive 1024 bytes of data.

```
byte[] data = new byte[1024];
DatagramPacket packet  = new DatagramPacket(data, data.length);
```

The following code creates a packet that has buffer size of 1024, and receive data starting at offset 8 and it will receive only 32 bytes of data.

```
byte[]  data2 = new byte[1024];
DatagramPacket packet2  = new DatagramPacket(data2, 8, 32);
```

Data in the packet always has offset and length specified. We need to use offset and length to read the data from a packet.

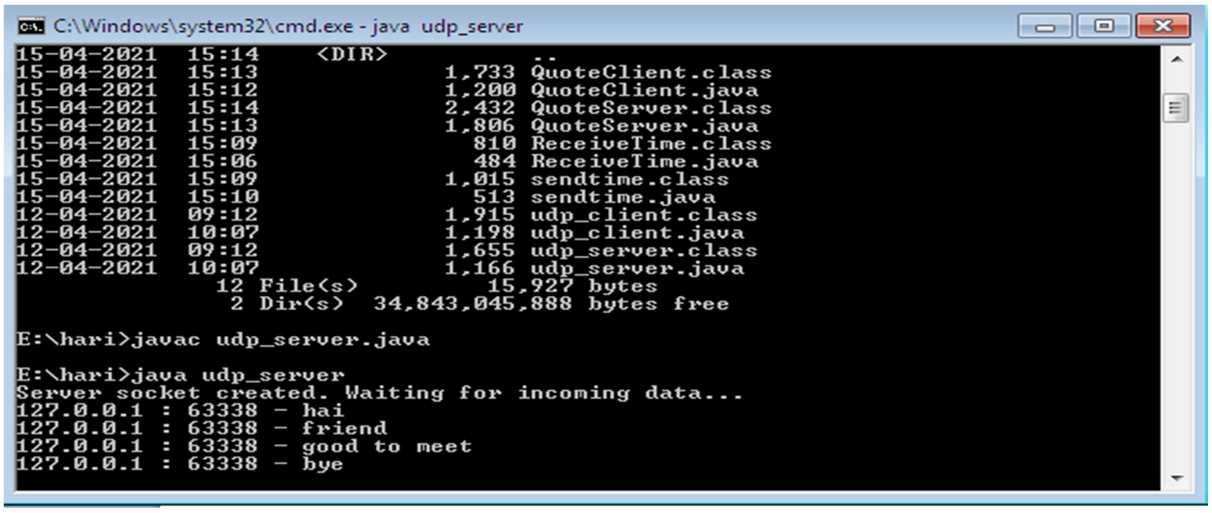
UDP packets have smaller headers compared to TCP headers. Also data communication is faster since no acknowledgement is exchanged for reliable packet delivery.

In this quick tutorial we shall learn how to use udp sockets to make a simple client and server program.
UDP sockets can be used in java with the DatagramSocket class.

UDP Server

```
import java.io.*;
import java.net.*;
public class udp_server
{
public static void main(String args[])
{
DatagramSocket sock = null;
try
{
//1. creating a server socket, parameter is local port number
sock = new DatagramSocket(7777);
//buffer to receive incoming data
byte[] buffer = new byte[65536];
DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
//2. Wait for an incoming data
echo("Server socket created. Waiting for incoming data...");
//communication loop
while(true)
{
sock.receive(incoming);
byte[] data = incoming.getData();
String s = new String(data, 0, incoming.getLength());
//echo the details of incoming data - client ip : client port - client message
echo(incoming.getAddress().getHostAddress() + " : " + incoming.getPort() + " - " + s);
s = "OK : " + s;
DatagramPacket dp = new DatagramPacket(s.getBytes() , s.getBytes().length ,
incoming.getAddress() , incoming.getPort());
sock.send(dp);
}
}
catch(IOException e)
{
System.err.println("IOException " + e);
}
}
//simple function to echo data to terminal
public static void echo(String msg)
{
System.out.println(msg);
}
}
```

Output:



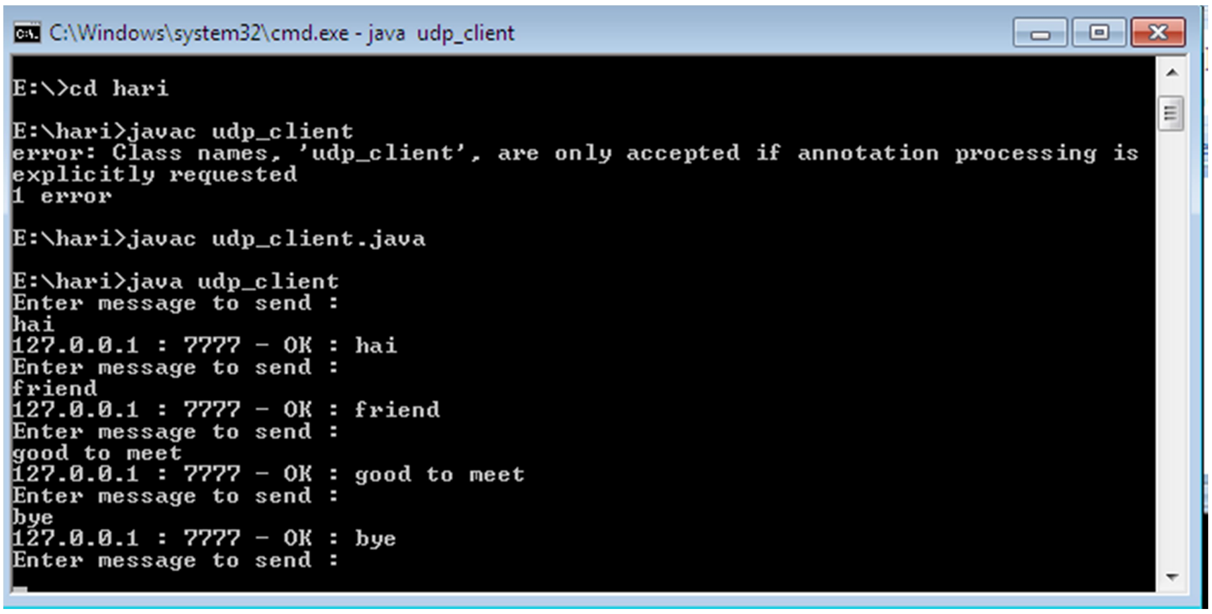
```
15-04-2021  15:14    <DIR>          ..
15-04-2021  15:13             1,733 QuoteClient.class
15-04-2021  15:12             1,200 QuoteClient.java
15-04-2021  15:14             2,432 QuoteServer.class
15-04-2021  15:13             1,806 QuoteServer.java
15-04-2021  15:09               810 ReceiveTime.class
15-04-2021  15:06               484 ReceiveTime.java
15-04-2021  15:09             1,015 sendtime.class
15-04-2021  15:10               513 sendtime.java
12-04-2021  09:12             1,915 udp_client.class
12-04-2021  10:07             1,198 udp_client.java
12-04-2021  09:12             1,655 udp_server.class
12-04-2021  10:07             1,166 udp_server.java
              12 File(s)         15,927 bytes
               2 Dir(s)  34,843,045,888 bytes free

E:\hari>javac udp_server.java

E:\hari>java udp_server
Server socket created. Waiting for incoming data...
127.0.0.1 : 63338 - hai
127.0.0.1 : 63338 - friend
127.0.0.1 : 63338 - good to meet
127.0.0.1 : 63338 - bye
```

UDP_Client

```
import java.io.*;
import java.net.*;
public class udp_client
{
public static void main(String args[])
{
DatagramSocket sock = null;
int port = 7777;
String s;
BufferedReader cin = new BufferedReader(new InputStreamReader(System.in));
try
{
sock = new DatagramSocket();
InetAddress host = InetAddress.getByName("localhost");
while(true)
{
//take input and send the packet
echo("Enter message to send : ");
s = (String)cin.readLine();
byte[] b = s.getBytes();
DatagramPacket  dp = new DatagramPacket(b , b.length , host , port);
sock.send(dp);
//now receive reply
//buffer to receive incoming data
byte[] buffer = new byte[65536];
DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
sock.receive(reply);
byte[] data = reply.getData();
s = new String(data, 0, reply.getLength());
//echo the details of incoming data - client ip : client port - client message
echo(reply.getAddress().getHostAddress() + " : " + reply.getPort() + " - " + s);
}
}
catch(IOException e)
{
System.err.println("IOException " + e);
}
}
//simple function to echo data to terminal
public static void echo(String msg)
{
```

```
System.out.println(msg);
}
}
```

Output:



```
E:\>cd hari

E:\hari>javac udp_client
error: Class names, 'udp_client', are only accepted if annotation processing is
explicitly requested
1 error

E:\hari>javac udp_client.java

E:\hari>java udp_client
Enter message to send :
hai
127.0.0.1 : 7777 - OK : hai
Enter message to send :
friend
127.0.0.1 : 7777 - OK : friend
Enter message to send :
good to meet
127.0.0.1 : 7777 - OK : good to meet
Enter message to send :
bye
127.0.0.1 : 7777 - OK : bye
Enter message to send :
```

Experment-8

AIM:
To implement raw sockets (like packet capturing and filtering using java)

DESCRIPTION:
Raw socket is created to define the transmission of packet. Packet is captured for checking error. Error containing packets are filtered during transmission. Raw socket using TCP/IP protocol is created . Packet length is defined along with TCP header attached. Packets with error are checked using CRC mechanism and filtered.

ALGORITHM :
1. Start the program and to include the necessary header files.
2. To define the packet length.
3. To declare the IP header structure using TCP header.
4. Using simple checksum process to check the process.
5. Using TCP \IP communication protocol to execute the program.
6. And using TCP\IP communication to enter the Source IP and port number and Target IP address and port number.
7. The Raw socket () is created and accept the Socket ( ) and Send to ( ), ACK
8. Stop the program

PROGRAM:
```
//---cat rawtcp.c---
// Run as root or SUID 0, just datagram no data/payload
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
// Packet length
#define PCKT_LEN 8192
// May create separate header file (.h) for all
// headers' structures
// IP header's structure
struct  ipheader {
unsigned char iph_ihl:5, /* Little-endian */
iph_ver:4;
unsigned char iph_tos;
unsigned short int iph_len;
unsigned short int iph_ident;
unsigned char iph_flags;
unsigned short int iph_offset;
unsigned char iph_ttl;
unsigned char iph_protocol;
unsigned short int iph_chksum;
unsigned int iph_sourceip;
unsigned int iph_destip;
};
/* Structure of a TCP header */
struct tcpheader {
unsigned short int tcph_srcport;
unsigned short int tcph_destport;
unsigned int tcph_seqnum;
unsigned int tcph_acknum;
unsigned char tcph_reserved:4, tcph_offset:4;
// unsigned char tcph_flags;
unsigned int
tcp_res1:4, /*little-endian*/
tcph_hlen:4, /*length of tcp header in 32-bit
words*/
tcph_fin:1, /*Finish flag "fin"*/
tcph_syn:1, /*Synchronize sequence numbers to
start a connection*/
tcph_rst:1, /*Reset flag */
tcph_psh:1, /*Push, sends data to the
application*/
tcph_ack:1, /*acknowledge*/ tcph_urg:1, /*urgent pointer*/
tcph_res2:2;
unsigned short int tcph_win;
unsigned short int tcph_chksum;
unsigned short int tcph_urgptr;
};
// Simple checksum function, may use others such as Cyclic
Redundancy Check, CRC
unsigned short csum(unsigned short *buf, int len)
{
unsigned long sum;
for(sum=0; len>0; len--)
sum += *buf++;
sum = (sum >> 16) + (sum &0xffff);
sum += (sum >> 16);
return (unsigned short)(~sum);
}
int main(int argc, char *argv[])
{
int sd;
```

```
// No data, just datagram
char buffer[PCKT_LEN];
// The size of the headers
struct ipheader *ip = (struct ipheader *) buffer;
struct tcpheader *tcp = (struct tcpheader *) (buffer +
sizeof(struct ipheader));
struct sockaddr_in sin, din;
int one = 1;
const int *val = &one;
memset(buffer, 0, PCKT_LEN); if(argc != 5)
{
printf("
- Invalid parameters!!!
\n");
printf("
- Usage: %s <source hostname/IP> <source port>
<target hostname/IP> <target port>
\n", argv[0]);
exit(
-1);
}
sd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
if(sd < 0)
{
perror("socket() error");
exit(
-1);
}
else
printf("socket()
-SOCK_RAW and tcp protocol is OK.
\n");
// The source is redundant, may be used later if needed
// Address family
sin.sin_family = AF_INET;
sin_family = AF_INET;
// Source port, can be any, modify as needed
sin.sin_port = htons(atoi(argv[2]));
din.sin_port = htons(atoi(argv[4]));
// Source IP, can be any, modify as needed
sin.sin_addr.s_addr = inet_addr(argv[1]);
din.sin_addr.s_addr = inet_addr(argv[3]);
// IP structure
ip->iph_ihl = 5;
```

```
ip->iph_ver = 4;
ip->iph_tos = 16;
ip->iph_len = sizeof(struct ipheader) + sizeof(struct tcpheader);
ip->iph_ident = htons(54321);
ip->iph_offset = 0;
ip->iph_ttl = 64;
ip->iph_protocol = 6; // TCP
ip->iph_chksum = 0; // Done by kernel
// Source IP, modify as needed, spoofed, we accept through
command line argument
ip->iph_sourceip = inet_addr(argv[1]);
// Destination IP, modify as needed, but here we accept
through command line argument
ip->iph_destip = inet_addr(argv[3]);
// The TCP structure. The source port, spoofed, we accept
through the command line
tcp->tcph_srcport = htons(atoi(argv[2]));
// The destination port, we accept through command line
tcp->tcph_destport = htons(atoi(argv[4]));
tcp->tcph_seqnum = htonl(1);
tcp->tcph_acknum = 0;
tcp->tcph_offset = 5;
tcp->tcph_syn = 1;
tcp->tcph_ack = 0;
tcp->tcph_win = htons(32767);
tcp->tcph_chksum = 0; // Done by kernel
tcp->tcph_urgptr = 0;
// IP checksum calculation
ip->iph_chksum = csum((unsigned short *) buffer,
(sizeof(struct ipheader) + sizeof(struct tcpheader)));
// Inform the kernel do not fill up the headers' structure,
we fabricated our own
if(setsockopt(sd, IPPROTO_IP, IP_HDRINCL, val, sizeof(one))
< 0)
{
perror("setsockopt() error");
exit(-1);
} else
printf("setsockopt() is OK\n");
printf("Using:::::Source IP: %s port: %u, Target IP: %s
port: %u.\n", argv[1], atoi(argv[2]), argv[3],
atoi(argv[4]));
// sendto() loop, send every 2 second for 50 counts
unsigned int count;
```

```
for(count = 0; count < 20; count++)
{
if(sendto(sd, buffer, ip->iph_len, 0, (struct sockaddr
*)&sin, sizeof(sin)) < 0)
// Verify
{
perror("sendto() error");
exit(-1);
}
else
printf("Count #%u - sendto() is OK\n", count);
sleep(2);
}
close(sd);
return 0;
}
```

OUTPUT:

```
Setsockopt() is OK
Using Source IP : 172.17.1.72
Port : 3001
Target IP: 172.17.1.76
Port : 5001
```

RESULT:

Thus the above programs using raw sockets TCP \IP (like packet capturing and filtering) was executed and successfully.

Experiment- 9

AIM:

Write a C program to perform sliding window protocol.

DESCRIPTION A **sliding window frame** of data is sent from server to client. If that frame is received correctly at the client then acknowledgement signal is sent to server. Otherwise negative acknowledgement is sent. Frame size for the sliding window is provided by the user. Then the frame of data is sent with ACK signal or NACK signal.
A **sliding window protocol** is a feature of packet-based data transmission protocol. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in the data link layer of OSI model as well as in the TCP.
A **data frame** is an aggregate of numerous, partly overlapping collections of data and metadata that have been derived from massive amounts of network activity such as content production, consumption, and other user behavior.

ALGORITHM

1. Start the program.
2. Get the frame size from the user.
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program.

PROGRAM:

Sliding window protocol Client:

```
 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <sys/types.h>
#include <sys/ipc.h>
 #include <sys/msg.h>
 struct mymsgbuf
{
 long mtype;
 char mtext[25];
 };
FILE *fp;
 int main()
{
struct mymsgbuf buf;
 int msgid;
 int i=0,s;
 int count=0,frmsz;
int a[100];
char d;
 if((msgid=msgget(89,IPC_CREAT|0666))==-1)
{
printf("\n ERROR IN MSGGET");
exit(0);
 }
 printf("\n Enter the frame size:");
if((fp=fopen("check","r"))==NULL) printf("\n FILE NOT OPENED");
else
printf("\n FILE OPENED");
while(!feof(fp))
{
 d=getc(fp);
a[i]=d;
 i++;
}
s=i;
 for(i=0;i<frmsz;i++)
//print from the check file
 printf("\t %c",a[i]);
for(i=0;i<frmsz;i++)
{
 if((msgrcv(msgid,&buf,sizeof(buf),0,1))==-1)
{
printf("\n ERROR IN MSGRCV");
exit(0);
}
printf("\n RECEIVED FRAMES ARE:%c",buf.mtext[i]);
}
for(i=0;i<frmsz;i++)
{
 if(a[i]==buf.mtext[i])
 count++;
}
if(count==0)
{
printf("\n FRAMES WERE NOT RECEIVED IN CORRECT SEQ");
exit(0);
} if(count==frmsz)
{
printf("\n FRAMES WERE RECEIVED IN CORRECT SEQ");
} else
{
printf("\n FRAMES WERE NOT RECEIVED IN CORRECT SEQ");
}
}
```

//Sliding Window Protocol – Server

```
#include <stdio.h>
#include <stdlib.h>
 #include <string.h>
 #include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mymsgbuf
{
 long mtype;
 char mtext[25];
};
FILE *fp;
 int main()
{
struct mymsgbuf  buf;
int si,ei,sz; int msgid;
int i=0,s;
 int a[100];
char d;
 if((fp=fopen("send","r"))==NULL)
printf("\n FILE NOT OPENED");
else
printf("\n FILE OPENED");
printf("\n Enter starting and ending index of frame array:"); scanf("%d%d",&si,&ei); sz=ei-
si; if((msgid=msgget(89,IPC_CREAT|0666))==-1)
{
printf("\n ERROR IN MSGGET"); exit(0);
}
while(!feof(fp))
{ d=getc(fp); a[i]=d; i++;
}s
=i; buf.mtype=1; for(i=si;i<=ei;i++)
{
buf.mtext[i]=a[i];
}
for(i=si;i<=ei;i++) //the frames to be sent printf("\t %c",buf.mtext[i]); for(i=0;i<=sz;i++)
{ if((msgsnd(msgid,&buf,sizeof(buf),0))==-1)
{
printf("\n ERROR IN MSGSND"); exit(0);
}}
printf("\n FRAMES SENT"); return 0;
}
```

OUTPUT:

```
Enter the frame size : 5 File opened
Enter starting & ending index of frame array : 0 9 Frames sent
Received frames are: 0 3 6 7 9
```

Experiment- 10

AIM:
To write a program to get the MAC address from the system using Address Resolution Protocol.

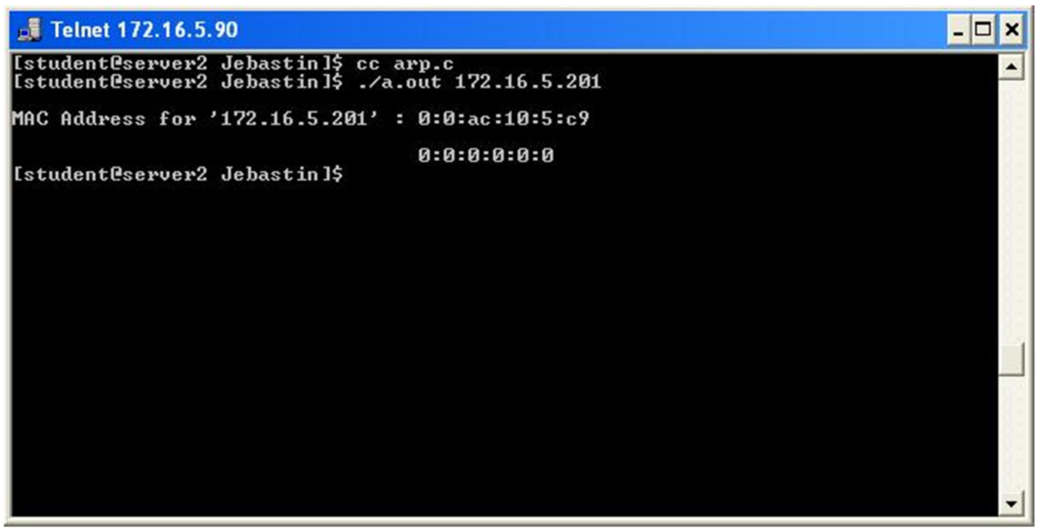
ALGORITHM:

```
STEP 1: Start
STEP 2: Declare the variables and structure for the socket
STEP 3: Specify the family, protocol, IP address and port number
STEP 4: Create a socket using socket() function
STEP 5: Call memcpy() and strcpy functions
STEP 6: Display the MAC address
STEP 7: Stop
```

SOURCE CODE:

```
#include<sys/types.h>
#include<sys/socket.h>
#include<net/if_arp.h>
#include<sys/ioctl.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<math.h>
#include<complex.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<netinet/if_ether.h>
#include<net/ethernet.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
        struct sockaddr_in sin={0};
        struct arpreq myarp={{0}};
        unsigned char *ptr;
        int sd;
        sin.sin_family=AF_INET;
        if(inet_aton(argv[1],&sin.sin_addr)==0)
        {
                printf("IP address Entered '%s' is not valid \n",argv[1]);
                exit(0);
        }
        memcpy(&myarp.arp_pa,&sin,sizeof(myarp.arp_pa));
        strcpy(myarp.arp_dev,"echo");
        sd=socket(AF_INET,SOCK_DGRAM,0);

       if(ioctl(sd,SIOCGARP,&myarp)==1)
        {
                printf("No Entry in ATP cache for '%s'\n",argv[1]);
                exit(0);
        }
        ptr=&myarp.arp_pa.sa_data[0];
        printf("\nMAC Address for '%s' : ",argv[1]);
        printf("%x:%x:%x:%x:%x:%x\n",*ptr,*(ptr+1),*(ptr+2),*(ptr+3),*(ptr+4),*(ptr+5));
        printf("\n\t\t\t\t%x:%x:%x:%x:%x:%x\n", myarp.arp_ha.sa_data[0],
        myarp.arp_ha.sa_data[1], myarp.arp_ha.sa_data[2],
        myarp.arp_ha.sa_data[3], myarp.arp_ha.sa_data[4], myarp.arp_ha.sa_data[5]);
        return 0;
}
```

OUTPUT:



```
[student@server2 Jebastin]$ cc arp.c
[student@server2 Jebastin]$ ./a.out 172.16.5.201

MAC Address for '172.16.5.201' : 0:0:ac:10:5:c9

                                 0:0:0:0:0:0
[student@server2 Jebastin]$
```

Experiment- 11

Aim:- Simulate the Implementing Routing Protocols using border gateway protocol(BGP)

Step by step BGP Algorithm

The number of nodes is obtained and stored in variable ‘n’ The distance between all nodes is accepted in the 4 two dimensional array ‘a’ It is values are printed back in the form of matrix The shortest pat between any 2 nodes is found using the expression a[i][j]=a[i][k]+a[k][j] The distance between nodes to itself is made ‘0’ The output Matrix is printed back to the user Finally terminate Border Gateway protocol program.

Program :

```
#include<stdio.h>
main()
{
int i,n,j,k,a[10][10],b[10][10];
printf("ENTER THE NO OF NODES");
scanf("%d",&n);
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("ENTER THE DISTANCE BETWEEN HOSTS %d,%d :",i+1,j+1);
scanf("%d",&a[i][j]);
}
}
printf("THE GIVEN INPUT\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
for(k=0;k<n;k++)
{
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(a[i][j]>a[i][k]+a[k][j])
{
a[i][j]=a[i][k]+a[k][j];
}
}
}
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
b[i][j]=a[i][j];
if(i==j)
b[i][j]=0;
}
}
printf("OUTPUT MATRIX \n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
}
```

Output CS1305- NETWORK LAB

```
[it28@localhost]$cc protocol.c
[it28@localhost]$./a.out
Enter the no of nodes: 2
ENTER THE DISTANCE BETWEEN HOSTS 1 1:1
ENTER THE DISTANCE BETWEEN HOSTS 1 2:2
ENTER THE DISTANCE BETWEEN HOSTS 2 1:3
ENTER THE DISTANCE BETWEEN HOSTS 2 2:4
The given output
1 2
3 4
Output Matrix
0 2
3 0
```

Experiment- 12

AIM :

Simulate the OPEN SHORTEST PATH FIRST routing protocol based on the cost assigned to the path.

```
#include <stdio.h>
#include <string.h>
int main( )
{
int count,src_router,i,j,k,w,v,min;
int cost_matrix[100][100],dist[100],last[100];
int flag[100];
printf("\n Enter the no of routers");
scanf("%d",&count);
printf("\n Enter the cost matrix values:");
for(i=0;i<count;i++)
{
for(j=0;j<count;j++)
{
printf("\n%d->%d:",i,j);
scanf("%d",&cost_matrix[i][j]);
if(cost_matrix[i][j]<0)cost_matrix[i][j]=1000;
}
}
printf("\n Enter the source router:");
scanf("%d",&src_router);
for(v=0;v<count;v++)
{
flag[v]=0;
last[v]=src_router;
dist[v]=cost_matrix[src_router][v];
}
flag[src_router]=1;
for(i=0;i<count;i++)
{
min=1000;
for(w=0;w<count;w++)
{
if(!flag[w])
if(dist[w]<min)
{
v=w;
min=dist[w];
}
}
flag[v]=1;
for(w=0;w<count;w++)
{
if(!flag[w])
if(min+cost_matrix[v][w]<dist[w])
{
dist[w]=min+cost_matrix[v][w];
last[w]=v;
}
}
}
for(i=0;i<count;i++)
{
printf("\n%d==>%d:Path taken:%d",src_router,i,i);
w=i;
while(w!=src_router)
{
printf("\n<--%d",last[w]);w=last[w];
}
printf("\n Shortest path cost:%d",dist[i]);
}
}
```

Output:

```
Enter the no of routers3
Enter the cost matrix values:
0->0:
[exam47@cselinux ~]$ ./a.out
Enter the no of routers2
Enter the cost matrix values:
0->0:3
0->1:4
1->0:5
1->1:6
Enter the source router:1
1==>0:Path taken:0
<--1
Shortest path cost:5
1==>1:Path taken:1
Shortest path cost:6
```