VIDYAA VIKAS COLLEGE OF ENGINEERING & TECHNOLOGY

TIRUCHENGODE – 637 214

DEPARTMENT OF MASTER OF COMPUTER APPLICATION

RECORD NOTEBOOK

NAM E : ……

REGISTER NUMBER : ……

SUBJECT CODE : ……

SUBJECT : ……

VIDYAA VIKAS COLLEGE OF ENGINEERING & TECHNOLOGY

TIRUCHENGODE – 637 214

DEPARTMENT OF MASTER OF COMPUTER APPLICATION

Certified that this the bonafide record of the work done by selven/selvi …….. With register number …… of **II Year/IV Semester** during the academic year 2016-2017 **MC7412 NETWORK PROGRAMMING LABORATORY.**

Submitted for the university practical examination held on ……

Staff-in-charge **Head of the Department**

Internal Examiner **External Examiner**

INDEX

S.NO	DATE	NAME OF THE PROGRAM	PAGE NO	MARKS	SIGNATURE
1		Implementation of File System Calls			
2		Implementation of ICP Techniques-Pipe			
3		Implementation of ICP Techniques-MessageQueue			
4		Implementation of ICP Techniques-Shared Memory			
5		Socket Programming using TCP Sockets			
6		Socket Programming using UDP Sockets			
7		Application using Socket (FTP)			
8		Simulation of Sliding Window Protocol			
9		Simulation of Routing Protocol			
10		Remote Procedure Call(RPC)			
11		Domain Name System(DNS)			
12		Hyper Text Transfer Protocol(HTTP)			
13		Electronic Mail			
14		Multi- User Chat			

EX.NO : 1 DATE :	Implementation of File System Calls

AIM

To develop a File System Calls using ‘ C’ Language in Unix.

ALGORITHM

Step1: Start the program.

Step2: Include the necessary header file to execute a system calls .

Step3: Declare the necessary data variable with their data type .

Step4: Declare the buffer as character with size of 20.

Step5: if(S_ISREG(buf.st_mode)) then print the true or false statement.

Step6: Create and open a text file and named as “jm.txt”.

Step7: Save and execute the program.

Step8: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<time.h>
int main()
{
	int s,fd1,fd2,len;
	struct stat buf;
	char buff[20],buff1[20];
	system("clear");
	fd1=open("jm1.txt",O_RDONLY);
	s=stat("jm1.txt",&buf);
	if(S_ISREG(buf.st_mode))
	{
		printf("\n\n Regular File\n");
	}
	else
	{
		printf("\n\n Irregular File\n");
	}
	printf("File Size is:\n",buf.st_size);
	printf("File User id is:%d\n",buf.st_uid);
```

```
        printf("File Last Accesses Time:%s\n",ctime(&buf.st_atime));
        fd2=creat("jm.txt",S_IRUSR|S_IWUSR|S_IXUSR);
        fd2=open("jm.txt",O_WRONLY);
        if(fd1==-1)
        {
        }
        while((s=read(fd1,buff,1))>0)
        {
              buff1[10]=toupper(buff[0]);
              write(fd2,buff1,1);
        }
        len=lseek(fd2,0,SEEK_END);
        lseek(fd2,0,SEEK_SET);
        read(fd2,buff,len);
        printf("Buffer Information%s\n",buff);
        printf("The Length is%d\n",len);
        close(fd1);
        close(fd2);
        return 0;
}
```

OUTPUT

```
[vvmca26@localhost ~]$ cc call.c

[vvmca26@localhost ~]$ ./a.out call

Regular File

File Sizeis:44

File User id is:552

File Last Access Time: wed Apr  6 04:20:36 2016

Buffer Inforamtion

The Length is 44

[vvmca26@localhost ~]$ jm1.txt

I,This Is Sample Program For SystemCalls
```

RESULT

Thus the C program for implementation of file system calls has been executed successfully.

EX.NO : 2 DATE :	Implementation of ICP Techniques – Pipe

AIM

To implement a ICP techniques of pipe using ‘ C’ Language in Unix.

ALGORITHM

Step1: Start the program.

Step2: Define the parent and child function and pass the parameter (int,int).

Step3: In main program we declare the pipe using array size of 2.

Step4: Check the if condition if((pid=fork())=0).

Step5: In parent class pass the arguments (int rfd,int wfd).

Step6: Declare the variable named as len and buff.

Step7: In child class pass the arguments (int wfd,int rfd).

Step8: Save and run the program.

Step9: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<unistd.h>
void parent(int,int);
void child(int,int);
int main()
{
	int pipe1[2],pipe2[2];
	pid_t pid;
	pipe(pipe1);
	pipe(pipe2);
	if((pid=fork())==0)
	{
		close(pipe1[1]);
		close(pipe2[0]);
		child(pipe1[1],pipe2[0]);
	}
	sleep(5);
	close(pipe1[0]);
	close(pipe2[1]);
	parent(pipe1[1],pipe2[0]);
}
```

```
void parent(int rfd,int wfd)
{
	int len;
	char buff[80];
	printf("\n Enter mesage for child:");
	scanf("%s",buff);
	len=strlen(buff);
	if(buff[len-1]=='\n')
		len--;
	write(wfd,buff,len);
	read(rfd,buff,80);
	printf("\n Parent Process");
	printf("\n ~~~~~~~~~~~~~~~");
	printf("\n Message from Parent:%s\n",buff);
}
void child(int wfd,int rfd)
{
	int len;
	char buff[80];
	printf("\n Enter Message for Parent:");
	scanf("%s",buff);
	len=strlen(buff);
	if(buff[len-1]=='\n')
		len--;
```

```
        write(wfd,buff,len);
        read(rfd,buff,80);
        printf("\n Child Process:");
        printf("\n ~~~~~~~~~~~~~~~");
        printf("\n Message from Parent:%s\n",buff);
}
```

OUTPUT

```
[vvmca26@localhost ~]$ cc pipe1.c
[vvmca26@localhost ~]$ ./a.out pipe1.c
 Enter Message for Parent:hi
 Child Process:
 ~~~~~~~~~~~~~~~
 Message from Parent:hi
 Enter mesage for child:welcome
 Parent Process
 ~~~~~~~~~~~~~~~
 Message from Parent:welcome
```

RESULT

Thus the C program for implementation of ICP techniques using pipe has been executed successfully.

EX.NO : 3 DATE :	Implementation of ICP Techniques – Message Queue

AIM

To implement a ICP techniques for Message Queue using ‘ C’ Language in Unix.

ALGORITHM

Step1: Start the program.

Step2: Include the necessary header files for message queue.

Step3: In main program declare the data variable with require data member.

Step4: Declare the message queueid for get a message from IPC_CREAT.

Step5: Check the for(i=0;i<n;i++)

Step6: Save and run a program.

Step7: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<string.h>
#include<mqueue.h>
#include<sys/ipc.h>
#include<sys/msg.h>
int main()
{
    int msqid,pid,len[20],i,n;
    struct msqid_ds qstatus;
    char str[15];
    struct
    {
        int mtype;
        char mtext[100];
    }
    buff;
    fflush(stdin);
    printf("\n Enter the number of message:");
    scanf("%d",&n);
    msqid=msgget(0,IPC_CREAT|0666);
    printf("\n Message queue create with key:%d \n",msqid);
```

```
    for(i=0;i<n;i++)
    {
        printf("\n Enter message for queue\n");
        scanf("%s",str);
        strcat(buff.mtext,str);
        buff.mtype=1;
        len[i]-strlen(buff.mtext);
        msgsnd(msqid,&buff,len[i],0);
        msgctl(msqid,IPC_STAT,&qstatus);
        printf("\n Message queue id:%d\n",msqid);
        printf("\nMessage on the queue:\n",qstatus.msg_qnum);
    }
    printf("\nReading Messages:\n");
    msgrcv(msqid,&buff,len[1],0,0);
    printf("\n Messages are:%s\n",buff.mtext);
    msgctl(msqid,IPC_RMID,&qstatus);
}
```

OUTPUT

```
[vvmca25@localhost ~]$ vi que.c
[vvmca25@localhost ~]$ cc que.c
[vvmca25@localhost ~]$ ./a.out
Enter the no of msg:3
Message queue create with key:131072
Enter Message for queue
one
Message queue id:131072
Messageon the queue:
Enter Message for queue
 two
Message queue id:131072
Messageon the queue:
Enter Message for queue
three
Message queue id:131072
Messageon the queue:
Reading Messages:
Message are:one two three
```

RESULT

Thus the C program for implementation of ICP techniques using message queue has been executed successfully.

EX.NO : 4 DATE :	Implementation of ICP Techniques – Shared Memory

AIM

To implement a ICP techniques for Shared Memory using ‘ C’ Language in Unix.

ALGORITHM

Step1: Start the program.

Step2: Define the NUM_ELEM , SEM_EMPTY and SEM_FULL.

Step3: In main program declare the date member with data type.

Step4: Declare the union semun with their member function.

Step5: Check the if(pid==0)

Step6: Check the for(i=0;i<26;i++)

Step7: Save and run the program.

Step8: Print the alphabet letters in sequence order.

Step9: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/shm.h>
#define NUM_ELEM 10
#define SEM_EMPTY 0
#define SEM_FULL 1
main()
{
        int rc;
        int pid;
        int semid;
        int shmid;
          int status;
        int i;
        char elem;
        union semun
        {
                int val;
                struct semid_ds *buf;
                ushort *array;
        }mysemun;
        struct sembuf waitempty={SEM_EMPTY,-1,SEM_UNDO};
        struct sembuf signalempty={SEM_EMPTY,1,IPC_NOWAIT};
        struct sembuf waitfull={SEM_FULL,-1,SEM_UNDO};
        struct sembuf signalfull={SEM_FULL,1,IPC_NOWAIT};
        struct shmid_ds myshmid_ds;
        void *shmptr;
        semid=semget(IPC_PRIVATE,2,0666|IPC_CREAT);
        mysemun.val=NUM_ELEM;
        semctl(semid,SEM_EMPTY,SETVAL,mysemun);
        mysemun.val=0;
        semctl(semid,SEM_FULL,SETVAL,mysemun);
        shmid=shmget(IPC_PRIVATE,NUM_ELEM,0666|IPC_CREAT);
        pid=fork();
        if(pid==0)
        {
                shmptr=shmat(shmid,0,SHM_R);
                for(i=0;i<26;i++)
                {
                        semop(semid,&waitfull,1);
                        elem = *((char *)shmptr+(i%NUM_ELEM));
```

```
                printf("\t\t\t Consumed elem '%c'\n",elem);
                semop(semid,&signalempty,1);
            }
        }
        else
        {
            shmptr = shmat(shmid,0,SHM_W);
            for(i=0;I       <26;i++)
            {
                semop(semid,&waitempty,1);
                elem='a'+i;
                printf("produced elem '%c'\n",elem);
                *((char *)shmptr+(i%NUM_ELEM))=elem;
                if(pid==        0)
                {
                    shmptr=shmat(shmid,0,SHM_R);
                    for(i=0;i<26;i++)
                    {
                        semop(semid,&waitfull,1);
                        elem=*((char *)shmptr+(i%NUM_ELEM));
                        printf("Consumed elem '%c'\n",elem);
                        semop(semid,&signalempty,1);
                    }
                }
                else
                {
                    shmptr = shmat(shmid,0,SHM_W);
                    for(i=0;i<26;i++)
                    {
                        semop(semid,&waitempty,1);
                        elem='a'+i;
                        printf("Produced elem '%c'\n",elem);
                        *((char *)shmptr+(i%NUM_ELEM))=elem;
                        semop(semid,&signalfull,1);
                    }
                }
            }
        }
}
```

OUTPUT

```
[anitha@AntiViruS ~]$ cc sharmem.c
[anitha@AntiViruS ~]$ ./a.out
produced elem 'a'
Produced elem 'a'
Produced elem 'b'
Produced elem 'c'
Produced elem 'd'
Produced elem 'e'
Produced elem 'f'
Produced elem 'g'
Produced elem 'h'
Produced elem 'i'
                Consumed elem 'a'
                Consumed elem 'b'
                Consumed elem 'c'
                Consumed elem 'd'
                Consumed elem 'e'
                Consumed elem 'f'
                Consumed elem 'g'
                Consumed elem 'h'
                Consumed elem 'i'
Produced elem 'j'
Produced elem 'k'
Produced elem 'l'
Produced elem 'm'
Produced elem 'n'
Produced elem 'o'
Produced elem 'p'
Produced elem 'q'
Produced elem 'r'
                Consumed elem 'j'
                Consumed elem 'k'
                Consumed elem 'l'
                Consumed elem 'm'
                Consumed elem 'n'
                Consumed elem 'o'
                Consumed elem 'p'
                Consumed elem 'q'
                Consumed elem 'r'
Produced elem 's'
Produced elem 't'
Produced elem 'u'
Produced elem 'v'
Produced elem 'w'
```

```
Produced elem 'x'
Produced elem 'y'
Produced elem 'z'
```

RESULT

Thus the C program for implementation of ICP techniques using shared memory has been executed successfully.

EX.NO : 5 DATE :	Socket Programming using TCP Sockets

AIM

To develop a Socket Program for TCP Sockets using 'C' Language in Unix.

ALGORITHM

CLIENT

Step1: Start the program.

Step2: Define the value of MAXLINE, PORTNO, structure of socket.

Step3: In main program we pass 2 argument as argc, argv.

Step4: Declare the sockfd, n as an integer variables and recvline as character.

Step5: if(argc!=2) thenPrint the usage tcpcli<IPaddress>

Step6: Assign the family of socket, port no and address of a socket.

Step7: Make a connection to the server & determine the length of the client address.

Step8: Read the value of the client, which sends to the server.

Step9: Stop the program.

SERVER

Step1: Start the program.

Step2: Define LISTENQ, MAXLINE, PORT, SOCKET structure globally.

Step3: Declare listenfd, connfd as an integer variables. Len as a socklen_t buffer as a character.

Step4: Assign the value of listenfd by calling the socket() function.

Step5: Define the family of socket, port no, client address from which port.

Step6: Calling the bind() and listen() function

Step7: for(; ;)

- **7.1:** Assign the length of clientaddr.
- **7.2:** Calling the accept() function and assign it in connfd.
- **7.3:** Assign the time and Close the connfd.

Step8: Stop the program.

PROGRAM (TCP SERVER)

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<string.h>
main()
{
        int sockint,namelen,ns;
        int sd,s;
        struct sockaddr_in cli,server;
        char buf[32]="hello";
        s = socket(AF_INET,SOCK_STREAM,0);
        server.sin_family=AF_INET;
        server.sin_port=0;
        server.sin_addr.s_addr=INADDR_ANY;
        bind(s,(struct sockaddr *)&server,sizeof(server));
        namelen=sizeof(server);
        getsockname(s,(struct sockaddr *)&server,&namelen);
        fprintf(stderr,"The assigned port is %d \n",htons(server.sin_port));
        listen(s,1);
        namelen=sizeof(cli);
        ns=accept(s,(struct sockaddr *)&cli,&namelen);
        recv(ns,buf,sizeof(buf),0);
        send(ns,buf,sizeof(buf),0);
```

```
        close(ns);
        close(s);
        printf("%s server finishes",buf);
}
```

PROGRAM (TCP CLIENT)

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<string.h>
main(int argc,char *argv[])
{
        int sockint,s;
        unsigned short port;
        struct sockaddr_in server;
        char buf[32];
        s=socket(AF_INET,SOCK_STREAM,0);
        port=(unsigned short)atoi(argv[2]);
        strcpy(buf,"Hello");
        fprintf(stderr,"Creating datagram packet \n");
        s=socket(AF_INET,SOCK_STREAM,0);
        server.sin_family=AF_INET;
        server.sin_port=htons(port);
        server.sin_addr.s_addr=inet_addr(argv[1]);
        connect(3,(struct sockaddr *)&server,sizeof(server));
        printf("Sending the message:%s\n",buf);
        send(s,buf,sizeof(buf),0);
        printf("msg send to server \n");
        recv(s,buf,sizeof(buf),0);
        printf("The message from the server %s \n",buf);
```

```
        close(5);
        printf("Client closed successfully");
}
```

OUTPUT (SERVER)

[anitha@AntiViruS ~]$ cc tcpks.c

[anitha@AntiViruS ~]$ ./a.out

The assigned port is 33843

hello server finishes[anitha@AntiViruS ~]$

OUTPUT (CLIENT)

[anitha@AntiViruS ~]$ cc tcpkc.c

[anitha@AntiViruS ~]$ ./a.out 172.16.1.5 33843

Creating datagram packet

Sending the message:Hello

RESULT

Thus the C program for socket programming using TCP socket has been executed successfully.

EX.NO : 6 DATE :	Socket Programming using UDP Sockets

AIM

To write a 'C' program for UDP echo client server chat.

ALGORITHM

SERVER

Step1: Start the program.

Step2: Define the value of port no, MAXLINE.

Step3: In sub function ,declare on as an integer, len in socklen_t as a character variables.

Step4: for(; ;)

4.1: Assign the client length.

4.2: call the recvfrom and send to functions.

Step5: In main, Declare sockfd as an integer and declare the servaddr structure & cliaddr structure

Step6: Call the socket() function and design (assign) its value to the variable sockfd.

Step7: Assign the socket family, port no, client address.

Step8: Call the bind() and accept() functions.

Step9: Stop the program.

CLIENT

Step1: Start the program.

Step2: Define the value of PORT NO, MAXLINE.

Step3: In dg_cli function call recvfrom() functions.

Step4: Assign the value of recv[n]=0 and print the recv & stdout.

Step5: In main, declare sockfd as integer and declare the servaddr structure.

Step6: Define the family of socket, port no and call the socket() function & assign it into sockfd.

Step7: Stop the program.

PROGRAM (UDP SERVER)

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
main()
{
        struct sockaddr_in sadd,cadd;
        int id,a,b,len,port;
        char rbuff[100];
        id=socket(PF_INET,SOCK_DGRAM,0);
        if(id<0)
                printf("Can't Create\n");
        else
                printf("Created\n");
                printf("Enter the port Address\n");
                printf("_____\n");
                scanf("%d",&port);
                sadd.sin_family=PF_INET;
                sadd.sin_addr.s_addr=htonl(INADDR_ANY);
                sadd.sin_port=htons(port);
                b=bind(id,(struct sockaddr*)&sadd,sizeof(sadd));
        if(b<0)
                printf("Can't Bind");
        else
                printf("Binded\n");
                printf("~~~~~~\n");
                len=sizeof(cadd);
                if(recvfrom(id,rbuff,sizeof(rbuff),0,(struct sockaddr*)&cadd,&len)<0)
                printf("Received Error\n");
                else
                printf("Server received =%s\n",rbuff);
                close(id);
}
```

PROGRAM (UDP CLIENT)

```
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
main()
{
        struct sockaddr_in sadd,cadd;
        int id,len,n,c,s,b,port;

        char str[100],serstr[100];
        id=socket(PF_INET,SOCK_DGRAM,0);
        if(id<0)
                printf("Can't Create\n");
        else
                printf("Socket is Created\n");
                printf("Enter the IP address\n");
                scanf("%s",serstr);
                printf("Enter the port Address\n");
                scanf("%d",&port);
                cadd.sin_family=PF_INET;
                cadd.sin_addr.s_addr=inet_addr(serstr);
                cadd.sin_port=htons(port);
                printf("Enter the Data\n");
                scanf("%s",str);
                b=bind(id,(struct sockaddr*)&cadd,sizeof(cadd));
        if(sendto(id,str,sizeof(str),0,(struct sockaddr*)&cadd,sizeof(cadd))<0)
                printf("Transmit Error");
        else
                printf("Server Transmitted=%s\n",str);
                close(id);
}
```

OUTPUT (SERVER)

[mit11mca20@mahendralinux ~]$ cc udpserver.c

[mit11mca20@mahendralinux ~]$ ./a.out

Created

Enter the port Address

_____

9876

Binded

OUTPUT (CLIENT)

[mit11mca20@mahendralinux ~]$ cc udpclient.c

[mit11mca20@mahendralinux ~]$ ./a.out

Socket is Created

Enter the IP address

10.0.24.18

Enter the port Address

9876

Enter the Data

helloworld

Server transmitted = helloworld

RESULT

Thus the C program for socket programmming using UDP socket has been executed successfully.

EX.NO : 7 DATE :	Application using Socket(FTP)

AIM

To write a C program for transfer ring a file using TCP.

ALGORITHM

SERVER

Step 1: Start the program.

Step 2: Create an unnamed socket for the server using parameters AF_INET asdomain and SOCK_STREAM as type.

Step 3: Get the server port number.

S**tep 4:**Register the host address to the system by using bind() system call in server side.

Step 5: Create a connection queue and wait for clients using listen() system call with the Number of clients requests as parameter.

Step 6: Create a Child process using fork( ) system call.

Step 7: If the process identification number is equal to zero accept the connection using accept( ) system call when the client request for connection.

Step 8: If pid is not equal to zero then exit the process.

Step 9: Stop the Program execution.

CLIENT

Step 1: Start the program.

Step 2: Create an unnamed socket for the client using parameters AF_INET as domain and SOCK_STREAM as type.

Step 3: Get the client port number.

Step 4: Now connect the socket to server using connect( ) system call.

Step 5: Enter the file name.

Step 6: The file is transferred from client to server using send ( ) function.

Step 7: Print the contents of the file in a new file.

Step 8: Stop the program.

PROGRAM (SERVER)

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
main()
{
        FILE *fp;
        int sd,newsd,ser,n,a,cli,pid,bd,port,clilen;
        char name[100],fileread[100],fname[100],ch,file[100],rcv[100];
        struct sockaddr_in servaddr,cliaddr;
        printf("Enter the port address: ");
        scanf("%d",&port);
        sd=socket(AF_INET,SOCK_STREAM,0);
        if(sd<0)
                printf("Can't Create \n");
        else
                printf("Socket is Created\n");
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
        servaddr.sin_port=htons(port);
        a=sizeof(servaddr);
        bd=bind(sd,(struct sockaddr*)&servaddr,a);
        if(bd<0)
                printf(" Can't Bind\n");
        else
                printf("\n Binded\n");
        listen(sd,5);
        clilen=sizeof(cliaddr);
        newsd=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
        if(newsd<0)
                printf("Can't Accept\n");
        else
                printf("Accepted\n");
        n=recv(newsd,rcv,100,0);
        rcv[n]='\0';
        fp=fopen(rcv,"r");
        if(fp==NULL)
        {
                 send(newsd,"error",5,0);
                close(newsd);
        }
        else
        {
                while(fgets(fileread,sizeof(fileread),fp))
```

```
                {
                        if(send(newsd,fileread,sizeof(fileread),0)<0)
                        {
                                printf("Can't send\n");
                        }
                        sleep(1);
                }
                if(!fgets( fileread,sizeof(fileread),fp))
                {
                        send(newsd,"completed",999999999,0);
                }
                return(0);
        }
}
```

PROGRAM (CLIENT)

```
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
main()
{
	FILE *fp;
	int csd,n,ser,s,cli,cport,newsd;
	char name[100],rcvmsg[100],rcvg[100],fname[100];
	struct sockaddr_in servaddr;
	printf("Enter the port");
	scanf("%d",&cport);
	csd=socket(AF_INET,SOCK_STREAM,0);
	if(csd<0)
	{
		printf("Error...");
	}
	else
		printf("Socket is Created...\n");
	 servaddr.sin_family=AF_INET;
	servaddr.sin_addr.s_addr=htonl(INADDR_ANY);servaddr.sin_port=htons(c
	port);
	if(connect(csd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
		printf("Error in Connection...\n");
	else
		printf("Connected...\n");
	printf("Enter the existing file name: ");
	scanf("%s",name);
	printf("\nEnter the new filename: ");
	scanf("%s",fname);
	fp=fopen(fname,"w");
	send(csd,name,sizeof(name),0);
	while(1)
	{
		s=recv(csd,rcvg,100,0);
		rcvg[s]='\0';
		if(strcmp(rcvg,"error ")==0)
			printf("File is not Available...\n");
```

```
                if(strcmp(rcvg,"completed")==0)
                {
                        printf("file is transferred...\n");
                        fclose(fp);
                        close(csd);
                        break;
                }
                else
                        fputs(rcvg,stdout);
                        fprintf(fp,"%s",rcvg);
        }
}
```

OUTPUT (SERVER)

```
[1me16@localhost ~]$ cc ftpclient.c
[1me16@localhost ~]$. /a.out
Enter the port address:
8663
Socket is Created
Binded
Connected…
```

OUTPUT (CLIENT)

```
[1me16@localhost ~]$ cc ftpserver.c
[1me16@localhost ~]$. /a.out
Socket is Created..
Connected
Enter the existing file name: net
Enter the new file name: network
Welcome to Network Lab
File is transferred...
```

RESULT

Thus the C program for file transfer protocol program has been executed successfully.

EX.NO : 8 DATE :	Simulation of Sliding Window Protocol

AIM

To implement the Sliding Window Protocol in Unix using 'C' language.

ALGORITHM

Step 1: Create a socket using Create() at the client side.

Step 2: Bind the socket using Bind() to the server.

Step 3: Enter the message in Client side that is to be sent to the server.

Step 4: At the server side, receive the frames and send acknowledgement for the next frame.

Step 4: Receive the acknowledgement at the client side and send the respective frame.

Step 5: Repeat steps 3 to 5 for the entire transmission.

Step 6: Stop the Program.

PROGRAM(SERVER)

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<sys/socket.h>
main()
{
        int a,bd,sd,newsd,port,clilen;
        char lost[20],sendmsg[20],recvmsg[20];
        struct sockaddr_in servaddr,cliaddr;
        sd=socket(AF_INET,SOCK_STREAM,0);
        if(sd<0)
                printf("Can't Create \n");
        else
                printf("Socket is Created\n");
                printf("Enter the port no\n");
                scanf("%d",&port);
                servaddr.sin_family=AF_INET;
                servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
                servaddr.sin_port=htons(port);
                a=sizeof(servaddr);
                bd=bind(sd,(struct sockaddr*)&servaddr,a);
        if(bd<0)
                printf(" Can't Bind\n");
        else
                printf("\n Binded\n");
                listen(sd,5);
                clilen=sizeof(cliaddr);
                newsd=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
        if(newsd<0)
                printf("Can't Accept\n");
        else
                printf("Accepted\n");
                printf("Enter the lost frame\n");
                scanf("%s",lost);
                send(newsd,lost,20,0);
                recv(newsd,recvmsg,20,0);
                printf("\n Frame %s is successfully received ",recvmsg);
}
```

PROGRAM (CLIENT)

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
main()
{
        int i,sd,n,port;
        char sendmsg[100],recvmsg[100];
        struct sockaddr_in servaddr;
        printf("Enter the port\n");
        scanf("%d",&port);
        sd=socket(AF_INET,SOCK_STREAM,0);
        if(sd<0)
                printf("Can't Create\n");
        else
                printf("Socket is Created\n");
                servaddr.sin_family=AF_INET;
                servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
                servaddr.sin_port=htons(port);
        if(connect(sd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
                printf("Can't Connect\n");
        else
                printf("Connected\n");
                printf("Enter the no of frames\n");
                scanf("%d",&n);
                printf("\nThe frames all\n");
        for(i=1;i<=n;i++)
                printf("Frame %d\n",i);
                recv(sd,recvmsg,20,0);
                printf("\n Lost frame %s is retransmitted ",recvmsg);
                strcpy(sendmsg,recvmsg);
                send(sd,sendmsg,20,0);
}
```

OUTPUT

SERVER

[mit11mca20@mahendralinux lab]$ cc slidingserver.c

[mit11mca20@mahendralinux lab]$ ./a.out

Enter the port

6541

Socket is Created

Connected

CLIENT

[mit11mca20@mahendralinux lab]$ cc slidingclient.c

[mit11mca20@mahendralinux lab]$ ./a.out

Enter the port

6541

Socket is Created

Connectected

Enter the no of frames : 3

The frames all

Frame 1

Frame 2

Frame 3

Server : Accepted

Enter the lost frame: 3

Frame 3 is successfully received

Client : Lost Frame 3 is retransmitted

RESULT

Thus the C program for sliding window protocol program has been executed successfully.

EX.NO : 9 DATE :	Simulation of Routing Protocols

AIM

To simulate the Open Shortest Path First routing protocol based on the cost assigned to each path.

ALGORITHM

Step 1: Read the number of nodes, n.

Step 2: Read the cost matrix for the path from each node to another node.

Step 3: Initialize SOURCE to 1 and DEST to 1.

Step 4: Compare D of a node which is the distance from source to that corresponding node.

Step 5: Repeat steps 6 to 8 for n-1 nodes.

Step 6: Choose the node that has not be included compare the distance to reach the node using the newlyincluded node.

Step 7: Take the minimum value as the new distance.

Step 8: Print the distance of the given path calculated by the program and the path.

Step 9: Read further source and destination from the user to calculate the shortest path.

Step 10: Exit when the user chooses -1 for source and destination.

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#define INFINITY      0x7FFF
static int adj[20][20];
int dist[20], previous[20], unoptimized[20], n;
void Init();
int UnoptimizedNodeExists();
void FindShortest(int src);
void main()
{
        int wt, stack[20], top=0, src, dst, v;
        //clrscr();
        AskNumberOfNodes:
        printf("Enter the number of nodes (max 20): ");
        scanf("%d", &n);
        if(n>20)
        {
                printf("\nSorry, so many are not allowed. Try again!\n");
                goto AskNumberOfNodes;
        }
        printf("\nEnter the edges one by one. -1, -1 to stop.\n");
        while(1)
        {
                printf("\nSource\t\t: ");
                scanf("%d", &src);
                printf("Destination\t: ");
                scanf("%d", &dst);
                if(src==-1 && dst==-1)
                        break;
                if(src<1 || src>n || dst<1 || dst>n)
                {
                        printf("\nInvalid input\n");
                        continue;
                }
                src--;
                dst--;
                printf("Edge weight\t: ");
                scanf("%d", &wt);
                if(wt<1)
                {
                        printf("\nWeight must be atleast 1! Edge neglected.\n");
                        continue;
                }
                adj[src][dst]=adj[dst][src]=wt;
```

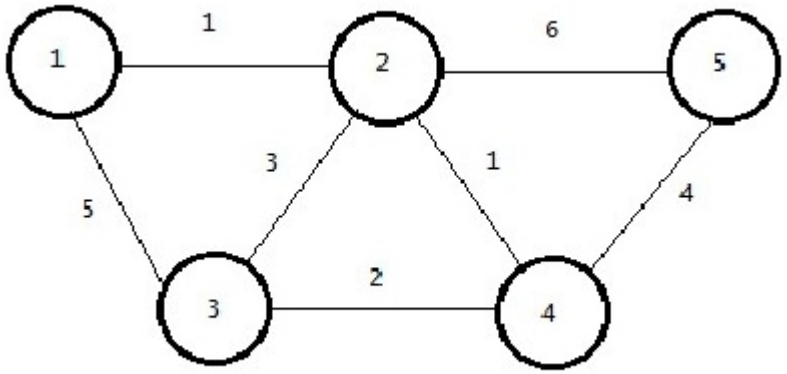
```
	}
	printf("\nTo find shortest path... use -1, -1 to stop.\n");
		while(1)
	{
		printf("\nEnter source and destination nodes: ");
		scanf("%d%d", &src, &dst);
		if(src==-1 && dst==-1)
			break;
		if(src<1 || src>n || dst<1 || dst>n)
		{
			printf("\nInvalid input\n");
			continue;
		}
		src--;
		dst--;
		FindShortest(src);
		v=dst;
		while(v!=(-1))
		{
			stack[top]=v;
			top++;
			v=previous[v];
		}
		if(top==1 && src!=dst)
			printf("\nSorry no such path found!\n");
		else
		{
			printf("\n");
			while(top)
			{
				top--;
				printf("%d - ", stack[top]+1);
			}
			printf("\b\b \n");
		}
	}
}
void Init()
{
	int i;
	for(i=0; i<n; i++)
	{
		previous[i]=(-1);
		dist[i]=INFINITY;
		unoptimized[i]=1;
	}
```

```
}
int UnoptimizedNodeExists()
{
	int i;
	for(i=0; i<n; i++)
	{
		if(unoptimized[i])
			return 1;
	}
	return 0;
}
void FindShortest(int src)
{
	int i, alt, u=src, v, minDistance;

	// Initialize prev[], dist[] and
	// Q := the set of all nodes in Graph ;
	Init();

	// dist_between(source, source) is obviously zero
	dist[src]=0;

	while(UnoptimizedNodeExists())
	{
		minDistance = INFINITY;
		// u := vertex in Q with smallest dist[] ;
		for(i=0; i<n; i++)
		{
			if(unoptimized[i] && dist[i]<minDistance)
			{
				u=i;
				minDistance=dist[i];
			}
		}
		if(minDistance == INFINITY)
		{
			printf("\nOne or more isolated node(s) found!\n");
			break;
		}
		// Remove u from Q
		unoptimized[u]=0;

		// For each neighbor v of u:
		for(v=0; v<n; v++)
		{
			// (where v is not removed from Q)
```

```
                if(adj[u][v] && unoptimized[v])
                {
                        alt = dist[u] + adj[u][v];
                        if(alt<dist[v])
                        {
                                dist[v]=alt;
                                previous[v]=u;
                        }
                }
        }
}
```

OUTPUT

Enter the number of nodes (max 20): 5
Enter the edges one by one. -1, -1 to stop.
Source : 1
Destination : 2
Edge weight : 1
Source : 2
Destination : 5
Edge weight : 6
Source : 2
Destination : 4
Edge weight : 1
Source : 2
Destination : 3
Edge weight : 3
Source : 5
Destination : 4
Edge weight : 4
Source : 3
Destination : 4
Edge weight : 2
Source : 1
Destination : 3
Edge weight : 5
Source : -1
Destination : -1



Graph Structure

To find shortest path... use -1, -1 to stop.
Enter source and destination nodes: 1 5
1 - 2 - 4 - 5
Enter source and destination nodes: 1 4
1 - 2 - 4
Enter source and destination nodes:-1-1

RESULT

Thus the C program for routing protocol program has been executed successfully.

EX.NO : 10 DATE :	Remote Procedure Call(RPC)

AIM

To write a program for addition of two numbers using RPC (Remote Procedure Call).

ALGORITHM

1. Define the remote interface
2.Implement the server
3.Implement the client
4.Compile the source files
5.Start the Java RMI registry, server, and client

The files needed for this program are:

addintf.java - a remote interface
addServer.java - a remote object implementation that implements the remote interface
addClient.java - a simple client that invokes a method of the remote interface
addImpl.java-a simple implementation file.

Implement the server

The implementation class add Server implements the remote interface addIntf, providing an implementation for the remote method. The method does not need to declare that it throws any exception because the method implementation itself does not throw Remote Exception nor does it throw any other checked exceptions.

INTERFACE PROGRAM

```
import java.rmi.*;
public interface AddIntf2 extends Remote
{
        double add(double d1,double d2) throws RemoteException;
}
```

Server Program

```
import java.rmi.*;
import java.net.*;
public class server
{
public static void main(String arg[])throws RemoteException
{
        try
        {
                impl obj=new impl();
                Naming.rebind("server",obj);
        }
        catch(Exception e) {}
        }
}
```

Implementation Program

```
import java.rmi.*;
import java.rmi.server.*;
public class impl extends UnicastRemoteObject implements AddIntf2
{
        public impl()throws RemoteException {}
        public double add(double d1,double d2) throws RemoteException
         {
                return (d1+d2);
          }
}
```

Client Program

```
import java.rmi.*;
import java.net.*;
public class client
{
	public static void main(String arg[])throws Exception
	{
		try
		{
			String serverURL="rmi://"+arg[0]+"/server";
			AddIntf2 ob=(AddIntf2)Naming.lookup(serverURL);
			double d1=Double.valueOf(arg[1]).doubleValue();
			double d2=Double.valueOf(arg[2]).doubleValue();
			System.out.print("Output ");
			System.out.println(ob.add(d1,d2));
		}
		catch(Exception e)
		{
			System.out.println(e);
		}
	 }
}
```

OUTPUT

ruler@ruler-Inspiron-N5010:~/rmi$ rmiregistry &[1] 2328

ruler@ruler-Inspiron-N5010:~/rmi$ javac impl.java

ruler@ruler-Inspiron-N5010:~/rmi$ rmic impl

ruler@ruler-Inspiron-N5010:~/rmi$ javac AddIntf2.java

ruler@ruler-Inspiron-N5010:~/rmi$ javac server.java

ruler@ruler-Inspiron-N5010:~/rmi$ java server

ruler@ruler-Inspiron-N5010:~/rmi$ javac client.java

ruler@ruler-Inspiron-N5010:~/rmi$ java client 10.17.11.104 10 20

Output 30.0

RESULT

Thus the RPC (Remote Procedure Call) program has been executed successfully.

EX.NO : 11 DATE :	Domain Name System

AIM

To develop a client application program that contact a given DNS server to resolve a given host name.

ALGORITHM

Step 1: Create a TCP socket in the client program

Step 2: Get the name of the host whose IP address in to be resolved using DNS.

Step 3: Pass this name to the DNS server through this socket.

Step 4: The server will respond with IP address of the host.

Step 5: Receive the IP address and print it.

Step 6: Stop the program.

PROGRAM (SERVER)

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
main()
{
        int sd,sd2,nsd,clilen,sport,len,i;
        char sendmsg[20],recvmsg[20];
        char ipid[20][20]={"206.190.36.45"," 173.194.38.176","173.194.36.54”};
        char hostid[20][20]={"www.yahoo.com","www.google.com","www.gmail.com"};
        struct sockaddr_in servaddr,cliaddr;
        printf("DNS Server Side\n");
        printf("Enter the Port\n");
        scanf("%d",&sport);
        sd=socket(AF_INET,SOCK_STREAM,0);
        if(sd<0)
                printf("Can't Create \n");
        else
                printf("Socket is Created\n");
                servaddr.sin_family=AF_INET;
                servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
                servaddr.sin_port=htons(sport);
                sd2=bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
        if(sd2<0)
                printf("Can't Bind\n");
        else
                printf("\n Binded\n");
                listen(sd,5);
                clilen=sizeof(cliaddr);
                nsd=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
        if(nsd<0)
                printf("Can't Accept\n");
        else
                printf("Accepted\n");
                recv(nsd,recvmsg,20,0);
        for(i=0;i<4;i++)
        {
                if(strcmp(recvmsg,hostid[i])==0)
                {
```

```
                send(nsd,ipid[i],20,20);
                break;
            }
        }
    }
```

PROGRAM (CLIENT)

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
main()
{
        int csd,cport,len;
        char sendmsg[20],recvmsg[20];
        struct sockaddr_in servaddr;
        printf("DNS Client Side\n");
        printf("Enter the Client port\n");
        scanf("%d",&cport);
        csd=socket(AF_INET,SOCK_STREAM,0);
        if(csd<0)
                printf("Can't Create\n");
        else
                printf("Socket is Created\n");
                servaddr.sin_family=AF_INET;
                servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
                servaddr.sin_port=htons(cport);
        if(connect(csd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
                printf("Can't Connect\n");
        else
                printf("Connected\n");
                printf("Enter the host address\n");
                scanf("%s",sendmsg);
                send(csd,sendmsg,20,0);
                recv(csd,recvmsg,20,20);
                printf("The Coresponding IP Address is\n");
                printf("%s",recvmsg);
}
```

OUTPUT

SERVER

[mit11mca20@mahendralinux lab]$ vi dnsserver.c

[mit11mca20@mahendralinux lab]$ cc dnsserver.c

[mit11mca20@mahendralinux lab]$ ./a.out

DNS Server Side

Enter the Port

1234

Socket is Created

Binded

CLIENT

[mit11mca20@mahendralinux lab]$ cc dnsclient.c

[mit11mca20@mahendralinux lab]$ ./a.out

DNS Client Side

Enter the Client port:9874

Socket is Created

Connected

Enter the host address

www.gmail.com

The Coresponding IP Address is

173.194.36.54

Server: Accepted

RESULT

Thus the C program for domain name system program has been executed successfully.

EX.NO : 12 DATE :	Hyper Text Transfer Protocol(HTTP)

AIM

To retrieve the data using Hyper Text Transfer Protocol .

ALGORITHM

Step 1: Start the program.

Step 2: Include the necessary header files for the program.

Step 3: Define the MAXLINE, Socket Address SERV_PORT and GET_CMD.

Step 4: Declare the server address ,sockfd and server port.

Step 5: Connect the server address to the ct.

Step 6: Check if(ct<0).

Step 7: Check for(;;) and check if((n=read(sockfd,line,MAXLINE))==0)

Step8: Close the sockfd;

Step9: Save and run the program.

Step10: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<strings.h>
#include<sys/socket.h>
#include<netdb.h>
#include<errno.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<sys/types.h>
#define MAXLINE 4096
#define SERV_PORT 80
#define SA struct sockaddr
#define GET_CMD "GET/HTTP/1.0\r\n\v\n"
int main(int argc,char **argv)
{
        int n,sockfd,ct;
        struct sockaddr_in servaddr;
        char line[MAXLINE];
        if(argc!=2)
                perror("usage web<IP address>");
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        if(sockfd<0)
                perror("socket error");
```

```
        bzero(&servaddr,sizeof(servaddr));
        servaddr.sin_family=AF_INET;
        servaddr.sin_port=htons(SERV_PORT);
        inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
        ct=connect(sockfd,(SA *)&servaddr,sizeof(servaddr));
        if(ct<0)
                perror("connect error");
        n=snprintf(line,sizeof(line),GET_CMD);
        write(sockfd,line,strlen(line));
        for(;;)
        {
                if((n=read(sockfd,line,MAXLINE))==0);
                break;
                printf("read %d bytes of home page\n",n);
                line[n]=0;
                if(fputs(line,stdout)==EOF)
                perror("fputs error");
        }
        printf("End of file on home page\n");
        close(sockfd);
}
```

OUTPUT

[cs258@admin cs258]$cc wc.c

[cs258@admin cs258]$./a.out wc.c 1 yahoomail.com

usage web<IP address>:success

RESULT

Thus the program for retrieving the data using HTTP is executed and the output verified successfully.

EX.NO : 13 DATE :	Electronic Mail(E-mail)

AIM

To write a C program for Electronic mail.

ALGORITHM

Step 1: Start the program.

Step 2: Define cknull(x) , cknltz and LIST_LEN.

Step 3: Create a text document named as sam.txt.

Step 4: Create a class named as void email_it(char *filename).

Step 5: check for(i=0; *email_list[i]>0*0;i++).

Step 6: check if(system(fpBuffer)==-1).

Step 7: Save and run the program.

Step 8: Stop the program execution.

PROGRAM

```
#include<stdlib.h>
#include<string.h>
#define cknull(x) if((x)==NULL) {perror(“”); exit(EXIT_FAILURE);}
#define cknltz(x) if((x)<0) {perror(“”); exit(EXIT_FAILURE);}
#define LIST_LEN 4
void email_it(char *filename);
main()
{
        char fname[15];
        printf(“Enter the filename\n”);
        scanf(“%s”,fname);
        email_it(fname);
}
void email_it(char *filename)
{
        char tmp[256]={0*0};
        char fpBuffer[400]={0*0};
        char email_list[LIST-LEN][256]={{mecse3@localhost.localdomain},{0*0};
        int i=0;
        for (i=0;*email_list[i]>0*0;i++)
        {
                Cknull(strcpy(tmp, email_list[i]));
                Cknltz(sprint(fpBuffer,”mail-s ‘%s%s’%s<%s”,”
                                Please Review:”,filename,tmp,filename));
                if(system(fpBuffer)==(-1))
                {
                        Perror(“Email failure”);
                        Exit(EXIT_FAILURE):

                }

        }

}
```

OUTPUT

[vvcet@localhost~]$ vi email.c

[vvcet@localhost~]$ cc email.c

[vvcet@localhost~]$ ./a.out

Enter the file name: sample.c

[vvcet@localhost~]$/home/lmel/dead.letter…..saved message

in /home/lmel/dead.letter….

RESULT

Thus the C program for Electronic mail has been executed successfully.

EX.NO : 14 DATE :	Multi-User Chat

AIM

To write a C program for implementing Client-Server Chat using TCP.

ALGORITHM

SERVER

Step 1: Start the program.

Step 2: Create an unnamed socket for the server using the parameters AF_INET as domain and the SOCK_STREAM as type.

Step 3: Name the socket using bind ( ) system call with the parameters server_sockfd and the server address (sin_addr and sin_sport).

Step 4: Create a connection queue and wait for clients using the listen( ) system call with he number of clients r equest as parameters.

Step 5: Get the client s id as input from the user to communicate. If the client s id is 0 then go to step 10 otherwise go to step 6.

Step 6: Accept the connection using accept ( ) system call when client requests for connection.

Step 7: Get the message which has to be sent to the client and check that it is not equal to Bye .

Step 8: If the message is not equal to „Bye then write the message to the client and Goto step 6.

Step 9: If the message is Bye then terminates the connection with current client and Goto step 5.

Step 10: Stop the program execution.

CLIENT

Step 1: Start the program.

Step 2: Create an unnamed socket for client using socket ( ) system.

Step 3: Call with parameters AF_INET as domain and SOCK_STREAM as type.

Step 4: Name the socket using bind( ) system call.

Step 5: Now connect the socket to server using connect ( ) system call.

Step 6: Read the message from the server socket and compare it with „Bye .

Step 7: If the message is not equal to „Bye then pr int the message to the server Output device and repeat the steps 6 & 7.

Step 8: Get the message from the client side.

Step 9: Write the message to server sockfd and goto step 4.

Step 10: If the message is equal to „Bye then print good bye message and terminate the process.

Step 11: Stop the process.

PROGRAM (SERVER)

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
main()
{
        int i,sd,sd2,nsd,clilen,sport,len;
        char sendmsg[20],rcvmsg[20];
        struct sockaddr_in servaddr,cliaddr;
        printf("Enter the port no:\n");
        scanf("%d",&sport);
        sd=socket(AF_INET,SOCK_STREAM,0);
        if(sd<0)
                printf("Can't Create \n");
        else
                printf("Socket is Created\n");
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
        servaddr.sin_port=htons(sport);
        sd2=bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
        if(sd2<0)
                printf("Can't Bind\n");
        else
                printf("\n Binded\n");
        listen(sd,5);
        do
        {
                printf("Enter the PROGRAM(CLIENT no to communicate)\n");
                scanf("%d",&i);
                printf("PROGRAM(CLIENT %d is connected)\n",i);
                clilen=sizeof(cliaddr);
                nsd=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
                if(nsd<0)
                        printf("Can't Accept\n");
                else
                        printf("Accepted\n");
                do
                 {
                        recv(nsd,rcvmsg,20,0);
                        printf("%s",rcvmsg);
                        fgets(sendmsg,20,stdin);
                        len=strlen(sendmsg);
                        sendmsg[len-1]='\0';
                        send(nsd,sendmsg,20,0);
```

```
                wait(20);
            }while(strcmp(sendmsg,"bye")!=0);
        }while(i!=0);
}
```

PROGRAM (CLIENT – 1)

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
main()
{
	int csd,cport,len;
	char sendmsg[20],revmsg[20];
	struct sockaddr_in servaddr;
	printf("Enter the port no:\n");
	scanf("%d",&cport);
	csd=socket(AF_INET,SOCK_STREAM,0);
	if(csd<0)
		printf("Can't Create\n");
	else
		printf("Socket is Created\n");
	servaddr.sin_family=AF_INET;
	servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
	servaddr.sin_port=htons(cport);
	if(connect(csd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
		printf("Can't Connect\n");
	else
		printf("Connected\n");
	do
	{
		fgets(sendmsg,20,stdin);
		len=strlen(sendmsg);
		sendmsg[len-1]='\0';
		send(csd,sendmsg,20,0);
		wait(20);
		 recv(csd,revmsg,20,0);
		printf("%s",revmsg);
	}
	while(strcmp(revmsg,"bye")!=0);
}
```

PROGRAM (CLIENT – 2)

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
main()
{
	int csd,cport,len;
	char sendmsg[20],revmsg[20];
	struct sockaddr_in servaddr;
	printf("Enter the port no:\n");
	scanf("%d",&cport);
	csd=socket(AF_INET,SOCK_STREAM,0);
	if(csd<0)
		printf("Can't Create\n");
	else
		printf("Socket is Created\n");
	servaddr.sin_family=AF_INET;
	servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
	servaddr.sin_port=htons(cport);
	if(connect(csd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
		printf("Can't Connect\n");
	else
		printf("Connected\n");
	do
	{
		fgets(sendmsg,20,stdin);
		len=strlen(sendmsg);
		sendmsg[len-1]='\0';
		send(csd,sendmsg,20,0);
		wait(20);
		recv(csd,revmsg,20,0);
		printf("%s",revmsg);
	}
	while(strcmp(revmsg,"bye")!=0);
}
```

OUTPUT
SERVER

[1me2@localhost ~]$ vi Multiuserserver.c
[1me2@localhost ~]$ cc Multiuserserverc
[1me2@localhost ~]$./a.out
Enter the port no:8543
socket is created
Binded
Enter the **PROGRAM(CLIENT** to communicate): 1
PROGRAM(CLIENT 1 is connected)
Accepted
hiiiii
Enter the **PROGRAM(CLIENT** no to communicate): 2
PROGRAM(CLIENT 2 is connected)
Accepted
hello
Enter the **PROGRAM(CLIENT** no to communicate): 0

CLIENT SIDE 1

[1me2@localhost ~]$ vi multiuser**PROGRAM(CLIENT**1.c
[1me2@localhost ~]$ cc multiuser**PROGRAM(CLIENT**1.c
[1me2@localhost ~]$./a.out
Enter the port no:8543
Socket is created
Connected
Hiiiiii

CLIENT SIDE –2

[1me2@localhost ~]$ vi multiuser**PROGRAM(CLIENT**2.c
[1me2@localhost ~]$ cc multiuser**PROGRAM(CLIENT**2.c
[1me2@localhost ~]$./a.out
Enter the port no
8543
Socket is created
Connected
Hello

RESULT

Thus the C program for chat multi client-server chat program using tcp has been executed successfully.