ABES Institute of Technology Ghaziabad

Affiliated to Dr. A.P.J. AKTU, Lucknow

LAB FILE

Department of Computer Science & Engineering

Subject Name: Distributed System lab

Subject Code: RCS-752

Session: 2020-2021

Semester: 7th

Submitted To:	**Submitted By:**
Ms. Swati Singh	**Student Name - Mihir Luthra**
	Roll. No. - 1729010091
	Section: 4CS-B

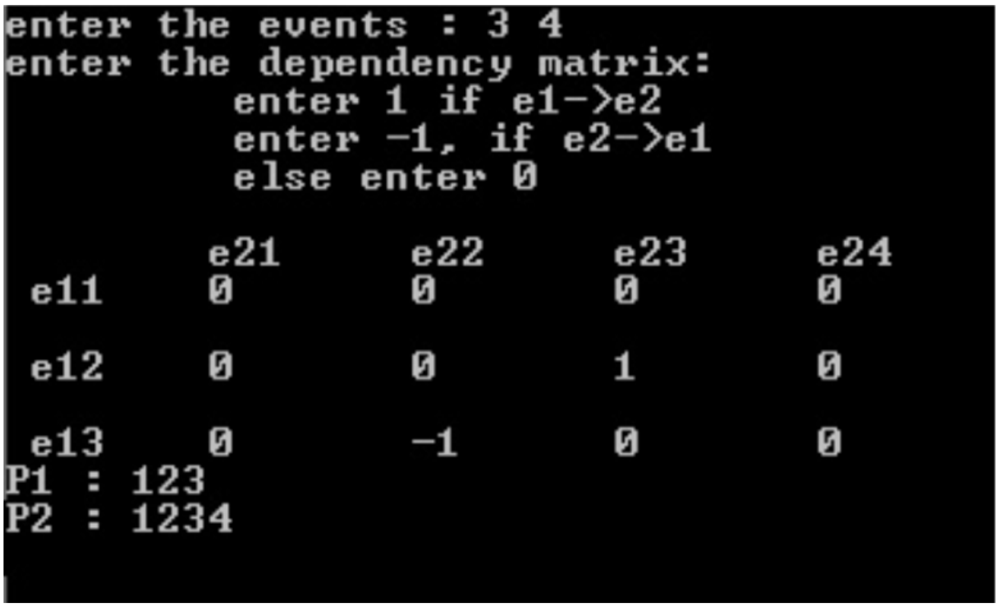
S. NO.	PRACTICALS	DATE	SIGN
1	Write a program to simulate the functioning of Lamport’s Logical clock in ‘C’.		
2	Write a program to simulate the Distributed Mutual Exclusion in ‘C’.		
3	Write a program to implement a Distributed chat server using TCP sockets in ‘C’.		
4	Implement RPC mechanism for a file transfer across a network in ‘C’.		
5	Write a JAVA code to implement ‘Java RMI’ mechanism for accessing methods of remote systems.		
6	Write a code in ‘C’ to implement sliding window protocol		
7	Implement CORBA mechanism using C++ program at one end and java program at the Other		

Q.1 Write a program to simulate the functioning of Lamport's Logical clock in 'C'.

Code :

```
#include<stdio.h>
#include<conio.h>
int max1(int a, int b)    //to find the maximum timestamp between two events
{
            if (a>b)
            return a;
else
return b;
}

int main()
{
int i,j,k,p1[20],p2[20],e1,e2,dep[20][20];
printf("enter the events : ");
scanf("%d %d",&e1,&e2);
for(i=0;i<e1;i++)
p1[i]=i+1;
for(i=0;i<e2;i++)
p2[i]=i+1;
printf("enter the dependency matrix:\n");
printf("\t enter 1 if e1->e2 \n\t enter -1, if e2->e1 \n\t else enter 0 \n\n");
for(i=0;i<e2;i++)
printf("\te2%d",i+1);
for(i=0;i<e1;i++)
{
printf("\n e1%d \t",i+1);
for(j=0;j<e2;j++)
scanf("%d",&dep[i][j]);
}

for(i=0;i<e1;i++)
{
            for(j=0;j<e2;j++)
            {
                        if(dep[i][j]==1)     //change the timestamp if dependency exist
                        {           p2[j]=max1(p2[j],p1[i]+1);
                                    for(k=j;k<e2;k++)
                                    p2[k+1]=p2[k]+1;
                        }
                        if(dep[i][j]==-1)    //change the timestamp if dependency exist
                        {
                                    p1[i]=max1(p1[i],p2[j]+1);
                                    for(k=i;k<e1;k++)
            p2[k+1]=p1[k]+1;
                        }

            }
}
printf("P1 : ");     //to print the outcome of Lamport Logical Clock
for(i=0;i<e1;i++)
{
printf("%d",p1[i]);
}
printf("\n P2 : ");
for(j=0;j<e2;j++)
printf("%d",p2[j]);

getch();
return 0 ;

}
```

Output:



```
enter the events : 3 4
enter the dependency matrix:
         enter 1 if e1->e2
         enter -1, if e2->e1
         else enter 0

        e21     e22     e23     e24
 e11    0       0       0       0

 e12    0       0       1       0

 e13    0       -1      0       0
P1 : 123
P2 : 1234
```

Q.2 Write a program to simulate the Distributed Mutual Exclusion in ‘C’.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *functionC();
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;
main()
{
int rc1, rc2;
pthread_t thread1, thread2;
/* Create independent threads each of which will execute functionC */
if( (rc1=pthread_create( &thread1, NULL, &functionC, NULL)) )
{
printf("Thread creation failed: %d\n", rc1);
}
if( (rc2=pthread_create( &thread2, NULL, &functionC, NULL)) )
{
printf("Thread creation failed: %d\n", rc2);
}
/* Wait till threads are complete before main continues. Unless we */
/* wait we run the risk of executing an exit which will terminate */
/* the process and all threads before the threads have completed. */
pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
exit(0);
}
void *functionC()
{
pthread_mutex_lock( &mutex1 );
counter++;
printf("Counter value: %d\n",counter);
pthread_mutex_unlock( &mutex1 );
}
```

Results:

```
Counter value: 1 Counter value: 2
```

join1.c

```
#include <stdio.h>

#include <pthread.h>

#define NTHREADS 10

void *thread_function(void *);

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;

int counter = 0;

main()

{

  pthread_t thread_id[NTHREADS];

int i, j;

for(i=0; i < NTHREADS; i++)

{

pthread_create( &thread_id[i], NULL, thread_function, NULL );}

for(j=0; j < NTHREADS; j++)

{

pthread_join( thread_id[j], NULL);

}

/* Now that all threads are complete I can print the final result. */

/* Without the join I could be printing a value before all the threads */

/* have been completed.*/

printf("Final counter value: %d\n", counter);

}
```

```
void *thread_function(void *dummyPtr)

{

printf("Thread number %ld\n", pthread_self());

pthread_mutex_lock( &mutex1 );

counter++;

pthread_mutex_unlock( &mutex1 );

}
```

Results:

```
Thread number 1026

Thread number 2051

Thread number 3076

Thread number 4101

Thread number 5126

Thread number 6151

Thread number 7176

Thread number 8201

Thread number 9226

Thread number 10251

Final counter value: 10
```

cond1.c

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t condition_var = PTHREAD_COND_INITIALIZER;
```

```
void *functionCount1();

void *functionCount2();

int count = 0;#define COUNT_DONE 10

#define COUNT_HALT1 3

#define COUNT_HALT2 6

main()

{

pthread_t thread1, thread2;

pthread_create( &thread1, NULL, &functionCount1, NULL);

pthread_create( &thread2, NULL, &functionCount2, NULL);

pthread_join( thread1, NULL);

pthread_join( thread2, NULL);

printf("Final count: %d\n",count);

exit(0);}

// Write numbers 1-3 and 8-10 as permitted by functionCount2()

void *functionCount1()

{

for(;;)

{

// Lock mutex and then wait for signal to relase mutex

pthread_mutex_lock( &count_mutex );

// Wait while functionCount2() operates on count

// mutex unlocked if condition varialbe in functionCount2() signaled.

pthread_cond_wait( &condition_var, &count_mutex );

count++;
```

```
printf("Counter value functionCount1: %d\n",count);

pthread_mutex_unlock( &count_mutex );

if(count >= COUNT_DONE) return(NULL);

}

}

// Write numbers 4-7

void *functionCount2()

{

for(;;)

{

pthread_mutex_lock( &count_mutex );

if( count < COUNT_HALT1 || count > COUNT_HALT2 )

{

// Condition of if statement has been met.

// Signal to free waiting thread by freeing the mutex.

// Note: functionCount1() is now permitted to modify "count".

pthread_cond_signal( &condition_var );

}

else

{

count++;

printf("Counter value functionCount2: %d\n",count);

}

pthread_mutex_unlock( &count_mutex );

if(count >= COUNT_DONE) return(NULL);
```

```
}

}
```

Results:

```
Counter value functionCount1: 1

Counter value functionCount1: 2

Counter value functionCount1: 3

Counter value functionCount2: 4

Counter value functionCount2: 5

Counter value functionCount2: 6

Counter value functionCount2: 7

Counter value functionCount1: 8

Counter value functionCount1: 9

Counter value functionCount1: 10

Final count: 10
```

Q.3 Implement a Distributed Chat Server using TCP Sockets in ‘C’.

Code:

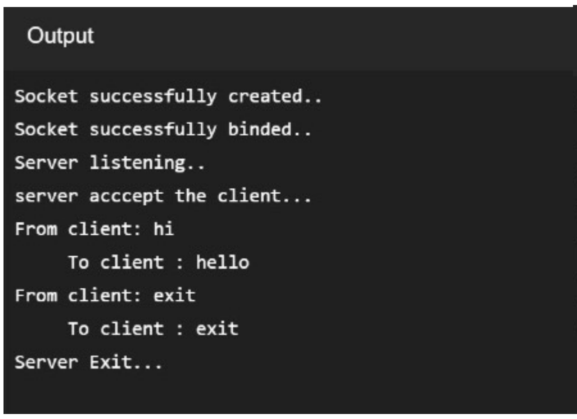
```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int sockfd)
{
        char buff[MAX];
        int n;
        // infinite loop for chat
        for (;;) {
                bzero(buff, MAX);

                // read the message from client and copy it in buffer
                read(sockfd, buff, sizeof(buff));
                // print buffer which contains the client contents
                printf("From client: %s\t To client : ", buff);
                bzero(buff, MAX);
                n = 0;
                // copy server message in the buffer
                while ((buff[n++] = getchar()) != '\n')
                        ;

                // and send that buffer to client
                write(sockfd, buff, sizeof(buff));

                // if msg contains "Exit" then server exit and chat ended.
                if (strncmp("exit", buff, 4) == 0) {
                        printf("Server Exit...\n");
                        break;
                }
        }
}

// Driver function
int main()
{
        int sockfd, connfd, len;
        struct sockaddr_in servaddr, cli;

        // socket create and verification
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd == -1) {
                printf("socket creation failed...\n");
                exit(0);
        }
        else
                printf("Socket successfully created..\n");
        bzero(&servaddr, sizeof(servaddr));

        // assign IP, PORT
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(PORT);

        // Binding newly created socket to given IP and verification
        if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
                printf("socket bind failed...\n");
                exit(0);
        }
        else
                printf("Socket successfully binded..\n");

        // Now server is ready to listen and verification
        if ((listen(sockfd, 5)) != 0) {
                printf("Listen failed...\n");
                exit(0);
        }
        else
                printf("Server listening..\n");
        len = sizeof(cli);

        // Accept the data packet from client and verification
        connfd = accept(sockfd, (SA*)&cli, &len);
        if (connfd < 0) {
                printf("server accept failed...\n");
                exit(0);
        }
        else
                printf("server accept the client...\n");

        // Function for chatting between client and server
        func(connfd);

        // After chatting close the socket
        close(sockfd);
}
```

Client side:

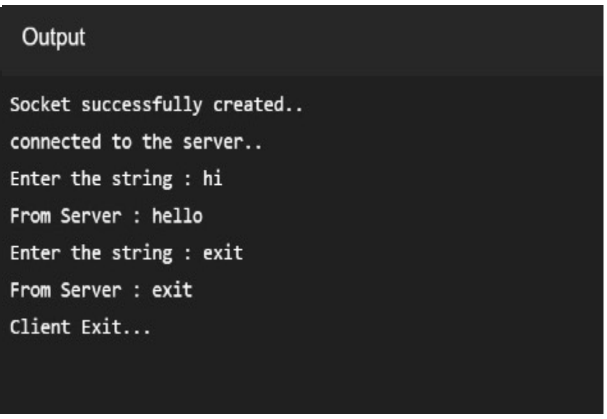
```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
        char buff[MAX];
        int n;
        for (;;) {
                bzero(buff, sizeof(buff));
                printf("Enter the string : ");
                n = 0;
                while ((buff[n++] = getchar()) != '\n')
                        ;
                write(sockfd, buff, sizeof(buff));
                bzero(buff, sizeof(buff));
                read(sockfd, buff, sizeof(buff));
                printf("From Server : %s", buff);
                if ((strncmp(buff, "exit", 4)) == 0) {
                        printf("Client Exit...\n");
                        break;
                }
        }
}

int main()
{
        int sockfd, connfd;
        struct sockaddr_in servaddr, cli;

        // socket create and verification
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd == -1) {
                printf("socket creation failed...\n");
                exit(0);
        }
        else
                printf("Socket successfully created..\n");
        bzero(&servaddr, sizeof(servaddr));

        // assign IP, PORT
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
        servaddr.sin_port = htons(PORT);
```

```
	// connect the client socket to server socket
	if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
		printf("connection with the server failed...\n");
		exit(0);
	}
	else
		printf("connected to the server..\n");

	// function for chat
	func(sockfd);

	// close the socket
	close(sockfd);
}
```

Output:-



```
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: hi
     To client : hello
From client: exit
     To client : exit
Server Exit...
```



```
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```

Q4. Implement RPC mechanism for a file transfer across a network in ‘C’.

Code:

```
/*
* The xdr routine:
* on decode, read from wire, write onto fp
* on encode, read from fp, write onto wire
*/
#include <stdio.h>
#include <rpc/rpc.h>

xdr_rcp(XDR *xdrs, FILE *fp) {
        unsigned long size;
        char buf[BUFSIZ], *p;
        if (xdrs->x_op == XDR_FREE)/* nothing to free */
        return 1;

        while (1) {
            if (xdrs->x_op == XDR_ENCODE) {
                if((size = fread(buf, sizeof(char), BUFSIZ, fp)) == 0
                                                    && ferror(fp)) {
                    fprintf(stderr, "can’t fread\n");
                    return (1);
                }
            }

        p = buf;
        if (!xdr_bytes(xdrs, &p, &size, BUFSIZ))
            return 0;

        if (size == 0)
```

```
                        return 1;

                if (xdrs->x_op == XDR_DECODE) {
                        if (fwrite(buf, sizeof(char), size,fp) != size) {
                                fprintf(stderr, "can’t fwrite\n");
                                return (1);
                        }
                }
        }
}




/*
* The sender routines
*/

#include <stdio.h>
#include <netdb.h>
#include <rpc/rpc.h>
#include <sys/socket.h>
#include <sys/time.h>

main(int argc, char **argv) {
                int xdr_rcp();
                int err;
                if (argc < 2) {
                        fprintf(stderr, "usage: %s servername\n", argv[0]);
                        exit(-1);
        }
        if ((err = callrpctcp(argv[1], RCPPROG, RCPPROC,
        RCPVERS, xdr_rcp, stdin, xdr_void, 0) != 0)) {
            clnt_perrno(err);
            fprintf(stderr, "can’t make RPC call\n");
            exit(1);
        }
        exit(0);
}

callrpctcp(host, prognum, procnum, versnum,
inproc, in, outproc, out)
        char *host, *in, *out;
        xdrproc_t inproc, outproc;
        {
            struct sockaddr_in server_addr;
            int socket = RPC_ANYSOCK;
            enum clnt_stat clnt_stat;
            struct hostent *hp;
            register CLIENT *client;
            struct timeval total_timeout;

            if ((hp = gethostbyname(host)) == NULL) {
                fprintf(stderr, "can’t get addr for ’%s’\n", host);
                return (-1);
            }

            bcopy(hp->h_addr, (caddr_t)&server_addr.sin_addr,
            hp->h_length);
            server_addr.sin_family = AF_INET;
```

```
                server_addr.sin_port = 0;

                if ((client = clnttcp_create(&server_addr, prognum,
                versnum, &socket, BUFSIZ, BUFSIZ)) == NULL) {
                        perror("rpctcp_create");
                        return (-1);
                }

                total_timeout.tv_sec = 20;
                total_timeout.tv_usec = 0;
                clnt_stat = clnt_call(client, procnum,
                inproc, in, outproc, out, total_timeout);
                clnt_destroy(client);
                return (int)clnt_stat;
        }



/*
* The receiving routines
*/
#include <stdio.h>
#include <rpc/rpc.h>

main(){
        register SVCXPRT *transp;
        int rcp_service(), xdr_rcp();
        if ((transp = svctcp_create(RPC_ANYSOCK,BUFSIZ, BUFSIZ)) == NULL) {
                fprintf("svctcp_create: error\n");
                exit(1);
        }

        pmap_unset(RCPPROG, RCPVERS);

        if (!svc_register(transp, RCPPROG, RCPVERS, rcp_service,
IPPROTO_TCP)) {

                fprintf(stderr, "svc_register: error\n");

                exit(1);

        }

        svc_run(); /* never returns */

        fprintf(stderr, "svc_run should never return\n");

}


rcp_service(register struct svc_req *rqstp, register SVCXPRT *transp) {

                switch (rqstp->rq_proc) {

                        case NULLPROC:

                                if (svc_sendreply(transp, xdr_void, 0) == 0) {

                                        fprintf(stderr, "err: rcp_service");

                                        return (1);

                                }

                                return;


                        case RCPPROC_FP:

                                if (!svc_getargs(transp, xdr_rcp, stdout)) {

                                        svcerr_decode(transp);

                                        return;

                                }

                                if (!svc_sendreply(transp, xdr_void, 0)) {

                                        fprintf(stderr, "can’t reply\n");

                                        return;

                                }

                                return (0);
                        default:
                                svcerr_noproc(transp);
                                return;
                }
}
```

Q5. Implement Java RMI mechanism for accessing methods of remote systems.

Code:

```
import java.rmi.*;

import java.rmi.server.*;

public class Hello extends UnicastRemoteObject implements HelloInterface {

private String message;

public Hello (String msg) throws RemoteException {

message = msg;

}

public String say() throws RemoteException {

return message;

}

}
```

HelloClient.java

```
import java.rmi.Naming;

public class HelloClient

{

public static void main (String[] argv) {

try {

HelloInterface hello =(HelloInterface) Naming.lookup ("//192.168.10.201/Hello");

System.out.println (hello.say());

}

catch (Exception e){

System.out.println ("HelloClient exception: " + e);}

}

}
```

HelloInterface.java

```
import java.rmi.*;
```

```
public interface HelloInterface extends Remote {

public String say() throws RemoteException;

}
```

HelloServer.java

```
import java.rmi.Naming;

public class HelloServer

{

public static void main (String[] argv)

{

try {

Naming.rebind ("Hello", new Hello ("Hello,From Roseindia.net pvt ltd!"));

System.out.println ("Server is connected and ready for operation.");

} catch (Exception e) {

System.out.println ("Server not connected: " + e);

}

}

}
```

Q.6 Sliding Window Protocol

```
#include <stdio.h>
#include <iostream.h>
#include <string>
#define THANKS -1
void main()
{
FILE *r_File1;
FILE *w_File2;
int m_framecount;
int frameCount = 0;
long currentP = 0;
long sentChar = 0;
long recvedChar = 0;
char s_name[100];
char d_name[100];
char *sp = s_name;
char *dp = d_name;
int slidingWin;
int frameSize;
int dataSize;
bool isEnd = false;
struct FRAME{
int s_flag;
intsequenceNo;
char data[90];
int n_flag;
};
FRAME frame;
frame.s_flag = 126;//set start flag
frame.n_flag = 126;//set end flag
memset(frame.data, 0, 91);//use 0 to fill full the member array in structure frame.
struct ACK{

int s_flag;
int nextSeq;
int n_flag;
}ack;
//initialize start flag and end flag in structure ack.
ack.s_flag = 126;
ack.n_flag = 126;
ack.nextSeq = NULL;
//ask the user to enter the file name and size of the sliding window.
lable1 : cout <<"Please enter source file's name!"<<endl;
cin >> sp;
cout <<"Please enter destination file's name!"<<endl;
cin >> dp;
```

```
lable2: cout <<"Please choose size of sliding window 2--7"<<endl;
cin >> slidingWin;
if((slidingWin >7 )| (slidingWin < 2))
{
cout << "wrong enter"<<endl;
goto lable2;
}
lable3: cout<< "Please enter the size of frame 14--101 Only!" << endl;
cin >>frameSize;
if((frameSize > 101) | (frameSize < 14))
{ cout << "please enter right number!"<< endl;
goto lable3;
}
//use frameSize to decide the size of the data array in the structure frame.
dataSize = frameSize - 12;
//dynamic generate a frame array with user enter's size of sliding window
FRAME *pf = new FRAME[slidingWin];
int seqNo = 0;
//strat loop for transmission.
while (ack.nextSeq != THANKS)
{
cout << "THE PROCESS ON SENDER SIDER..."<<endl;
//open a source file by read mode.
if((r_File1 = fopen(sp, "rb")) == NULL)

{
cout << "source file could not be opened please check it and re-start!" <<endl;
goto label1;
}
else
{
cout<<"Opening a file for read...";
cout <<endl;
cout <<endl;
//after opening the file, use fseek to resume the last position of a file pointer.
//Then start to read from that position.
fseek(r_File1,currentP,SEEK_SET);
//start loop for create frame array
for (int i = 0; i < slidingWin ; i++)// i is the frame array's index
{
frame.sequenceNo = seqNo;
if ((seqNo >= 7) == true)
{
seqNo = 0;//set sequence number
}
else
{
seqNo = seqNo +1;
}
//This loop is used to fill the characters read from opened file to char array data which
//is a member of the structure frame.
//we have to reserve a byte for \0 which is used to identify the end of the data array.
//that means each time we only read data size -1 characters to the data array.
for (int j = 0; j < dataSize -1; j++)
{ //if it is not end of file read a character from file then save it into data
//field in frame structure.
frame.data[j]= fgetc(r_File1);
sentChar++;//calculate how many characters will be sent.*/
if (frame.data[j]
{
cout<< "There is the end of file"<<endl;
isEnd = true;
//sentChar++;
break;
}
}
if (isEnd == true)
{
pf[i] = frame; //save a frame into frame array.
//frameCount = i;
frameCount++;
m_framecount = i +1;
cout <<endl;
cout << "The sequence number is " << pf[i].sequenceNo <<endl;
cout << "The start flag is " << pf[i].s_flag <<endl;
cout << "The Data is---->" << pf[i].data <<endl;
cout << "The end flag is " << pf[i].n_flag <<endl;
cout << "There are " <<frameCount <<" frames has been created!"<<endl;
cout << "frame " << pf[i].sequenceNo <<" has been transported!";
cout<< endl;
fclose(r_File1);
break;
}
pf[i] = frame;//sava current frame to frame buffer.
//display some information about the frame buffer.
frameCount++;
m_framecount = i +1;
cout <<endl;
cout << "The sequence number is " << pf[i].sequenceNo <<endl;
cout << "The start flag is " << pf[i].s_flag <<endl;
cout << "The Data is---->" << pf[i].data <<endl;
cout << "The end flag is " << pf[i].n_flag <<endl;
cout << "There are total " <<frameCount <<" frames has been created!"<<endl;
//cout << "frame " << pf[i].sequenceNo <<" has been transported!";
cout<< endl;
currentP = ftell(r_File1);//to record the current position of a file pointer
}
fflush(r_File1);//refresh
}

//print out some information.
cout <<endl;
cout <<"Total " << sentChar << " characters have been sent on this session!"<<endl;
cout <<endl;
cout << "waiting for ACK!" <<endl;
cout <<endl;
cout <<endl;
int nextNoRecord = 0;
cout<<"THE PROCESS ON RECEIVER SIDE..."<<endl;
//open a file for write
if((w_File2 = fopen(dp, "ab")) != NULL)
{
cout<<"opening a file for write..."<<endl;
for (int m = 0; m < m_framecount ; m++)
{
for (int n = 0; n < dataSize -1; n++)
{//check whether islast character.
if(pf[m].data[n]
{
ack.nextSeq = THANKS;
//fputc(pf[m].data[n],w_File2);
recvedChar++;
break;
}
//write the character from the current frame 's which in t index of the data field.
fputc(pf[m].data[n],w_File2);
recvedChar++;
}
cout << "The string ---->" << pf[m].data <<" written succeed"<<endl;
fflush(w_File2);//refresh
if(ack.nextSeq == THANKS)
{
fclose(w_File2);
break;
}
nextNoRecord= pf[m].sequenceNo;
}

cout <<endl;
cout <<"Total "<<recvedChar << " characters have been received on this session"<<endl;
cout <<endl;
cout << "send acknowledgement!" <<endl;
cout <<endl;
cout <<endl;
if (ack.nextSeq != THANKS)
{
cout<<"CheckACK"<<endl;
if (nextNoRecord
{
ack.nextSeq =0 ;
}
else
{
ack.nextSeq = nextNoRecord +1;
}
cout << "The next expect frame is " << ack.nextSeq <<endl;
}
else
{ cout<<"CheckACK"<<endl;
cout << "The acknowledgement is thanks. The transmission complete..."<<endl;
//delete the frame buffer array .
delete []pf;
}
}
else
{cout << "File could not be opened" << endl;}
cout <<endl;
cout <<endl;
}
/*can be used to check how many bytes in the specified file
numRead = 0;
fseek(r_File1,0,SEEK_END);
numRead = ftell(r_File1);
cout << "There are " << numRead <<" Bytes in the file" << endl;*/
}
```

Output:

```
1: use fixed source file name and fixed destination file name and fixed sliding
window size (5) to test the program.
Read file successfully.
Create frames successfully.
Save frames into the frame buffer which is size 5 successfully.
Write data from frames successfully.
Returns to ACK successfully.
Re-create new frames successfully.
Search the end of the source file successfully.
2: use keyboard to input the "source file name", "destination file name", "sliding
windows size", and "frame size" to test program
Read file successfully.
Create frames successfully.
Save frames into the frame buffer which is size 5 successfully.
Write data from frames successfully.
Returns to ACK successfully.
Re-create new frames successfully.
Search the end of source successfully.
```

Q.7 Implement CORBA mechanism by using the “C++” program at one end and “Java” program on the other.

Creating the Server

```
#include <iostream>

#include "OB/CORBA.h"

#include <OB/Cosnaming.h>

#include "crypt.h"

#include "cryptimpl.h"

using namespace std;

int main(int argc, char** argv)

{

// Declare ORB and servant object

CORBA::ORB_var orb;

CryptographicImpl* CrypImpl = NULL;

try {

// Initialize the ORB.

orb = CORBA::ORB_init(argc, argv);

// Get a reference to the root POA

CORBA::Object_var rootPOAObj =

orb->resolve_initial_references("RootPOA");

// Narrow it to the correct type
```

```
PortableServer::POA_var rootPOA =

PortableServer::POA::_narrow(rootPOAObj.in());

// Create POA policies

CORBA::PolicyList policies;

policies.length(1);

policies[0] =

rootPOA->create_thread_policy

(PortableServer::SINGLE_THREAD_MODEL);

// Get the POA manager object

PortableServer::POAManager_var manager = rootPOA->the_POAManager();

// Create a new POA with specified policies

PortableServer::POA_var myPOA = rootPOA->create_POA

("myPOA", manager, policies);

// Free policies

CORBA::ULong len = policies.length();

for (CORBA::ULong i = 0; i < len; i++)

policies[i]->destroy();

// Get a reference to the Naming Service root_context

CORBA::Object_var rootContextObj =

orb->resolve_initial_references("NameService");

// Narrow to the correct type

CosNaming::NamingContext_var nc =

CosNaming::NamingContext::_narrow(rootContextObj.in());
```

```
// Create a reference to the servant

CrypImpl = new CryptographicImpl(orb);

// Activate object

PortableServer::ObjectId_var myObjID =

myPOA->activate_object(CrypImpl);

// Get a CORBA reference with the POA through the servant

CORBA::Object_var o = myPOA->servant_to_reference(CrypImpl);

// The reference is converted to a character string

CORBA::String_var s = orb->object_to_string(o);

cout << "The IOR of the object is: " << s.in() << endl;

CosNaming::Name name;

name.length(1);

name[0].id = (const char *) "CryptographicService";

name[0].kind = (const char *) "";

// Bind the object into the name service

nc->rebind(name,o);

// Activate the POA

manager->activate();

cout << "The server is ready.

Awaiting for incoming requests..." << endl;

// Start the ORB

orb->run();

} catch(const CORBA::Exception& e) {
```

```
// Handles CORBA exceptions

cerr << e << endl;

}

// Decrement reference count

if (CrypImpl)

CrypImpl->_remove_ref();

// End CORBA

if (!CORBA::is_nil(orb)){

try{

orb->destroy();

cout << "Ending CORBA..." << endl;

} catch (const CORBA::Exception& e)

{

cout << "orb->destroy() failed:" << e << endl;

return 1;

}

}

return 0;

}
```

Implementing the Client

```
#include <iostream>

#include <string>
```

```
#include "OB/CORBA.h"

#include "OB/Cosnaming.h"

#include "crypt.h"

using namespace std;



int main(int argc, char** argv)

{

// Declare ORB

CORBA::ORB_var orb;

try {

// Initialize the ORB

orb = CORBA::ORB_init(argc, argv);

// Get a reference to the Naming Service

CORBA::Object_var rootContextObj =

orb->resolve_initial_references("NameService");

CosNaming::NamingContext_var nc =

CosNaming::NamingContext::_narrow(rootContextObj.in());

CosNaming::Name name;

name.length(1);

name[0].id = (const char *) "CryptographicService";

name[0].kind = (const char *) "";

// Invoke the root context to retrieve the object reference

CORBA::Object_var managerObj = nc->resolve(name);
```

```
// Narrow the previous object to obtain the correct type

::CaesarAlgorithm_var manager =

::CaesarAlgorithm::_narrow(managerObj.in());

string info_in,exit,dummy;

CORBA::String_var info_out;

::CaesarAlgorithm::charsequence_var inseq;

unsigned long key,shift;

try{

do{

cout << "\nCryptographic service client" << endl;

cout << "----------------------------" << endl;

do{ // Get the cryptographic key

if (cin.fail())

{

cin.clear();

cin >> dummy;

}

cout << "Enter encryption key: ";

cin >> key;

} while (cin.fail());

do{ // Get the shift

if (cin.fail())

{
```

```
cin.clear();

cin >> dummy;

}

cout << "Enter a shift: ";

cin >> shift;

} while (cin.fail());

// Used for debug pourposes

//key = 9876453;

//shift = 938372;



getline(cin,dummy); // Get the text to encrypt

cout << "Enter a plain text to encrypt: ";

getline(cin,info_in);

// Invoke first remote method

inseq = manager->encrypt

(info_in.c_str(),key,shift);

cout << "------------------------------------------"

<< endl;

cout << "Encrypted text is: "

<< inseq->get_buffer() << endl;

// Invoke second remote method

info_out = manager->decrypt(inseq.in(),key,shift);

cout << "Decrypted text is: "
```

```
<< info_out.in() << endl;

cout << "--------------------------------------"

<< endl;

cout << "Exit? (y/n): ";

cin >> exit;

} while (exit!="y");

// Shutdown server message

manager->shutdown();

} catch(const std::exception& std_e){

cerr << std_e.what() << endl;

}

}catch(const CORBA::Exception& e) {

// Handles CORBA exceptions

cerr << e << endl;

}

// End CORBA

if (!CORBA::is_nil(orb)){

try{

orb->destroy();

cout << "Ending CORBA..." << endl;

} catch(const CORBA::Exception& e)

{

cout << "orb->destroy failed:" << e << endl;
```

```
return 1;

}

}

return 0;

}
```