

**INSTITUTE OF TECHNOLOGY AND MANAGEMENT
GIDA GORAKHPUR**



**BRANCH: COMPUTER SCIENCE AND ENGINEERING
SUBJECT: DISTRIBUTED SYSTEM LAB
LAB CODE: (RCS751)
CSE-B41**

**SUBMITTED BY:
ABHISHEK MISHRA
ROLL NO: 1712010006**

**SUBMITTED TO:
MR. SHAILESH PATEL SIR**

GRADE:

SIGNATURE

INDEX

Experiment No.	Experiment Number	Date	Remark
1.	Simulate the functioning of Lamport's Logical Clock in C.		
2.	Simulate the Distributed Mutual Exclusion in C.		
3.	Implement a Distributed Chat Server using TCP Sockets in C.		
4.	Implement RPC mechanism for a file transfer across a network in C.		
5.	Implement Java RMI mechanism for accessing methods of remote systems.		
6.	Simulate Balanced Sliding Window Protocol in C.		
7.	Implement CORBA mechanism by using C++program at one end and Java program on the other.		

EXPERIMENT 1

Aim: Simulate the functioning of Lamport's Logical Clock in C

Descriptions:

The algorithm of Lamport timestamps is a simple algorithm used to determine the order of events in a distributed computer system. As different nodes or processes will typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead, and conceptually provide a starting point for the more advanced vector clock method. They are named after their creator, Leslie Lamport.

Algorithms:

- A process increments its counter before each event in that process, When a process sends a message, it includes its counter value with the message;
- On receiving a message, the counter of the recipient is updated, if necessary, to the greater of its current counter and the timestamp in the received message.
- The counter is then incremented by 1 before the message is considered received.

Program:

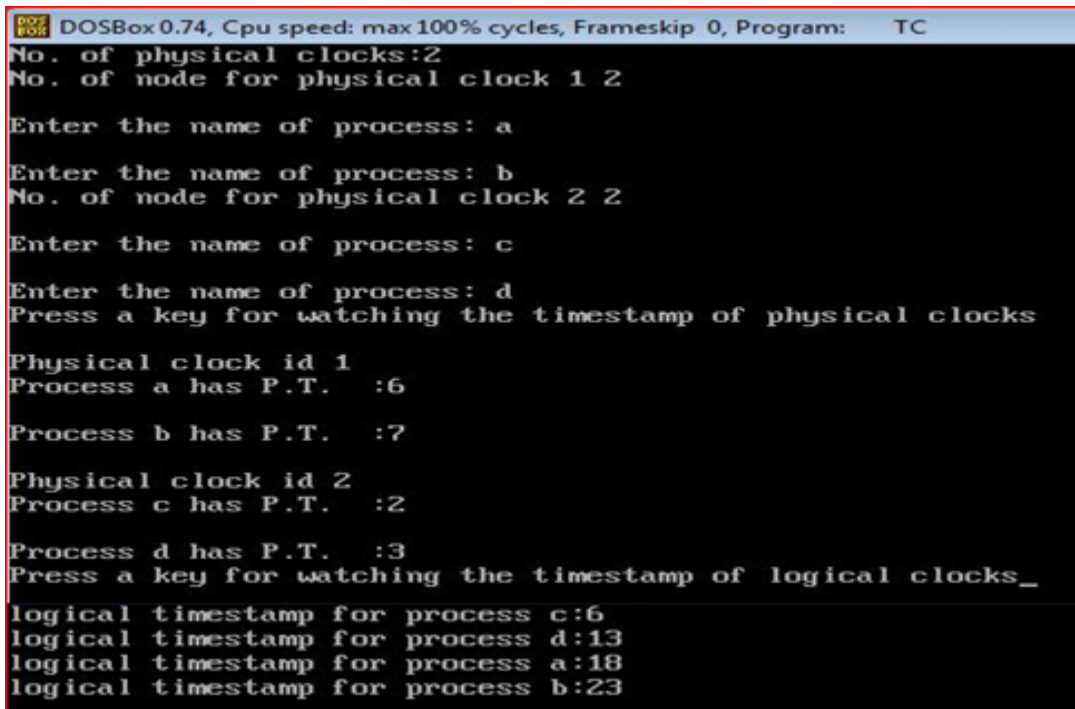
```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int i,j,k,x=0;
    char a[10][10];
    int n, num[10],b[10][10];
    clrscr();
    printf("No. of physical clocks:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("No. of node for physical clock %d", i+1);
        scanf("%d",&num[i]);
        x=0;
        for(j=0;j<num[i];j++)
        {
            printf("\nEnter the name of process: ");
            scanf("%s",&a[i][j]);
            b[i][j]=x+rand()%10;
            x=b[i][j]+1;
        }
    }
    printf("Press a key for watching the timestamp of physical clocks\n");
    getch();
    for(i=0;i<n;i++)
    {
```

```

printf("\nPhysical clock id %d",i+1);
for(j=0;j<num[i];j++)
{
printf("\nProcess %c ",a[i][j]);
printf("has P.T. :%d",b[i][j]);
printf("\n");
}}
printf("Press a key for watching the timestamp of logical clocks");
getch();
clrscr();
x=0;
for(i=0;i<10;i++){
for(j=0;j<n;j++){
for(k=0;k<num[j];k++){
if(b[j][k]==i)
{
x=rand()%10+x;
printf("logical timestamp for process %c",a[j][k]);
printf(":%d",x);
printf("\n");
}}}}
getch();
}

```

Output:



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
No. of physical clocks:2
No. of node for physical clock 1 2
Enter the name of process: a
Enter the name of process: b
No. of node for physical clock 2 2
Enter the name of process: c
Enter the name of process: d
Press a key for watching the timestamp of physical clocks
Physical clock id 1
Process a has P.T. :6
Process b has P.T. :7
Physical clock id 2
Process c has P.T. :2
Process d has P.T. :3
Press a key for watching the timestamp of logical clocks_
logical timestamp for process c:6
logical timestamp for process d:13
logical timestamp for process a:18
logical timestamp for process b:23

```

EXPERIMENT 2

Aim: Simulate the Distributed Mutual Exclusion in C.

Algorithms :

- Pushing its request in its own queue (ordered by time stamps)
- Sending a request to every node.
- Waiting for replies from all other nodes.
- If own request is at the head of its queue and all replies have been received, enter critical section.
- Upon exiting the critical section, remove its request from the queue and send a release message to every process.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<time.h>
void main()
{
int cs=0,pro=0;
double run=5;
char key='a';
time_t t1,t2;
clrscr();
printf("Press a key (except q) to enter

a process into critical section");
printf("\n Press q for exit.");
t1=time(NULL)-5;
while(key!='q')
{
while(!kbhit())
{
if(cs!=0)
{
t2=time(NULL);
if(t2-t1>run)
{
printf("Process %d ",pro-1);
printf("exits critical section.\n");
cs=0;
}}
key=getch();
if(key!='q')
{
if(cs!=0)
{
printf("Error: Another process is currently executing critical section please wait till its execution
is over.\n");
else{
printf("Process %d ",pro);
printf(" entered critical section\n");
cs=1;
pro++;
t1=time(NULL);
}}}}}
```

Output:

```
Press a key (except q) to enter a process into critical section
Press q for exit.Process 0 entered critical section
Error: Another process is currently executing critical section please wait till
its execution is over.
Error: Another process is currently executing critical section please wait till
its execution is over.
Process 0 exits critical section.
Process 1 entered critical section
Error: Another process is currently executing critical section please wait till
its execution is over.
Error: Another process is currently executing critical section please wait till
its execution is over.
Process 1 exits critical section.
Process 2 entered critical section
Error: Another process is currently executing critical section please wait till
its execution is over.
Error: Another process is currently executing critical section please wait till
its execution is over.
Error: Another process is currently executing critical section please wait till
its execution is over.
Process 2 exits critical section.
```

EXPERIMENT 3

Aim: Implement a Distributed Chat Server using TCP Sockets in C.

Descriptions:

TCP is a connection-oriented protocol that provides a reliable. flow of data between two computers. Example applications that. use such services are HTTP, FTP, and Telnet.

Program :

Server.C:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#define serv_addr.sin_family = AF_INET;
#define serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
#define serv_addr.sin_port = htons(5000);
struct sockaddr_in
{
    short int  sin_family ;
    int        sin_port;
    struct in_addr sin_addr;
};
int main(void)
{
    int listenfd = 0,connfd = 0;

    struct sockaddr_in serv_addr;

    char sendBuff[1025];
    int numrv;

    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    printf("socket retrieve success\n");

    memset(&serv_addr, '0', sizeof(serv_addr));
    memset(sendBuff, '0', sizeof(sendBuff));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(5000);

    bind(listenfd, (struct sockaddr*)&serv_addr,sizeof(serv_addr));

    if(listen(listenfd, 10) == -1){
```

```

    printf("Failed to listen\n");
    return -1;
}

while(1)
{
    connfd = accept(listenfd, (struct sockaddr*)NULL ,NULL); // accept awaiting request
    strcpy(sendBuff, "Message from server");
    write(connfd, sendBuff, strlen(sendBuff));

    close(connfd);
    sleep(1);
}
return 0;
}

```

Client.C

```

#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
#define serv_addr.sin_family = AF_INET;
#define serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
#define serv_addr.sin_port = htons(5000);
struct sockaddr_in
{
    short int  sin_family ;
    int        sin_port;
    struct in_addr sin_addr;
};
int main(void)
{
    int sockfd = 0,n = 0;
    char recvBuff[1024];
    struct sockaddr_in serv_addr;

    memset(recvBuff, '0' ,sizeof(recvBuff));
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0))< 0)
    {
        printf("\n Error : Could not create socket \n");
        return 1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5000);

```



```

serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

if(connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0)
{
    printf("\n Error : Connect Failed \n");
    return 1;
}

while((n = read(sockfd, recvBuff, sizeof(recvBuff)-1)) > 0)
{
    recvBuff[n] = 0;
    if(fputs(recvBuff, stdout) == EOF)
    {
        printf("\n Error : Fputs error");
    }
    printf("\n");
}

if( n < 0)
{
    printf("\n Read Error \n");
}

return 0;
}

```

Output:

```
socket retrieve success
█
```

```
Message from server
```

Experiment 4

Aim: Implement RPC mechanism for a file transfer across a network in C.

Server.c

```
#include <transfer.h>
#include <stdio.h>
#include <conio.h>
#include <socket.h>
char opened_file[MAXLEN];
FILE *ofile;
long long int total = 0;
int *transf_1_svc(file *argp, struct svc_req *rqstp)
{
    static int result;
    static char tempName[MAXLEN];
    strcpy(tempName, "uploaded_");
    strcat(tempName, argp->name);
    strcpy(argp->name, tempName);
    total += argp->nbytes;
    if (strcmp(opened_file, "") == 0 && ofile == NULL) {
        printf("Receiving new file %s.\n", argp->name);
        strcpy(opened_file, argp->name);
        ofile = fopen(argp->name, "ab+");
    }
    if (strcmp(opened_file, argp->name) == 0) {
        //printf("\r%lld bytes of file %s were received.", total, argp->name);
        fflush(stdout);
        fwrite(argp->data, 1, argp->nbytes, ofile);
        if (argp->nbytes < MAXLEN) {
            printf("\nFinished receiving %s.\n", argp->name);
            total = 0;
            fclose(ofile);
            ofile = NULL;
            strcpy(opened_file, "");
        }
    }
    return (&result);
}
```

Client.c

```
#include "transfer.h"
#include <time.h>
void
transfer_1(char *host, char *filetotransf)
{
    CLIENT *clnt;
    int *result_1;
    file transf_1_arg;
```

```

FILE *ofile;
long long int total = 0;
clnt = clnt_create (host, TRANSFER, TRANSFER_1, "tcp");
if (clnt == NULL) {
    clnt_pcreateerror (host);
    exit (1);
}
ofile = fopen(filetotransf, "rb");
if(ofile == NULL) {
    printf("File not found.\n");
    exit(1);
}
printf("Sending file %s.\n", filetotransf);
strcpy(transf_1_arg.name, filetotransf);
clock_t begin = clock();
while(1) {
    transf_1_arg.nbytes = fread(transf_1_arg.data, 1, MAXLEN, ofile);
    total += transf_1_arg.nbytes;
    //printf("\r%lld bytes of %s sent to server.", total, transf_1_arg.name);
    result_1 = transf_1(&transf_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    if(transf_1_arg.nbytes < MAXLEN) {
        printf("\nUpload finished.\n");
        break;
    }
}
clock_t end = clock();
double upload_time = (double)(end - begin) / CLOCKS_PER_SEC;
printf("Upload time: %lf\n", upload_time);
clnt_destroy (clnt);
fclose(ofile);
}
int main (int argc, char *argv[])
{
    char *host;
    char *filetotransf;
    if (argc < 3) {
        printf ("usage: %s <server_host> <file>\n", argv[0]);
        exit (1);
    }
    host = argv[1];
    filetotransf = argv[2];
    transfer_1 (host, filetotransf);
exit (0);
}

```

Output: Files are successfully transferred and/or able to open on client side.

EXPERIMENT 5

Aim: Implement Java RMI mechanism for accessing methods of remote systems.

Descriptions:

RMI stands for Remote Method Invocation. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package java.rmi.

Program:

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Server extends ImplExample {
    public Server() { }
    public static void main(String args[]) {
        try {
            ImplExample obj = new ImplExample();
            Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", stub);
            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Client {
    private Client() { }
    public static void main(String[] args) {
        try {
            Registry registry = LocateRegistry.getRegistry(null);
            Hello stub = (Hello) registry.lookup("Hello");
            stub.printMsg();
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Output:

```
C:\EXAMPLES\rmi>javac *.java  
C:\EXAMPLES\rmi>start rmiregistry  
C:\EXAMPLES\rmi>java Server  
Server ready  
This is an example RMI program
```

EXPERIMENT 6

Aim: Simulate Balanced Sliding Window Protocol in C

Descriptions:

A sliding window protocol is a feature of packet-based data transmission protocols. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in the Data Link Layer (OSI layer 2) as well as in the Transmission Control Protocol (TCP). Conceptually, each portion of the transmission (packets in most data link layers, but bytes in TCP) is assigned a unique consecutive sequence number, and the receiver uses the numbers to place received packets in the correct order, discarding duplicate packets and identifying missing ones. The problem with this is that there is no limit on the size of the sequence number that can be required.

Program:

```
#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;
frame.n_flag = 126;
```

```

memset(frame.data, 0, 91);
struct ACK{
int s_flag;
int nextSeq;
int n_flag;
}ack;
ack.s_flag = 126;
ack.n_flag = 126;
ack.nextSeq = NULL;
cout <<"Please enter source file's name!"<<endl; cin >> sp;
cout <<"Please enter destination file's name!"<<endl; cin>>dp;
lable2: cout <<"Please chose 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;
}
dataSize = frameSize - 12;
FRAME *pf = new FRAME[slidingWin]; int seqNo = 0;
while (ack.nextSeq != THANKS)
{
cout << "THE PROCESS ON SENDER SIDER..."<<endl;
if((r_File1 = fopen(sp, "rb")) == NULL)
{
cout << "source file could not be opened please check it and re-start!" <<endl;
goto lable1;
}
else
{
cout<<"Opening a file for read...";
cout <<endl;
cout <<endl;
fseek(r_File1,currentP,SEEK_SET);
for (int i = 0; i < slidingWin ; i++){
frame.sequenceNo = seqNo;
if ((seqNo >= 7) == true){
seqNo = 0;
}
else{
seqNo = seqNo +1;
}
}
for (int j = 0; j < dataSize -1; j++)
{
frame.data[j]= fgetc(r_File1);

```

```

sentChar++;
if (frame.data[j]
{
cout<< "There is the end of file"<<endl; isEnd =
true;
//sentChar++;
break;
}
}if (isEnd == true)
{
pf[i] = frame;
//frameCount = i;
frameCount++;
m_framecount = i +1;
cout <<endl;
cout << "The squence 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.
frameCount++;
m_framecount = i +1;
cout <<endl;
cout << "The squence 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
}
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;
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 current frame 's which in t index of data flied.
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 []pf;
}}
else

```

```

{cout << "File could not be opened" << endl;}
cout << endl;
cout << endl;
}
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 program.

Read file successfully.

Create frames successfully.

Save frames into 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 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 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.

EXPERIMENT 7

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

Description:

CORBA is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.

Program:

Server.cpp

```
#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)
{
    CORBA::ORB_var orb; CryptographicImpl*
    CrypImpl = NULL; try {
    orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var rootPOAObj = orb->resolve_initial_references("RootPOA");
    PortableServer::POA_var rootPOA = PortableServer::POA::_narrow(rootPOAObj.in());
    CORBA::PolicyList policies;
    policies.length(1);
    policies[0] = rootPOA->create_thread_policy
    (PortableServer::SINGLE_THREAD_MODEL);
    PortableServer::POAManager_var manager = rootPOA->the_POAManager();
    PortableServer::POA_var myPOA = rootPOA->create_POA
    ("myPOA", manager, policies);
    CORBA::ULong len = policies.length();
    for (CORBA::ULong i = 0; i < len; i++)
    policies[i]->destroy();
    CORBA::Object_var rootContextObj = orb->resolve_initial_references("NameService");
    CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow(rootContextObj.in());
    CrypImpl = new CryptographicImpl(orb);
    PortableServer::ObjectId_var myObjID = myPOA->activate_object(CrypImpl);
    CORBA::Object_var o = myPOA->servant_to_reference(CrypImpl
    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 *) "";
    nc->rebind(name,o);
    manager->activate();
```

```

cout << "The server is ready.
Awaiting for incoming requests..." << endl;
orb->run();
} catch(const CORBA::Exception& e) {
cerr << e << endl;
}
if (CrypImpl)
CrypImpl->_remove_ref();
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;
}

```

Client.cpp

```

#include <iostream>
#include <string>
#include "OB/CORBA.h"
#include "OB/Cosnaming.h"
#include "crypt.h"
using namespace std;
int main(int argc, char** argv)
{
CORBA::ORB_var orb; try {
orb = CORBA::ORB_init(argc, argv);
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 *) "";
CORBA::Object_var managerObj = nc->resolve(name);
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{
if (cin.fail())
{
cin.clear();
cin >> dummy;
}
cout << "Enter a shift: ";
cin >> shift;
} while (cin.fail());
getline(cin,dummy);
cout << "Enter a plain text to encrypt: ";
getline(cin,info_in);
inseq = manager->encrypt
(info_in.c_str(),key,shift);
cout << "-----" <<
endl;
cout << "Encrypted text is: " << inseq-
>get_buffer() << endl;
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) {
cerr << e << endl;
}
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;

```

}

Output:

Running the Client-server Application Once we have implemented the client and the server, it's time to connect them. Because our demonstration client and server exchange object references via the naming service, we must ensure that the naming service (which is called nameserv in Orbacus) is running. We use some command-line options to tell the naming service the host and port on which it should listen. `nameserv -OAhost localhost -OApport 8140` After this, we can start the server with a command-line option to tell it how to contact the naming service. `server -ORBInitRef NameService=corbaloc:iiop:localhost:8140/NameService` Finally we can start the client, again with a command-line option to tell it how to contact the naming service. `client -ORBInitRef NameService=corbaloc:iiop:localhost:8140/NameService`