ASSIGNMENT-1

1. Create a message queue with message get (msgget()) function & IPC_CREAT.Also show the perror function that when message queue creation fails it will generation an error.

Program code:

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
main()
{
      int msgid; /*return value from msgget()*/
      key_t key;/*key to be passed to msgget()*/
      int mykey;
      msgid=msgget((key)|mykey,IPC_CREAT|0);
      printf("Message queue is created with key value %d",msgid);
}
```

Output:

Message queue is created with key value0.

2. IPC message queue Creation with file permission.

Program Code:

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
int main()
{
      int msgid; /*return value from msgget()*/
      key_t key; /*key to be passed to msgget()*/
      int mykey;
      msgid=msgget((key)|2,IPC_CREAT|06444);
      if(msgid<0)
      perror("Message failed");
      else
      printf("Message queue is created with key value %d",msgid);
}
```

Output:

Message queue is created with key value 32769.

3. IPC message queue creation with flag IPC_CREAT and IPC_EXCL.

Program code:

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
      int msgid; /*specified or received msg queue id */
      key_t key=11; /* key for msgget() */
      msgid=msgget(key,IPC_CREAT|IPC_EXCL);
      if(msgid<0)
      {
      perror("Message failed");
      exit(1);
      }
      else
      printf("Message queue is created with key value %d",msgid);
}
```

Output:

Message queue is created with key value 98307.

ASSIGNMENT-2

1. **Write a program to send a text message from one terminal to another terminal via IPC message queue.**
   (Hints: Create a message queue using msgget() function send & receive using Msgsnd(),msgrcv().)

Program Code:

Receive message:-

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
int main()
```

```
{
        struct msgbht
        {
                long mtype;
                char mtext[100];
        }send,recv;
        int msg,len;
        msg=msgget((key_t)25,IPC_CREAT|0666);
        if(msg<0)
        {
                perror("Message failed");
                exit(1);
        }
        strcpy(send.mtext,"i am fine thank you!");
        send.mtype=2;
        len=strlen(send.mtext);
        if(msgrcv(msg,&recv,100,1,0)==-1)
        {
                perror("Message failed");
                exit(1);
        }
        if(msgsnd(msg,&send,len,0)==-1)
        {
                perror("Message failed");
                exit(1);
        }
        printf("Message from program 1 is : \n%s\n",recv.mtext);
}
```

Program Code:

Send message:-

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
int main()
{
     struct msgbht
     {
          long mtype;
          char mtext[100];
     }send,recv;
     int msg,len;
     msg=msgget((key_t)25,IPC_CREAT|0666);
     if(msg<0)
     {
          perror("Message failed");
```

```
            exit(1);
      }
      strcpy(send.mtext,"Hello,How are you?");
      send.mtype=1;
      len=strlen(send.mtext);
      if(msgsnd(msg,&send,len,0)==-1)
      {
            perror("Message failed");
            exit(1);
      }
      if(msgrcv(msg,&recv,100,2,0)==-1)
      {
            perror("Message failes");
            exit(1);
      }
      printf("Message from program 2 is : \n%s\n",recv.mtext);
}
```

Output:

```
Message from program1 is:
Hello ,How are you?
Message from program2 is:
I am fine thank you!
```

ASSIGNMENT-3

1. Write a program to send four successive messages from the sender. The receiver will accept those messages & print them one after another as a o/p string.Receiver will not reply for those messages.

Program Code:

```
#include<sys/ipc.h>
#include<sys/msg.h>
#include<string.h>
#include<stdlib.h>
int main()
{
      struct msgbht
      {
            long mtype;
            char mtext[100];
      }
       send,recv;
      int msg,len;
      msg=msgget((key_t)25,IPC_CREAT|0666);
```

```
	if(msg<0)
	{
		perror("Message failed");
		exit(1);
	}

	strcpy(send.mtext,"Hello , How are you?");
	send.mtype=1;
	len=strlen(send.mtext);
	if(msgsnd(msg,&send,len,0)==-1)

	 {

		perror("Message failed");
		exit(1);


	}

	strcpy(send.mtext,"\nMsg2:Where are you?");
	send.mtype=2;
	len=strlen(send.mtext);
	if(msgsnd(msg,&send,len,0)==-1)

	 {
		perror("Message failed");
		exit(1);
	}

	strcpy(send.mtext,"\nMsg3:What are you doing?");
	send.mtype=3;
	len=strlen(send.mtext);
	if(msgsnd(msg,&send,len,0)==-1)

	 {
		perror("Message failed");
		exit(1);
	}

	strcpy(send.mtext,"\nMsg4:Why are you here now?");
	send.mtype=4;
	len=strlen(send.mtext);
	if(msgsnd(msg,&send,len,0)==-1)

	 {
		perror("Message failed");
		exit(1);
	}
}
```

Program Code:

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

```
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
int main()
{
      struct msgbht
      {
            long mtype;
            char mtext[100];
      }recv;
      int i;
      int qid=msgget((key_t)25,IPC_CREAT|0666);
      if(qid<0)
      {
            perror("Message failed");
            exit(1);
      }
      for(i=0;i<4;i++)
      {
            if(msgrcv(qid,&recv,100,0,0)==-1)
            {
                  printf("Msg %d(i+1)failed",i);
                  exit(1);
            }
            printf("%s",recv.mtext);
      }
}
```

Output:

```
Hello,How are you?
Where are you?
What are you doing?
Why are you here now?
```

ASSIGNMENT-4

1. Creation of a one way pipe in a single process.

Program code:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
      int pipefd[2],n;
      char buff[100];
      pipe(pipefd);
      printf("\nreadfd=%d",pipefd[0]);
      printf("\nwritefd=%d",pipefd[1]);
      write(pipefd[1],"helloworld",12);
      n=read(pipefd[0],buff,sizeof(buff));
      printf("\nsize of data%d",n);
      printf("\ndata from pipe:%s",buff);
}
```

Output:

```
Readfd=3
Writefd=4
Size of data:12
Data from pipe:Helloworld
```

2. Creation of a one way pipe in two processes.

Program Code:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
      int pipefd[2],n,pid;
      char buff[100];
```

```
      pipe(pipefd);
      printf("\nreadfd=%d",pipefd[0]);
      printf("\nwritefd=%d",pipefd[1]);
      pid=fork();
      if(pid==0)
      {
            close(pipefd[0]);
            printf("\nCHILD PROCESS SENDING DATA\n");
            write(pipefd[1],"hello world",12);
      }
      else
      {
            close(pipefd[1]);
            printf("PARENT PROCESS RECEIVES DATA\n");
            n=read(pipefd[0],buff,sizeof(buff));
            printf("\nsize of data%d",n);
            printf("\ndata received from child through
pipe:%s",buff);
      }
}
```

Output:

```
Readfd=3
Writefd=4
CHILD PROCESS SENDING DATA
Writefd=4
PARENT PROCESS RECEIVES DATA
Size of data=12
Received from child through pipe=hello world
```

3. Creation of a two way pipe between two processes.

Program Code:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
      int p1[2],p2[2],n,pid;
      char buf1[25],buf2[25];
      pipe(p1);
      pipe(p2);
      printf("\nreadfds=%d%d\n",p1[0],p2[0]);
      printf("\nwritefds=%d%d\n",p1[1],p2[1]);
      pid=fork();
      if(pid==0)
```

```
      {
            close(p1[0]);
            printf("\nCHILD PROCESS SENDING DATA\n");
            write(p1[1],"where is GCE",25);
            close(p2[1]);
            read(p2[0],buf1,25);
            printf("Reply from parent:%s\n",buf1);
      }
      else
      {
            close(p1[1]);
            printf("PARENT PROCESS RECEIVES DATA\n");
            n=read(p1[0],buf2,sizeof(buf2));
            printf("\nData received from child through
pipe:%s\n",buf2);
            sleep(3);
            close(p2[0]);
            write(p2[1],"in haldia",25);
            printf("\n Reply send\n");
      }
}
```

Output:

```
Readfd=3 5
Writefd=4 6
PARENT PROCESS RECEIVES DATA
SIZE OF DATA 30
Data received from child through pipe: where is HIT
Reply send
CHILD PROCESS SENDING DATA
Reply from parent: in haldia
```

ASSIGNMENT-5

1. Interprocess communication through FIFO between client and server.

Program Code:

Client:-

```
#include<stdio.h>
#include<ctype.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>
#include<string.h>
main()
{
      int wrfd,rdfd,n;
```

```
      char buf[50],line[50];
      wrfd=open("np1",O_WRONLY);
      rdfd=open("np2",O_RDONLY);
      printf("Enter the line of text");
      gets(line);
      write(wrfd,line,strlen(line));
      n=read(rdfd,buf,50);
      buf[n]='\0';
      printf("Full duplex client:read from the pipe:%s\n",buf);

}
```

Program Code:

Server:

```
#include<stdio.h>
#include<ctype.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>
#include<string.h>
main()
{
      int wrfd,rdfd,n,d,ret_val,count;
      char buf[50];
      ret_val=mkfifo("np1",0666);
      ret_val=mkfifo("np2",0666);
      rdfd=open("np1",O_RDONLY);
      wrfd=open("np2",O_WRONLY);
      n=read(rdfd,buf,50);
      buf[n]='\0';
      printf("Full duplex server:read from the pipe:%s\n",buf);

      count=0;
      while(count<n)
      {
            buf[count]=toupper(buf[count]);
            count++;
      }
      write(wrfd,buf,strlen(buf));
}
```

Output:

```
Enter the line of text hello
Full duplex client:read from the pipe:HELLO
```

ASSIGNMENT-6

1.Write a program to create an integer variable using shared memory concept and increment the variable simultaneously by two processes. Use semaphores to avoid race condition.

Program Code:

```
#include<sys/stat.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/shm.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<string.h>
#define SIZE 10
int *integer=0;
main()
{
      int shmid;
      key_t key_10;
      char *shm;
      int semid,pid;
      shmid=shmget((key_t)key_10,SIZE,IPC_CREAT|0666);
      shm=shmat(shmid,NULL,0);
      semid=semget(0*20,1,IPC_CREAT|0666);
      integer=(int*)shm;
      pid=fork();
      if(pid==0)
      {
            int i=0;
            while(i<10)
            {
                  sleep(3);
                  printf("\nChild process use shared memeory");
                  accessmem(semid);
                  i++;
            }
      }
      else
      {
            int j=0;
            while(j<10)
            {
                  sleep(2);
                  printf("\nParent process uses shared memory");
                  accessmem(semid);
                  j++;
            }
      }
      semctl(semid,IPC_RMID,0);
}
int accessmem(int semid)
{
```

```
      struct sembuf sop;
      sop.sem_num=0;
      sop.sem_op=1;
      sop.sem_flg=0;
      semop(semid,&sop,1);
      (*integer)++;
      printf("\t integer variable =%d",(*integer));
}
```

Output:

```
Parent process  uses shared memory integer variable=1
Child process uses shared memory integer variable=2
```

ASSIGNMENT-7

1. Write a program to design Berkeley sockets for inter process communication between client and server across the network.

In this program we illustrate the use of Berkeley sockets for inter process communication across the network. we show the communication between aserver process and a client process.

Since many server processes may be running in a system, we identify the desired server process by a "port number". Standard server processes have a worldwide unique port number associated with it. For example, the port number of SMTP(the send mail process) is 25. To see a list of server processes and their port numbers see the file /etc/services.

```
/* THE CLIENT PROCESS */
/* Please read the file Tserver.c before you read this file. To
run this,you must first change the IP address
  specified in the line:
  serv_addr.sin_addr=inet_addr("192.168.2.221"); to the IP_address
of the machine ehere you are running the server. */

/* compile--- cc Tclient.c -o cli
```

```
 * execute--- ./cli 6000 10.10.2.157       (6000 is argv[1] and
10.10.2.157 is argv[2])
 * output----Message from server: yes , I am ready    */
```

Program code:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<string.h>
#include<stdlib.h>
main(int agrc, char *argv[])
{
     int sockfd; /* socket descriptors*/
     struct sockaddr_in serv_addr;   /*structure variable
declaration*/
     int i;                /* Variable declaration*/
     char buf[100];/* We will use this buffer for communication*/

     /* opening a socket is exactly similar to the server
process*/
     if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)    /* server
socket creation*/
     {
          printf(" unable to create socket\n");
          exit(0);
     }
     /* Recall that we specified INADDR_ANY when we specified the
server address in the server.
      since the clientcan run on a different machine, we must
specify the IP addreee of the server.

      To run this client, you must change the IP address specified
below to the IP address of the machine
     where you are running the server.*/

     /* Below is the protocol definition*/
     serv_addr.sin_family=AF_INET;
     serv_addr.sin_addr.s_addr=inet_addr(argv[2]);;
     serv_addr.sin_port=htons(atoi(argv[1]));

     /* with the following information provided in serv_addr,
     the connect() system call establishes a connection with the
server process. */

     if((connect(sockfd,(struct sockaddr *)
&serv_addr,sizeof(serv_addr)))<0)
     {
          printf("Unable to connect to server\n");
          exit(0);
     }
```

```
     /* After connection , the client can send or receive
messages. However, please note that recv() will block
     when the server is not sending and vice versa. similarly
send() will blockwhen the server is not reveiving
     and vice versa.For non blocking modes,refer to online man
pages*/

                /* receiving information*/
                for(i=0;i<100;i++)
                buf[i]='\0';
                recv(sockfd,buf,100,0);
                printf("%s\n",buf);
                for(i=0;i<100;i++)
                buf[i]='\0';
                strcpy(buf,"Message from the client: hello! Are
you ready?");
                send(sockfd,buf,100,0);
                close(sockfd);
}

/* THE SERVER PROCESS */
/* COMPILE THIS PROGRAM WITH cc Tserver.c -o ser and then execute
it as ./ser */
/* Compile--- Tserver.c -o ser
 * execute--- ./ser 6000 10.10.2.157       (6000 is argv[1] and
10.10.2.157 is argv[2])
 * output---Message from the client: hello! Are you ready?" */
```

Program code:

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define serv_port 6000

main(int agrc, char *argv[])
{
     int sockfd,newsockfd; /* socket descriptors*/
     int clilen;          /* Variable declaration*/
     struct sockaddr_in cli_addr, serv_addr;    /*structure
variable declaration*/
     int i;               /* Variable declaration*/
     char buf[100];/* We will use this buffer for communication*/

     /* The following system call opens a socket.
     The first parameter indicates the family of the protocol to
be followed.
     For internet protocols we use AF_INET. For TCP sockets the
second parameter is SOCK_STREAM.
        The third parameter is  set to 0 for user applications. */
```

```
     if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)    /* server
socket creation*/
     {
          printf(" cannot create socket\n");
          exit(0);
     }

     /* The structure "sockaddr_in" is defined in <netinet/in.h>
for the internet family of protocols.
     This has three main fields. The field "sin_family" specifies
the family and is therefore AF_INET for the
     internet family. The field "sin_addr" specifies the internet
address of the server.
     This field is set to INADDR_ANY for machines having a single
IP address.
     The field "sin_port" specifies the port number of the
server.*/

     /* Below is the protocol definition*/
     serv_addr.sin_family=AF_INET;
     serv_addr.sin_addr.s_addr=INADDR_ANY;
     serv_addr.sin_port=htons(atoi(argv[1]));

     /* with the following information provided in serv_addr,
     we associate the server with its port using the bind() system
call. */

     if(bind(sockfd,(struct sockaddr *)
&serv_addr,sizeof(serv_addr))<0)
     {
          printf("Unable to bind local address\n");
          exit(0);
     }
     /* Below statement specifies that upto a 5 concurrent client
requests
     will be queued up while the system is executing the "accept"
system call below.*/

     listen(sockfd,5);

     /* In this program we are illustrating a concurrent server --
one which forks to accept multiple
     client connections concurrently. As soon as the server
accepts a connection from a client,it
     forks a child which communicates with the client, while the
parent becomes free to accept a
     new connection. To facilitate this, the accept system call
returns a new socket descriptor which
     can be used by the child. The parent continues with the
original socket descriptor.*/

     while(1)   /*chat server*/
     {
          /* The accept() system call accepts a client connection.
It blocks the server until a client comes.
```

```
          The accept() system call fills up the client's details
in a struct sockaddr which is passed as a parameter.
          The length of the structure is noted in clilen. Note
that the new socket descriptor returned by the accept()
          system call is stored in "newsockfd".*/

          /* Accepting client */

          clilen= sizeof(cli_addr);
          newsockfd=accept(sockfd,(struct sockaddr *) &cli_addr,
&clilen);
          if(newsockfd<0)
          {
               printf("Accept error\n");
               exit(0);
          }

          /* Client accept complete */
          /* Having successfully accepted a client connection, the
server now forks.
          The parent closes the new socket descriptor and loops
back to accept the next connection.*/

          if(fork()==0)
          {
               /* This child process will now communicate with
the client through the send() and recv()
               system calls.*/

               /* server socket closed*/
               close(sockfd);  /* Close the old socket since all
communications will be through the new socket*/

               /*We initialize the buffer,copy the message to it,
and send the message to the client.*/

               /* sending information*/
               for(i=0;i<100;i++)
               buf[i]='\0';
               strcpy(buf,"Message from server: yes , I am
ready");
               send(newsockfd,buf,100,0);

               /* sending information continue*/
               /* We again initialize the buffer, and receive a
message from the client.*/

               for(i=0;i<100;i++)
               buf[i]='\0';
               recv(newsockfd,buf,100,0);
               printf("%s\n",buf);

               /*closing client socket*/
               close(newsockfd);
               exit(0);
          }
```

```
            close(newsockfd);
      }
}
```

ASSIGNMENT-8

1. Design TCP iterative client and server application to reverse the given input sentence.

Program Code:
Client:-

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define MAXLINE 20
#define SERV_PORT 5777
main(int argc,char *argv)
{
      char sendline[MAXLINE],revline[MAXLINE];
      int sockfd;
      struct sockaddr_in servaddr;
      sockfd=socket(AF_INET,SOCK_STREAM,0);
      bzero(&servaddr,sizeof(servaddr));
      servaddr.sin_family=AF_INET;
      servaddr.sin_port=ntohs(SERV_PORT);
      connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
      printf("\nEnter the data to be send");
      while(fgets(sendline,MAXLINE,stdin)!=NULL)
      {
            write(sockfd,sendline,strlen(sendline));
            printf("\nline send");
            read(sockfd,revline,MAXLINE);
            printf("\n Reverse of the given sentence
is:%s",revline);
            printf("\n");
      }
      exit(0);
}
```

Program code:

Server:-

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define MAXLINE 20
#define SERV_PORT 5777
main(int argc,char *argv)
{
      int i,j;
      ssize_t n;
      char line[MAXLINE],revline[MAXLINE];
      int listenfd,connfd,clilen;
      struct sockaddr_in servaddr,cliaddr;
      listenfd=socket(AF_INET,SOCK_STREAM,0);
      bzero(&servaddr,sizeof(servaddr));
      servaddr.sin_family=AF_INET;
      servaddr.sin_port=ntohs(SERV_PORT);
      bind(listenfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
      listen(listenfd,1);
      for(;;)
      {
            clilen=sizeof(cliaddr);
            connfd=accept(listenfd,(struct
sockaddr*)&cliaddr,&clilen);
            printf("connect to client");
            while(1)
            {
                  if((n=read(connfd,line,MAXLINE))==0)
                  break;
                  line[n-1]='\0';
                  j=0;
                  for(i=n-2;i>=0;i--)
                  revline[j++]=line[i];
                  revline[j]='\0';
                  write(connfd,revline,n);
            }
      }
}
```

Output:

```
Tty1 cc server.c
./a.out
tty2  cc client.c
./a.out
Enter the data to be sent:Hi
Reverse of the given sentence:iH
```

ASSIGNMENT-9

1. Write a program to implement stop and wait protocol in data link layer.

```
/* THE RECEIVER PROCESS */
/* COMPILE THIS PROGRAM WITH cc StopWaitReceiver.c -o rec and then
execute it as ./rec */
/* compile---cc StopWaitReceiver.c -o rec
 * execute---./rec 10.10.2.157 (Ip address of the server)
 * Enter port number:5577
 */
```

Program Code:
StopWaitReceiver:-

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

main(int argc,char *argv[])
{
     int sockfd; /* socket descriptors*/
     struct sockaddr_in serv_addr;   /*structure variable
declaration*/
     int i;
     long int p;                /* Variable declaration*/
     char buf[5];/* We will use this buffer for communication*/
     printf("Enter the port address:");
     scanf("%ld",&p);

     if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)    /* server
socket creation*/
     {
          printf(" Error creation in socket!\n");
          exit(0);
     }

     /* Below is the protocol definition*/
     serv_addr.sin_family=AF_INET;
     serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
     serv_addr.sin_port=htons(p);

     if((connect(sockfd,(struct sockaddr *)
&serv_addr,sizeof(serv_addr)))<0)
     {
          printf("Unable to connect to server\n");
          exit(0);
```

```
	}

		while(1)
	{

			/* receiving information*/
			for(i=0;i<5;i++)
			buf[i]='\0';
			recv(sockfd,buf,5,0);
			printf("FRAME:%s\n",buf);
			if(buf[0]=='0')
			buf[0]='1';
			else
			buf[0]='0';
			send(sockfd,buf,5,0);
	}
	close(sockfd);
}


/* THE SENDER PROCESS */
/* COMPILE THIS PROGRAM WITH cc StopWaitSender.c -o sen and then
execute it as ./sen */
/* compile---cc StopWaitSender.c -o sen
 * execute---./sen 10.10.2.157 (Machine ip address)
 * Enter port number:5577
 */
```

Program Code:
StopWaitSender:-

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

main()
{
	int sockfd,newsockfd; /* socket descriptors*/
	int clilen;		/* Variable declaration*/
	struct sockaddr_in cli_addr, serv_addr;	/*structure
variable declaration*/
	int i,J=0;
	long int p;			/* Variable declaration*/
	char buf[5],f='0';/* We will use this buffer for
communication*/
	printf("Enter the port address:");
	scanf("%ld",&p);

	if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)	/* server
socket creation*/
```

```
     {
          printf(" Error creation in socket!\n");
          exit(0);
     }

     /* Below is the protocol definition*/
     serv_addr.sin_family=AF_INET;
     serv_addr.sin_addr.s_addr=INADDR_ANY;
     serv_addr.sin_port=htons(p);

     /* with the following information provided in serv_addr,
     we associate the server with its port using the bind() system
call. */

     if(bind(sockfd,(struct sockaddr *)
&serv_addr,sizeof(serv_addr))<0)
     {
          printf("Error in binding\n");
          exit(0);
     }
     /* Below statement specifies that upto a 5 concurrent client
requests
     will be queued up while the system is executing the "accept"
system call below.*/

     listen(sockfd,5);

          clilen= sizeof(cli_addr);
          newsockfd=accept(sockfd,(struct sockaddr *) &cli_addr,
&clilen);
          if(newsockfd<0)
          {
               printf("Accept error\n");
               exit(0);
          }



     while(1)   /*chat server*/
     {

               /* sending information*/
               for(i=0;i<5;i++)
               buf[i]='\0';
               buf[0]=f;
               send(newsockfd,buf,5,0);

               /* sending information continue*/
               /* We again initialize the buffer, and receive a
message from the client.*/

               for(i=0;i<5;i++)
               buf[i]='\0';
               recv(newsockfd,buf,5,0);
               printf("ACK:%s\n",buf);
               if(f=='0')
```

```
                    f=1;
               else
                    f='0';
     }

               /*closing client socket*/
               close(newsockfd);
               close(sockfd);
}
```

ASSIGNMENT-10

1. Write a program to implement Go back N protocol in data link layer.

```
/* THE RECEIVER PROCESS */
/* COMPILE THIS PROGRAM WITH cc GoBackNReceiver.c -o rec and then
execute it as ./rec */
/*compile---cc GoBackNReceiver.c -o rec
 * execute--./rec 10.10.2.157
 * Enter port address:5577
 */
```

Program Code:
GoBackNReceiver:-

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

main(int argc,char *argv[])
{
     int sockfd; /* socket descriptors*/
     struct sockaddr_in serv_addr;   /*structure variable
declaration*/
     int i;
     long int p;               /* Variable declaration*/
     char buf[5];/* We will use this buffer for communication*/
     char r[8]={'0','1','2','3','0','1','2','3'};
     int rf=0;
     printf("Enter the port address:");
     scanf("%ld",&p);

     if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)    /* server
socket creation*/
     {
          printf(" Error creation in socket!\n");
          exit(0);
     }
```

```
      /* Below is the protocol definition*/
      serv_addr.sin_family=AF_INET;
      serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
      serv_addr.sin_port=htons(p);

      if((connect(sockfd,(struct sockaddr *)
&serv_addr,sizeof(serv_addr)))<0)
      {
            printf("Unable to connect to server\n");
            exit(0);
      }

            while(1)
      {

                  /* receiving information*/
                  for(i=0;i<5;i++)
                  buf[i]='\0';
                  recv(sockfd,buf,5,0);
                  printf("FRAME:%s\n",buf);
                  while(r[rf]==buf[0])
                  {
                        rf++;
                  }
                  if(rf==8)
                  {
                        break;
                  }

                  buf[0]=r[rf];
                  send(sockfd,buf,5,0);
      }
      close(sockfd);
}
```

```
/* THE SENDER PROCESS */
/* COMPILE THIS PROGRAM WITH cc GoBackNSender.c -o sen and then
execute it as ./sen */
/*compile---cc GoBackNSender.c -o sen
 * execute--./sen 10.10.2.157
 * Enter port address:5577
 */
```

Program Code:
GoBackNSender:-

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

main()
{
     int sockfd,newsockfd; /* socket descriptors*/
     int clilen;          /* Variable declaration*/
     struct sockaddr_in cli_addr, serv_addr;    /*structure
variable declaration*/
     int i,J=0;
     long int p;                /* Variable declaration*/
     char buf[5];/* We will use this buffer for communication*/
     char s[8]={'0','1','2','3','0','1','2','3'};
     int sf=0,sl=2;
     printf("Enter the port address:");
     scanf("%ld",&p);

     if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)    /* server
socket creation*/
     {
          printf(" Error creation in socket!\n");
          exit(0);
     }

     /* Below is the protocol definition*/
     serv_addr.sin_family=AF_INET;
     serv_addr.sin_addr.s_addr=INADDR_ANY;
     serv_addr.sin_port=htons(p);

     /* with the following information provided in serv_addr,
     we associate the server with its port using the bind() system
call. */

     if(bind(sockfd,(struct sockaddr *)
&serv_addr,sizeof(serv_addr))<0)
     {
          printf("Error in binding\n");
          exit(0);
     }
```

```
     /* Below statement specifies that upto a 5 concurrent client
requests
     will be queued up while the system is executing the "accept"
system call below.*/

     listen(sockfd,5);

          clilen= sizeof(cli_addr);
          newsockfd=accept(sockfd,(struct sockaddr *) &cli_addr,
&clilen);
          if(newsockfd<0)
          {
               printf("Accept error\n");
               exit(0);
          }


  while(1) /*chat server*/
     {

               /* sending information*/
               for(i=0;i<5;i++)
               buf[i]='\0';
               buf[0]=s[sf];
               send(newsockfd,buf,5,0);
               /* sending information continue*/
               /* We again initialize the buffer, and receive a
message from the client.*/
               for(i=0;i<5;i++)
               buf[i]='\0';
               recv(newsockfd,buf,5,0);
               printf("ACK:%s\n",buf);
               while(s[sf]!=buf[0])
               {
                    sf++;
                    sl++;
               }
               if(sf==8)
               {
               break;
               }
     }
               /*closing client socket*/
               close(newsockfd);
               close(sockfd);
}
```