PROGRAM:-

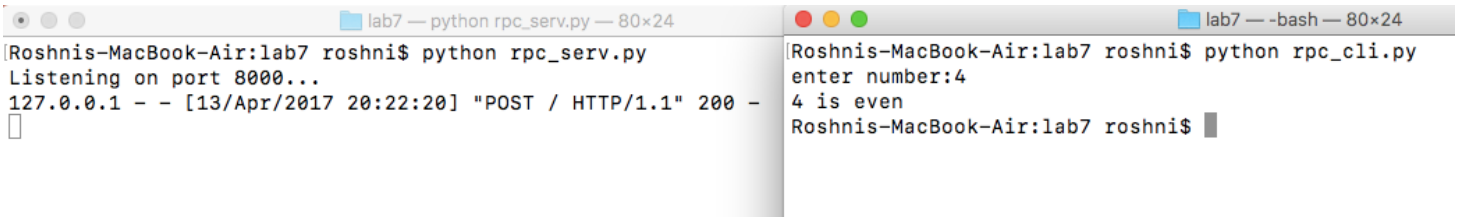
rpc_serv.py

```
import xmlrpclib
from SimpleXMLRPCServer import SimpleXMLRPCServer

def is_even(n):
        return n % 2 == 0

server = SimpleXMLRPCServer(("localhost", 8000))
print "Listening on port 8000..."
server.register_function(is_even, "is_even")
server.serve_forever()
```

rpc_client.py

```
import xmlrpclib
proxy = xmlrpclib.ServerProxy("http://localhost:8000/")
k=input("enter number:")
if proxy.is_even(k):
        ans="even"
else:
        ans="odd"
print str(k)+" is "+ans
```

OUTPUT:-



PROGRAM:-

Bellmanford.c

```
#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;
		 //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(i=0;i<nodes;i++)
			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\nFor router %d\n",i+1);
                printf("-----------------------------------");
                printf("\n|Destination\t|Via\t|Distance|");
                printf("\n-----------------------------------");
                for(j=0;j<nodes;j++)
                {
                        printf("\n|%d\t\t|%d\t|\t%d|",j+1,rt[i].from[j]+1,rt[i].dist[j]);
                        printf("\n_________________________________________");
                }
        }
        printf("\n\n");
        return 0;
}
```

OUTPUT:-

```
Roshnis-MacBook-Air:lab8 roshni$ gcc bellmanford.c
Roshnis-MacBook-Air:lab8 roshni$ ./a.out

Enter the number of nodes :3

Enter the cost matrix :
0 2 7
2 0 1
7 1 0


For router 1
-------------------------------------
|Destination    |Via    |Distance|
-------------------------------------
|1              |1      |       0|
-------------------------------------
|2              |2      |       2|
-------------------------------------
|3              |2      |       3|
-------------------------------------

For router 2
-------------------------------------
|Destination    |Via    |Distance|
-------------------------------------
|1              |1      |       2|
-------------------------------------
|2              |2      |       0|
-------------------------------------
|3              |3      |       1|
-------------------------------------

For router 3
-------------------------------------
|Destination    |Via    |Distance|
-------------------------------------
|1              |2      |       3|
-------------------------------------
|2              |2      |       1|
-------------------------------------
|3              |3      |       0|
-------------------------------------

Roshnis-MacBook-Air:lab8 roshni$
```

PROGRAM:-

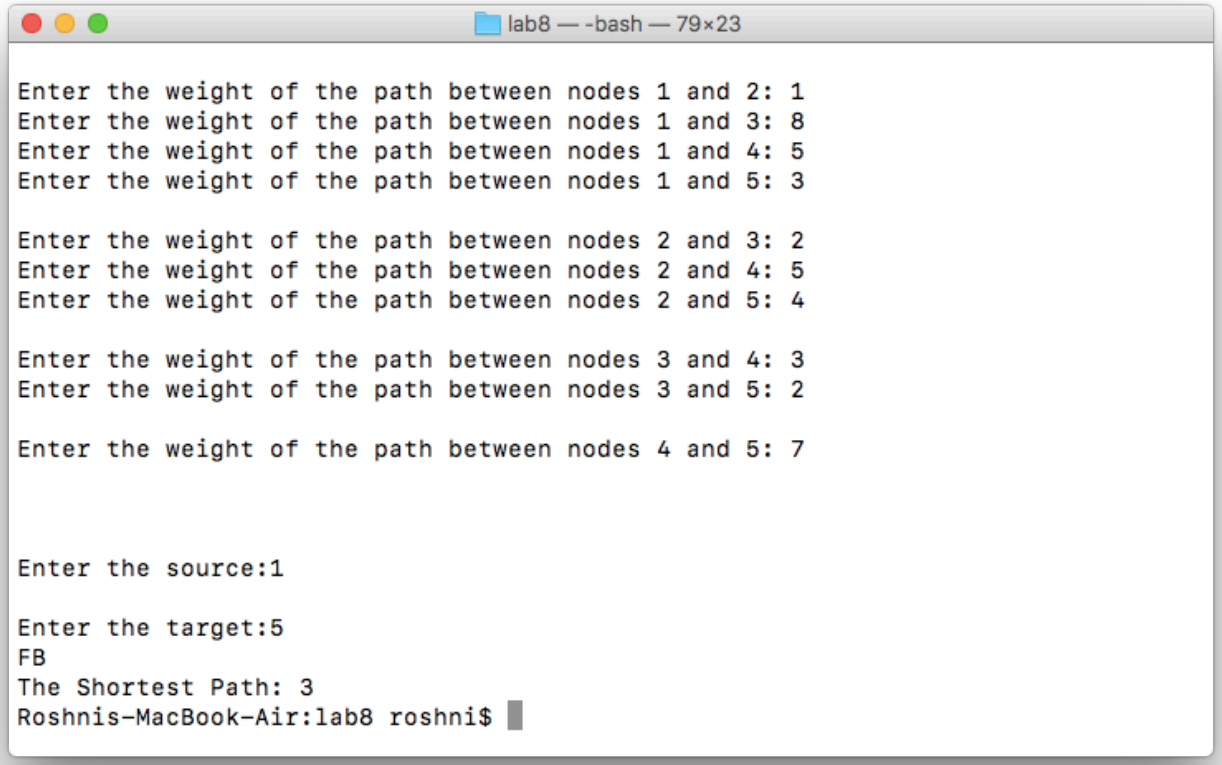
Djikstra.c

```
#include<stdio.h>
//#include<process.h>
#include<string.h>
#include<math.h>
#define IN 99
#define N 6
int dijkstra(int cost[][N], int source, int target);
int main()
{
	int cost[N][N],i,j,w,ch,co;
	int source, target,x,y;
	printf("\t The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C
\n\n");
	for(i=1;i< N;i++)
		for(j=1;j< N;j++)
			cost[i][j] = IN;
	for(x=1;x< N;x++)
	{
		for(y=x+1;y< N;y++)
		{
			printf("Enter the weight of the path between nodes %d and %d: ",x,y);
			scanf("%d",&w);
			cost [x][y] = cost[y][x] = w;
		}
		printf("\n");
	}
	printf("\nEnter the source:");
	scanf("%d", &source);
	printf("\nEnter the target:");
	scanf("%d", &target);
	co = dijkstra(cost,source,target);
	printf("\nThe Shortest Path: %d\n",co);
}
int dijkstra(int cost[][N],int source,int target)
{
	int dist[N],prev[N],selected[N]={0},i,m,min,start,d,j;
```

```
char path[N],rev[N];
for(i=1;i< N;i++)
{
	dist[i] = IN;
	prev[i] = -1;
}
start = source;
selected[start]=1;
dist[start] = 0;
while(selected[target] ==0)
{
	min = IN;
	m = 0;
	for(i=1;i< N;i++)
	{
		d = dist[start] +cost[start][i];
		if(d< dist[i]&&selected[i]==0)
		{
			dist[i] = d;
			prev[i] = start;
		}
		if(min>dist[i] && selected[i]==0)
		{
			min = dist[i];
			m = i;
		}
	}
	start = m;
	selected[start] = 1;
}
start = target;
j = 0;
while(start != -1)
{
	path[j++] = start+65;
	start = prev[start];
}
path[j]='\0';
//strrev(path);
for(i=0;i<j;i++)
```

```
{
        rev[i]=path[j-i];
}
printf("%s", path);
return dist[target];  }
```

OUTPUT;-



PROGRAM:-

slid_serv.c

```
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<stdlib.h>
#include<arpa/inet.h>
#define SIZE 4
int main(int argc, char *argv[])
{
	int sfd,s,nsfd,len,i,j,status;
	char str[100],frame[100],temp[100],ack[20];
	struct sockaddr_in sa,ca;
	sfd=socket(AF_INET,SOCK_STREAM,0);
	//window size
	if(sfd<0)
	{
		perror("Error");
		exit(-1);
	}
	bzero(&sa,sizeof(sa));
	sa.sin_family=AF_INET;
	sa.sin_addr.s_addr=htonl(INADDR_ANY);
	sa.sin_port=htons(atoi(argv[1]));
	s=bind(sfd,(struct sockaddr*)&sa,sizeof(sa)); //assign a name to the server
	if(s<0)
	{
		perror("Bind Error");
		exit(-1);
	}
	listen(sfd,5);
	len=sizeof(&ca);
	nsfd=accept(sfd,(struct sockaddr*)&ca,&len); //take a connection request
	printf(" Enter the text : ");
	scanf("%s",&str);
	//read a string to be transmitted
```

```
	i=0;
	while(i<strlen(str))
	{
		memset(frame,0,100);
		strncpy(frame,str+i,SIZE);
		//generate a frame
		write(1," Transmitting Frames: ",23);
		len=strlen(frame);
		for(j=0;j<len;j++)
			{
				sprintf(temp," %d ",j+status);
				strcat(frame,temp);
			}
			write(nsfd,&frame,sizeof(frame)); //Retransmit the error frame
		}
		i+=SIZE;
	}
	write(nsfd,"exit",sizeof("exit"));
	printf("\nExiting.............\n");
	sleep(2);
	close(nsfd);
	close(sfd);
	return 0;
	//End of transmission
}
```

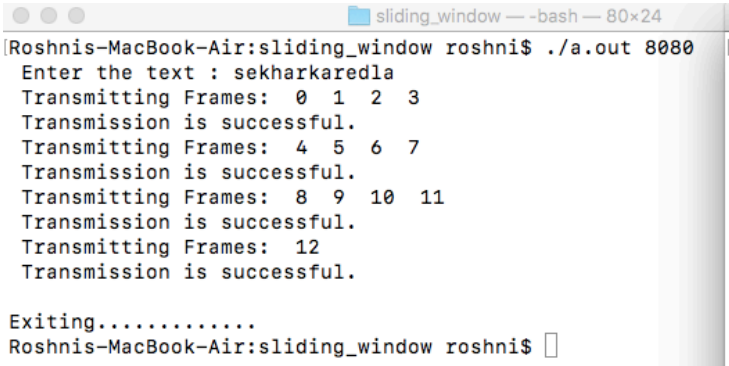
slid_client.c

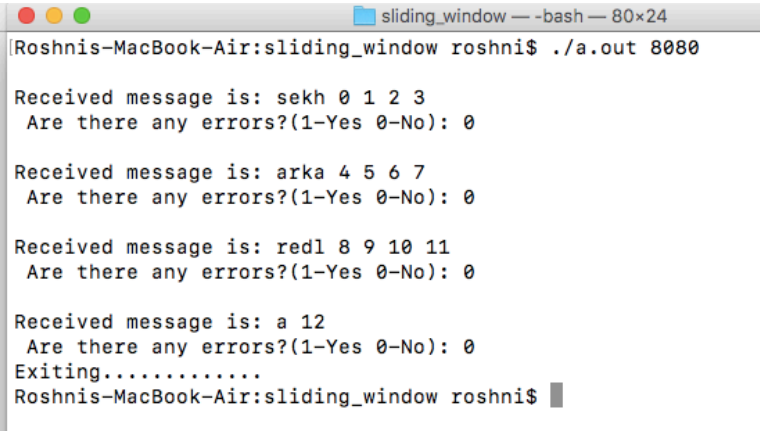
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
int main(int argc, char *argv[])
```

```
{
	int sfd,lfd,len,choice,s,n;
	char str[100],str1[100],err[100];
	struct sockaddr_in saddr,caddr;
		sfd=socket(AF_INET,SOCK_STREAM,0); //create an unnamed TCP
socket
	if(sfd<0)
	{
		perror("FdError");
		exit(-1);
	}
	bzero(&saddr,sizeof(saddr));
	saddr.sin_family=AF_INET;
	//initialize the server address buffer
	saddr.sin_addr.s_addr=inet_addr("127.0.0.1");
	saddr.sin_port=htons(atoi(argv[1]));
	s=connect(sfd,(struct sockaddr*)&saddr,sizeof(saddr));//connect to the sender
	if(s<0)
	{
		perror("connect error");
		exit(-1);
	}
	for(;;)
	{
		n=recv(sfd,&str,100,0);
		//read the frames from the sender
		if(!strncmp(str,"exit",4))
		{
			printf("Exiting.............\n");
			break;
		}
		str[n]='\0';
		printf("\nReceived message is: %s\n Are there any errors?(1-Yes 0-No):
",str);
		scanf("%d",&choice);
		if(!choice)
			write(sfd,”-1",sizeof("-1"));
		 else
		{
			printf("Enter the sequence no of the frame where error has occured: ");
```

```
                scanf("%s",&err);
                write(sfd,&err,sizeof(err));
                n=read(sfd,&str,20);
                str[n]='\0';
                printf("\n\nReceived the re-transmitted frames: %s\n\n",str);
            }
        }
    }
```

OUTPUT:-





PROGRAM:-

ftp_serv.c

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

/*for getting file size using stat()*/
#include<sys/stat.h>

/*for sendfile()*/
#include<sys/sendfile.h>

/*for O_RDONLY*/
#include<fcntl.h>

int main(int argc,char *argv[])
{
	struct sockaddr_in server, client;
	struct stat obj;
	int sock1, sock2;
	char buf[100], command[5], filename[20];
	int k, i, size, len, c;
	int filehandle;
	sock1 = socket(AF_INET, SOCK_STREAM, 0);
	if(sock1 == -1)
	{
		printf("Socket creation failed");
		exit(1);
	}
	server.sin_family=AF_INET;
	server.sin_port = htons(atoi(argv[1]));
	server.sin_addr.s_addr = htonl(INADDR_ANY);
	k = bind(sock1,(struct sockaddr*)&server,sizeof(server));
	if(k == -1)
	{
		printf("Binding error");
```

```
        exit(1);
}
k = listen(sock1,1);
if(k == -1)
{
        printf("Listen failed");
        exit(1);
}
len = sizeof(client);
sock2 = accept(sock1,(struct sockaddr*)&client, &len);
i = 1;
while(1)
{
        recv(sock2, buf, 100, 0);
        sscanf(buf, "%s", command);
        if(!strcmp(command, "ls"))
        {
                system("ls >temps.txt");
                i = 0;
                stat("temps.txt",&obj);
                size = obj.st_size;
                send(sock2, &size, sizeof(int),0);
                filehandle = open("temps.txt", O_RDONLY);
                sendfile(sock2,filehandle,NULL,size);
        }
        else if(!strcmp(command,"get"))
        {
                sscanf(buf, "%s%s", filename, filename);
                stat(filename, &obj);
                filehandle = open(filename, O_RDONLY);
                size = obj.st_size;
                if(filehandle == -1)
                        size = 0;
                send(sock2, &size, sizeof(int), 0);
                if(size)
                        sendfile(sock2, filehandle, NULL, size);

        }
        else if(!strcmp(command, "put"))
        {
```

```
                int c = 0, len;
                char *f;
                sscanf(buf+strlen(command), "%s", filename);
                recv(sock2, &size, sizeof(int), 0);
                i = 1;
                while(1)
                {
                        filehandle = open(filename, O_CREAT | O_EXCL |
O_WRONLY, 0666);
                        if(filehandle == -1)
                        {
                                sprintf(filename + strlen(filename), "%d", i);
                        }
                        else
                                break;
                         }
                f = malloc(size);
                recv(sock2, f, size, 0);
                c = write(filehandle, f, size);
                close(filehandle);
                send(sock2, &c, sizeof(int), 0);
        }
        else if(!strcmp(command, "pwd"))
        {
                system("pwd>temp.txt");
                i = 0;
                FILE*f = fopen("temp.txt","r");
                while(!feof(f))
                        buf[i++] = fgetc(f);
                buf[i-1] = '\0';
                fclose(f);
                send(sock2, buf, 100, 0);
        }
          else if(!strcmp(command, "cd"))
        {
                if(chdir(buf+3) == 0)
                        c = 1;
                else
                        c = 0;
                send(sock2, &c, sizeof(int), 0);
```

```
            }


            else if(!strcmp(command, "bye") || !strcmp(command, "quit"))
            {
                printf("FTP server quitting..\n");
                i = 1;
                send(sock2, &i, sizeof(int), 0);
                exit(0);
            }
        }
        return 0;
}
```

ftp_client.py

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

/*for getting file size using stat()*/
#include<sys/stat.h>

/*for sendfile()*/
#include<sys/sendfile.h>

/*for O_RDONLY*/
#include<fcntl.h>

int main(int argc,char *argv[])
{
struct sockaddr_in server;
struct stat obj;
int sock;
```

```
int choice;
char buf[100], command[5], filename[20], *f;
int k, size, status;
int filehandle;
sock = socket(AF_INET, SOCK_STREAM, 0);
if(sock == -1)
{
printf("socket creation failed");
exit(1);
}
server.sin_family = AF_INET;
server.sin_port = htons(atoi(argv[1]));
server.sin_addr.s_addr = htonl(INADDR_ANY);
k = connect(sock,(struct sockaddr*)&server, sizeof(server));
if(k == -1)
{
printf("Connect Error");
exit(1);
}
int i = 1;
while(1)
{
printf("Enter a choice:\n1- get\n2- put\n3- pwd\n4- ls\n5- cd\n6- quit\n");
scanf("%d", &choice);
switch(choice)
{
case 1:
printf("Enter filename to get: ");
scanf("%s", filename);
strcpy(buf, "get ");
strcat(buf, filename);
send(sock, buf, 100, 0);
recv(sock, &size, sizeof(int), 0);
if(!size)
{
printf("No such file on the remote directory\n\n");
break;
}
f = malloc(size);
recv(sock, f, size, 0);
```

```
while(1)
{
filehandle = open(filename, O_CREAT | O_EXCL | O_WRONLY, 0666);
if(filehandle == -1)
{
sprintf(filename + strlen(filename), "%d", i);//needed only if same directory is used
for both server and client
}
else break;
}
write(filehandle, f, size, 0);
close(filehandle);
strcpy(buf, "cat ");
strcat(buf, filename);
system(buf);
break;
case 2:
printf("Enter filename to put to server: ");
scanf("%s", filename);
filehandle = open(filename, O_RDONLY);
if(filehandle == -1)
{
printf("No such file on the local directory\n\n");
break;
}
strcpy(buf, "put ");
strcat(buf, filename);
send(sock, buf, 100, 0);
stat(filename, &obj);
size = obj.st_size;
send(sock, &size, sizeof(int), 0);
recv(sock, &status, sizeof(int), 0);
if(status)
printf("File stored successfully\n");
else
printf("File failed to be stored to remote machine\n");
break;
case 3:
strcpy(buf, "pwd");
send(sock, buf, 100, 0);
```

```
recv(sock, buf, 100, 0);
printf("The path of the remote directory is: %s\n", buf);
break;
case 4:
strcpy(buf, "ls");
send(sock, buf, 100, 0);
recv(sock, &size, sizeof(int), 0);
f = malloc(size);
recv(sock, f, size, 0);
filehandle = creat("temp.txt", O_WRONLY);
write(filehandle, f, size, 0);
close(filehandle);
printf("The remote directory listing is as follows:\n”);
system("cat temp.txt");
break;
case 5:
strcpy(buf, "cd ");
printf("Enter the path to change the remote directory: ");
scanf("%s", buf + 3);
send(sock, buf, 100, 0);
recv(sock, &status, sizeof(int), 0);
if(status)
printf("Remote directory successfully changed\n");
else
printf("Remote directory failed to change\n");
break;
case 6:
strcpy(buf, "quit");
send(sock, buf, 100, 0);
recv(sock, &status, 100, 0);
if(status)
{
printf("Server closed\nQuitting..\n");
exit(0);
}
printf("Server failed to close connection\n”);
}
}
}
```

OUTPUT:-

ftp — sekhar@debian: ~/cnlab — ssh sekhar(

```
sekhar@debian:~/cnlab$ gcc ftp_serv.c
sekhar@debian:~/cnlab$ sudo ./a.out 2222
FTP server quitting..
sekhar@debian:~/cnlab$
```

ftp — sekhar@debian: ~/cnlab — ssh sekhar@192.168.0.106 — 79×23

```
                else if(!strcmp(command, "bye") || !strcmp(command, "quit"))
                {
                        printf("FTP server quitting..\n");
                        i = 1;
                        send(sock2, &i, sizeof(int), 0);
                        exit(0);
                }
        }
        return 0;
}
Enter a choice:
1- get
2- put
3- pwd
4- ls
5- cd
6- quit
6
Server closed
Quitting..
sekhar@debian:~/cnlab$
```

PROGRAM:-

ping_prog.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>
#define PACKETSIZE 64
struct packet
{
	struct icmphdr hdr;
	char msg[PACKETSIZE - sizeof(struct icmphdr)];
};
int pid=-1;
struct protoent *proto=NULL;
/*--------------------------------------------------------------------*/
/*---checksum-standard 1s complement checksum ---*/
/*--------------------------------------------------------------------*/
unsigned short checksum (void *b, int len)
{
	unsigned short *buf = b;
	unsigned int sum=0;
	unsigned short result;
	for ( sum = 0; len > 1; len -= 2 )
	sum += *buf++;
	if ( len == 1 )
		sum += *(unsigned char*)buf;
		sum = (sum >> 16) + (sum & 0xFFFF);
		sum += (sum >> 16);
	result = ~sum;
	return result;
}
/*--------------------------------------------------------------------*/
/*---display-present echo info---*/
```

```
/*-----------------------------------------------------------------*/
void display(void *buf, int bytes)
{
	int i;
	struct iphdr *ip = buf;
	struct icmphdr *icmp = buf+ip->ihl*4;
	printf("----------------\n");
	for ( i = 0; i < bytes; i++ )
	{
		if ( !(i & 15) )
		printf("\nX:  ", i);
		printf("1X ", ((unsigned char*)buf)[i]);
	}
	printf("\=n");
	printf("IPv%d: hdr-size=%d pkt-size=%d protocol=%d TTL=%d src=%s ",ip-
>version, ip->ihl*4, ntohs(ip->tot_len), ip->protocol,ip->ttl,inet_ntoa(ip->saddr));
	printf("dst=%s\n", inet_ntoa(ip->daddr));
	if ( icmp->un.echo.id == pid )
	{
		printf("ICMP: type[%d/%d] checksum[%d] id[%d] seq[%d]\n",icmp-
>type, icmp->code, ntohs(icmp->checksum),icmp->un.echo.id,
icmp>un.echo.sequence);
	}
}
/*-----------------------------------------------------------------*/
/*---listener-separate process to listen for and collect messages--*/
/*-----------------------------------------------------------------*/
void listener(void)
{
	int sd;
	struct sockaddr_in addr;
	unsigned char buf[1024];
	sd = socket(PF_INET, SOCK_RAW, proto->p_proto);
	if ( sd < 0 )
	{
		perror("socket");
		exit(0);
	}
	for (;;)
	{
```

```
		int bytes, len=sizeof(addr);
		bzero(buf, sizeof(buf));
		bytes = recvfrom(sd, buf, sizeof(buf), 0, (struct sockaddr*)&addr, &len);
		if ( bytes > 0 )
			display(buf, bytes);
		else
			perror("recvfrom");
	}
	exit(0);
}
/*-----------------------------------------------------------------*/
/*---ping-Create message and send it.---*/
/*-----------------------------------------------------------------*/
void ping(struct sockaddr_in *addr)
{
	const int val=255;
	int i, sd, cnt=1;
	struct packet pckt;
	struct sockaddr_in r_addr;
	sd = socket(PF_INET, SOCK_RAW, proto->p_proto);
	if ( sd < 0 )
	{
		perror("socket");
		return;
	}
	if ( setsockopt(sd, SOL_IP, IP_TTL, &val, sizeof(val)) != 0)
		perror("Set TTL option");
	if ( fcntl(sd, F_SETFL, O_NONBLOCK) != 0 )
		perror("Request nonblocking I/O");
	for (;;)
	{
		int len=sizeof(r_addr);
		printf("Msg #%d\n", cnt);
		if ( recvfrom(sd, &pckt, sizeof(pckt), 0, (struct sockaddr*)&r_addr, &len) >
0 )
		printf("***Got message!***\n");
		bzero(&pckt, sizeof(pckt));
		pckt.hdr.type = ICMP_ECHO;
		pckt.hdr.un.echo.id = pid;
		for ( i = 0; i < sizeof(pckt.msg)-1; i++ )
```

```
                pckt.msg[i] = i+'0';
                pckt.msg[i] = 0;
                pckt.hdr.un.echo.sequence = cnt++;
                pckt.hdr.checksum = checksum(&pckt, sizeof(pckt));
            if ( sendto(sd, &pckt, sizeof(pckt), 0, (struct sockaddr*)addr, sizeof(*addr))
<= 0 )
                perror("sendto");
                sleep(1);
        }
}
/*-----------------------------------------------------------------*/
/*---main-look up host and start ping processes.---*/
/*-----------------------------------------------------------------*/
int main(int count, char *strings[])
{
        struct hostent *hname;
        struct sockaddr_in addr;
        if ( count != 2 )
        {
            printf("usage: %s <addr>\n", strings[0]);
            exit(0);
        }
        if ( count > 1 )
        {
            pid = getpid();
            proto = getprotobyname("ICMP");
            hname = gethostbyname(strings[1]);
            bzero(&addr, sizeof(addr));
            addr.sin_family = hname->h_addrtype;
            addr.sin_port = 0;
            addr.sin_addr.s_addr = *(long*)hname->h_addr;
            if ( fork() == 0 )
                listener();
            else
                ping(&addr);
            wait(0);
        }
        else
            printf("usage: myping <hostname>\n");
        return 0;   }
```

OUTPUT:-

```
ftp — sekhar@debian: ~/cnlab — ssh sekhar@192.168.0.106 — 79×23
sekhar@debian:~/cnlab$ sudo ./a.out www.cbit.ac.in
Msg #1
----------------

X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X =nIPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=55 src=202.65.14
1.231 dst=192.168.0.106
ICMP: type[0/0] checksum[7147] id[9127] seq[1]
Msg #2
***Got message!***
----------------

X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X 1X
X:  1X 1X 1X 1X =nIPv4: hdr-size=20 pkt-size=84 protocol=1 TTL=55 src=202.65.14
1.231 dst=192.168.0.106
```