INDEX

S.No.	Date	Aim/Objective	Remarks	T.Sign
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 java RMI mechanism for accessing methods of remote systems		
5		Simulate balanced sliding window protocol in C		
6		Implement CORBA mechanism by using C++ program at one end and java program on the other.		
7		WAP to implement RPC mechanism for a file transfer across a network.		
8		WAP for implementing Vector Clock.		
9		WAP for simulating Lamports's Logical Clock.		
10		WAP to implement 2-Phase Commit Client Server		

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

Descriptions: The algorithms 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.
Distributed algorithms such as resource synchronization often depend on some method of ordering events to function. For example, consider a system with two processes and a disk. The processes send messages to each other, and also send messages to the disk requesting access. The disk grants access in the order the messages were sent. For example, process {\displaystyle A) sends a message to the disk requesting write access, and then sends a read instruction message to process {\display style B) Process (display style B) receives the message, and as a result sends its own read request message to the disk. If there is a timing delay causing the disk to receive both messages at the same time, it can determine which

Program:
```
#include<stdio.h>
#include<conio.h>
#include<iostream.h> #include<stdlib.h>
#include<graphics.h>
#include<string.h>
#include<dos.h>

void main(){
int s[4][9].n.m-0; int ij.next-0,step 0;
int msg[10][4]-(0),totmsg.char op:
int pi.pi.ciejclrscr( k cout<<"aProgram for Lamport Logical Clock
cout<<"\nEnter Number Of Process"
cin>>n; for(i=0;i<n;i++){ cout<<"Enter number of STATES of process P"
"ins[i][8]:
cout<<"\nEnter message transit"; cout<<"\nFROM-nEnter Process Number P"
cinmsg[m][0];
cout<<"\nEnter Event Number e",cin>>msg[m][1]: cout<<"\nTOnEnter Process
Number P";cin>>msg[m][2];
cout<<"\nEnter Event Number e
cin>>msg[m][3]:
cout<<"\n\nPress 'y' to continue";op-getch();
cout<<opm++;
totmsg m
while(opy), m=0;
for (i=0: totmsg:i++){pi-msefilfol
int gd-DETECT,gm; initgraph(&gd,&gm,"C:\\TCBGI"); outtextxy(200,15,"Program
For Lamport Logical Clock");
//drawing process and eventsfor(i=0;i<n;i++){
char pl: itoa(i.pl.10);
outtextxy(5,100+next,"P"); outtextxy(13,100+next,p1);
line(100,100+next,600,100+next);
char p2: itoa(j.p2,10);
```

```
outtextxy(100+step,90+next, "e"); outtextxy(110+step,90+next,p2);
//timestampchar p3:
itoa(s[i][i]-1.p3,10); outtextxy(100+step. 110+next, "");outtextxy(110-
step.110+next,p3); circle(105+step,100+next,5); step+-50;
step=0; next+=100;
delay(2000);
for(m=0;m totmsg;m++){
//drawing message transit setlinestyle(SOLID_LINE,1,3);setcolor(m+4);
line(msg[m][1] 50+50,msg[m][0]*100+100,msg[m][3]*50+50,msg[m][2]*100+100);if
(msg[m][2] msg[m][0]){
line(msg[m][3] 50+50.msg[m][2]* 100+100.msg[m][3]*50+50.msg[m][2]*100+90);
line(msg[m][3]50+50.msg[m][2] 100+100,msg[m][3]*50+40,msg[m][2]*100+90);
}else{
line(msg[m][3] 50+50,msg[m][2]* 100+100,msg[m][3]*50+50,msg[m][2]*10+110);
line(msg[m][3] 50-50,msg[m][2]*100+100,msg[m][3]*50+40,msg[m][2]*100+110) }
getch();
}
```

<div align="center"><u>**EXPERIMENT-2**</u></div>

<u>**Aim:**</u>
Simulate the distributed mutual exclusion in C

<u>**Description:**</u>
Mutual exclusion: Concurrent access of processes to a shared resource or data is executed in a mutually exclusive manner. Only one process is allowed to execute the critical section (CS) at any given time. In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion. Message passing is the sole means for implementing distributed mutual exclusion. Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.

Three basic approaches for distributed mutual exclusion:

Token-based approach:
A unique token is shared among the sites. A site is allowed to enter its CS if it possesses the token.
Non-token-based approach: Two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next.
Quorum based approach: Each site requests permission to execute the CS from a subset of sites (called a quorum). Any two quorums contain a common site. This common site is responsible to make sure that only one request executes the CS at any time.

<u>**Program:**</u>

```
include <stdio.h>
include <stdlib.h>
include <pthread.h>

void *functionC();
pthread_mutex_t mutex] = PTHREAD_MUTEX_INITIALIZER;int counter = 0;

main ()

{int rel, re2;
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", rl); }
if (re2=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( thread], NULL);
```

```c
          pthread_join( thread2, NULL);exit(0);

}
* void *functionC()
{pthread_mutex_lock( &mutex! );counter-++,
printf{"Counter value: %d\n"counter);pthread_mutex_unlock( &mutex1 );
}
Compile: ce -Ipthread mutex Ic Run:
Counter value: | Counter value: 2joinl.c
#include <stdio.h>
#include <pthread.h>
#define NTHREADS 10
void *thread_function(void *);
pthread_mutex_t mutex} = PTL IREAD_MUTEX_INITIALIZE;
int counter = 0;
main()
{

pthread_t thread_id[NTHREADS];int i, js
for(i=0; i S NTHREADS; i++)

{
pthread_create( &thread_id[i], NULL, thread_function, NULL );}
for(j=0; | < 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( &mutex! );

}
```

Compile:cc-Ipthreadjoin1.c Run:./a.out Results:

Thread number 1026

```
Thread number 2051
Thread number 3076
Thread number 4101
Thread number 5126
Thread number 6151
Thread number 7176
Thread number 820]
Thread number 9226
Thread number 10251 Final counter value: 10cond1.c

#Include <stdio.h>
#include <stdlib.h>

#include <pthread.h-
pthread_mutex_1 count_mutex = PTHREAD_MUTEX_INITIALIZER;pthread_cond_t
condition_var

= PTHREAD_COND_INITIALIZER;
void *functionCount) ();void *functionCount2();


int count = 0;#define COUNT DONE 10//define COUNT_HALT1 3
#define COUNT_HALT2 6main()

{ pthread_t thread 1, thread2:

pthread_create( &threadl, NULL, & functionCountl, NULL); pthread_create(
&thread2, NULL,
&funetionCount2, NULL): pthread_join( thread1, NULL);

pthread_join( thread2, NULL): printf("Final count: %od\n" ,count);exit(0);}

// Write numbers 1-3 and 8-10 ay permitted by SunctionCount2(void
*functionCountl()

{
for(::)
{

// Lock mutex and then wait for signal to relase mutexpthread_mutex_lock(
&count_mutex );
//Wait while functionCount2() operates on count
//mutex unlocked if condition varialbe in SfunctionCount2() 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);
```

```c
    }
}
// Write numbers 4-7 void *functionCount2()
{
for(::)
{ pthread_mutex_lock( &count_mutex);
if( count < COUNT_HALT] || count > COUNT_HALT2 )
{
// Condition of if statement has been met.
// Signal to free waiting thread by freeing the mutex.
// Note: functionCount]() is now permitted to modify "count".

pthread_cond_signal( &condition_var );
}
else
{
count++;
printf("Counter value functionCount2: %od\n" count);
}

pthread_mutex_unlock( &count_mutex ); if(count >= COUNT_DONE) return(NULL);}}
```

Compile: ce -Ipthread cond1.c Run; ./a,out 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

EXPERIMENT-3

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

Description:
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:

```c
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define PORT 5555
#define MAXMSG 512intread_from_client (int filedes)

{char buffer[MAXMSG]; int nbytes;
nbytes = read (filedes, buffer, MAXMSG); if (nbytes <0)
{
/*Read error. */perror ("read");
exit (EXIT FAILURE);
}
else if (nbytes== 0)
/*End-of-file. */return -1;
else
{/* Data read. */
fprintf(stderr, "Server: got message: %s\n", buffer);return 0;
}
}

Int main (void){
extern int make_socket (uint16_t port); int sock;
fd set active_fd_set, read_fd_set; int i;
struct sockaddr_in clientname:size_t size;

/* Create the socket and set it up to accept connections. */
sock =make_socket (PORT);
if (listen (sock, 1)<0)
{vperror("listen"); exit (EXIT_FAILURE);
}
/* Initialize the set of active sockets. */
FD_ZERO (&active_fd_set); FD_SET (sock, &active_fd_set); while (1){

/*Block until input arrives on one or more active sockets. */
read_fd_set =active_fd_set;
```

```c
if (select (FD_SETSIZE, &read_fd_set, NULL, NULL, NULL)<0)
{
perror ("select");
exit (EXIT_FAILURE);
}
/*Service all the sockets with input pending.*/
for (i=0; i<FD_SETSIZE; ++i)
if (FD_ISSET (i, &read_fd_set))
{
if (i==sock)
{
/*Connection request on original socket. */int new;
size = sizeof (clientname);new accept (sock, (struct sockaddr *) &clientname,
&size);
if (new < 0){
perror ("accept");
exit (EXIT_FAILURE);
}
fprintf(stderr,

"Server: connect from host %s, port %hd.\n",inet_ntoa (clientname.sin_addr),
 ntohs (clientname.sin_port)); FD_SET (new, &active_fd_set);
}
else
{
/* Data arriving on an already-connected socket. */
if (read_from_client (i) < 0) {
close (i);
FD_CLR (i, &active_fd_set);
}}}}}
```

<h1 style="text-align:center">EXPERIMENT-4</h1>

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 inone 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.

Algorithm:
```
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.printin
Hellolnterface
(hello.say());
}
catch (Exception e){
System.out.println("HelloClient exception "e):
}
Hellointerface.javaimport java.rmi.*";
public interface Hellolnterface 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:");
}}}
```

Experiment 5

Aim:
Write a program for Simulate Balanced Sliding Window Protocol in C

Description:

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 (OS1 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.

Code:

```
#include <STDIO.H>
#include <iostream.h>
#include <string>
#define THANKS-1 void main()
{
FILE *r Filel; FILE *w File2; int m_framecount;
int frameCount = 0; long currentP=0; long sentChar = 0; long recvedChar = 0;
char d_name[100]; char *sp=s_name; char *dp=d_name; int sliding Win;
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 flug
memset(frame.data, 0, 91)://use 0 to fill full the member array in structure
frame. int s flag; int nextSeq;int n_flag:

}
ack:
//Initialize start flag and end flag in structure ack.acks flag = 126;
ack.n_flag 126, ack nextSeq = NULL;
//ask user to enter file name and size of sliding window, lablel:
cout<<"Please
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 >> sliding
W
if(sliding Win7 ) (sliding Win< 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 data array in structor frame.dataSize =
frameSize- 12; //dynamic generate a frame array with user enter's size of
sliding windowFRAME *pf=new
FRAME[sliding Win]; 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_Filel fopen(sp. "rb")) = NULL) { cout<<"source file could not be
opened please check it and re-start!"<<endl;goto lable 1;

else
(cout<<"Opening a file for read...":cout <<<endl;
cout <<endl;
Hafter open the file, use fseek to resume the last position of a file pointer.
//Then start to read from that position.
fseek(r Filel,currentP,SEEK SET);
//start loop for create frame array
for (int i=0; i<sliding Win; i++)
//i is the frame array's index
{frame sequenceNo-seqNo:if ((seqNo 7)= true)
(seqNo=0:
//set sequencce number
}
 else { seqNo seqNo+1;)
//This loop is used to fill the characters read front opened file to char
array data which As a memeber of structure frame. //we have to reseve a byte
for 10 which is used to identify the end of the data array. What means each
time we only read datasize-1 characters to the data array. for int 0:1
dataSize

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
{ cout<<"There is the end of file"<<endl;isEnd = true;

//sentChar++; break;
} }
 if (isEnd = true)

{pf[i]= frame; //save a frame into frame array. IframeCount=i; frameCount++;
m_framecount=i+1;cout <<endl;

cout<<"The squence number is " << pf[i].sequenceNo <<endl;cout << "
pf[i].s_flag <<endl; cout<<"The Data is >"<< pf[i].data <<<endl;
```

```cpp
cout<<"The end flag is " << pf[i].n_flag <<<endl;

cout << "There are " <<frameCount <<" frames has been created!"<<endl;
pf[i].sequenceNo <<" has been transported!";

cout<< endl; fclose(r_File1); break;

} pfli]= frame://sava current frame to frame buffer. //display some
informaiton of frame buffer.frameCount++;

m_framecount =i+1;cout <<endl;cout << "The squence number is " << pfl

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!"<<e

cout << "frame" << pili].sequenceNo <<" has been transported!";cout<< en
currentP= ftell(r_File1)://to record the current position of a file pointer

Mush(r_File1)://refresh
//print out some information.cout <<<<endl;
cout<<"Total" << sentChar <<" characters have been sent on this session
cout<<"waiting for ACK!"<<endl;cout <<endl;
cout <<endl;
int nextNoRecord rd = 6 ; 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 m_framecount
for (int n = 0 n < dataSize - l ;n++)
//check whether islast character.ifpf[m].data[n]
ack.nextSeq THANKS
fputc(pf[m].data[n] c\ F ile * 2 ) ;recvedChar: Break
//write the character from current frame 's which in t index of data flied.
fputc(pf[m].data[n],w_File2);
recvedChart:

string >"<< pf[m] data <<" written succeed"<<endl
Mush(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 s
<<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..."<<end
I/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 fill 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", "slidingwindows 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.

<h1 style="text-align:center">EXPERIMENT: 6</h1>

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:
```cpp
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; Cryptographiclmpl
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_contextCORBA::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(Cryplmpl);
// 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 *) "Cryptographic Service" name[0].kind (const char *)
        "";
        // Bind the object into the name servicenc->rebind(name,o);
        // Activate the POA manager->activate();
        cout<<"The server is ready
        Awaiting for incoming requests..." << endl;
        Start the ORBorb->run();
        } catch(const CORBA::Exception e) {
        // Handles CORBA exceptionscerr <<e << endl;
        }
        // Decrement reference countif (Cryplmpl)
        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(arge, argv);
        // Get a reference to the Naming ServiceCORBA::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 *) "Cryptographic Service":name[0].kind = (const char
        *) "";
        //Invoke the root context to retrieve the object referenceCORBA::Object_var
        managerObj = nc-
        >resolve(name);
        // Narrow the previous object to obtain the correct type
```

```cpp
::CaesarAlgorithm_var manager =
::CaesarAlgorithm::_narrow(managerObj.in());string info_in,exit,dummy;
CORBA::String_var info_out;
::Caesar Algorithm: charsequence_var insequnsigned long key,shift;
try
{do
{
cout<<"\nCryptographic service client" << endl;cout<"…….."<<endl;
do
{
// Get the cryptographic keyif
(
cin.fail())
{cin.clear();
cin>> dummy:
}
cout<<"Enter encryption key: "; cin>> key:
} while (ein.fail());do
{
// Get the shift
if (ein.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 methodinseq=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 messagemanager->shutdown();
} catch(const std::exception& std_e)
{cerr << std_e.what() <<<endl;
}}catch(const CORBA::Exception& e)
// Handles CORBA exceptionscerr <<e << endl;
}
//End CORBA
```

```cpp
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;
 }
```

AIM: Write a program to implement RPC mechanism for a file transfer across a network.

THEORY:

In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction. That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote. This is a form of client—server interaction (caller is client, executor is server), typically implemented via a request—response message-passing system. In the object-oriented programming paradigm, RPCs are represented by remote method invocation (RMI). The RPC model implies a level of location transparency, namely that calling procedures are largely the same whether they are local or remote, but usually they are not identical, so local calls can be distinguished from remote calls. Remote calls are usually orders of magnitude slower and less reliable than local calls, so distinguishing them is important.

RPCs are a form of inter-process communication (IPC), in that different processes have different address spaces: if on the same host machine, they have distinct virtual address spaces, even though the physical address space is the same; while if they are on different hosts, the physical address space is different. Many different (often incompatible) technologies have been used to implement the concept.

Sequence of events:

- The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
- The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshalling.
- The client's local operating system sends the message from the client machine to the server machine.
- The local operating system on the server machine passes the incoming packets to the server stub.
- The server stub unpacks the parameters from the message. Unpacking the parameters is called unmarshalling.
- Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

CODE:

```c
// server code for UDP socket programming
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
```

```c
#define IP PROTOCOL 0
#define PORT NO 15050
#define NET BUF SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0
#define nofile "File Not Found!"

//function to clear buffer
void clearBuf(char* b){
   int i;
   for (i = 0; i < NET BUF SIZE; i++)
        b[i]='\0';
}

// function to encrypt
char cipher (char ch) {
  return ch ˆ cipherKey;
}

 // function sending file
int sendFile(FILE* fp, char* buf, int s){
   int i, len;
   if (fp==NULL) {
        strcpy(buf, nofile);
        len = strlen(nofile);
        buf[len] = EOF;
        for (i = 0; i <len; i++)
              buf[i] = Cipher(buf[i]);
   return 1;
   }
   char ch, ch2;
  for (i = 0; i < s; i++) {
     ch = fgetc(fp);
     ch2 = cipher(ch);
     buf[i] = ch2;
     if (ch==EOF) return 1;
}
    return 0;
}

// driver code
int main(){
int sockfd, nBytes;
struct sockaddr in_addr_con;
int addrlen = sizeof(addr_con);
 addr_con.sin_family = AF INET;
addr_con.sin_port = htons(PORT_NO);
addr con.sin addr.s addr = INADDR_ANY;
char net_buf[NET BUF SIZE];
FILE* fp;
```

```c
// socket()
sockfd =  socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);
if (sockfd < 0)
    printf("\nfile descriptor not received!!\n");
else
  printf("\nfile descriptor %d received\n", sockfd);

// bind()
if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)
  printf("\nSuccessfully binded!\n");
else
  printf("\nBinding Failed!\n");

while (1) {
    printf("\nWaiting for file name...\n");
  // receive file name
  clearBuf(net_buf);
  nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
             sendrecvflag,(struct sockaddr*)&addr_con, &addrlen);

  fp = fopen(net_buf, "r");
  printf("\nFile Name Received: %s\n", net_buf);
  if (fp==NULL)
        printf("\nFile open failed!\n");
  else
        printf("\nFile Successfully opened!\n");

  while (l) {
       // process

        if sendFile(fp,net_buf, NET BUF SIZE)) {
               sendto(sockfd, net_buf, NET_BUF_SIZE, sendrecvflag,
               (struct sockaddr*)&addr_con, addrlen);
               break;
       }
       // send
       sendto(sockfd, net_buf, NET_BUF_SIZE, sendrecvflag,
      (struct sockaddr*)&addr_con, addrlen);
      clearBuf(net_buf);
     }
    if (fp !=NULL)
        fclose(fp);
  }
return 0;
}

// client code for UDP socket programming
#include <arpa/inet.h>
```

```c
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define IP_PROTOCOL 0
#define IP_ADDRESS "127.0.0.1" // localhost
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0

// function to clear buffer
void clearBuf(char* b){
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++) b[i] = '\0';
}

// function for decryption
char cipher(char ch){
    return ch ^ cipherKey;
}

// function to receive file
int recvFile(char* buf, int s){
    int i;
    char ch;
    for (i = 0; i < s; i++){
        ch = buf[i];
        ch = cipher(ch);
        if (ch==EOF)
                return 1;
        else
                printf("%c", ch);
    }
    return 0;
}

// driver code
int main(){
    int sockfd, nBytes;
    struct sockaddr in addr con;
    int addrlen = sizeof(addr con);
    addr_con.sin_family = AF INET;
    addr_con.sin_port = htons(PORT NO);
    addr_con.sin_addr.s_addr=inet_addr(IP_ADDRESS);
    char net_buf[NET BUF SIZE];
    FILE* fp;
```

```c
        // socket()
        sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

        if (sockfd < 0)
                printf("\nfile descriptor not received!!\n");
        else
                printf("\nfile descriptor %d receivcd\n", sockfd);

        while (1){
                printf("\nPlease enter file name to receive:\n");
                scanf("%s", net_buf);
                sendto(sockfd, net_buf, NET_BUF_SIZE, sendrecvflag,(struct
                        sockaddr*)&addr_con, addrlen);
                printf("\n---------Data Received ------- \n");

                while(l){
                        // receive
                    clearBuf(net_buf);
                    nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
                        sendrecvflag, (struct sockaddr*)&addr con, &addrlen);

                    // process
                    if  (recvFile(net_buf, NET_BUF_SIZE)) {
                            break;
                    }
            }
                printf("\n_____\n");
    }
    return 0;
}
```

Aim:
Write a program for Program for implementing Vector Clock

Introduction and Theory:

Vector Clock is an algorithm that generates partial ordering of events and detects causality violations in a distributed system. These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system, they detect whether a contributed event has caused another event in the distributed system. It essentially captures all the causal, relationships. This algorithm helps us Sahel every process with a vector(a list of integers) with an integer for each local dock of every process within the system. So for N given processes, there will be vector/ array of size N.

How does the vector clock algorithm work:

1. Initially all the clocks are set to zero.
2. Every time, an Internal event occurs in a process, the value of the processes's logical clock in the vector is incremented by 1
3. Also, every time a process sends a message, the value of the processes's logical dock in the vector is incremented by 1.
4. Every time, a process receives a message, the value of the processes's logical. clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element)

Code:

```
#include<stdio.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
long *p1(int i,long *comp);
long *p2(int i,long *comp);
long *p3(int i,long *comp);
void main()
{
    long start[]={0,0,0},*vector;
    clrscr();
    while(!kbhit())
    {
        p1(1,&start[0]);
    }
    printf("\n Process Vector\n");
    vector=p1(0,&start[0]);
    printf("p1[%ld%ld%ld]\n",*vector,*(vector+1),*(vector+2));
    vector=p2(0,&start[0]);
    printf("p2[%ld%ld%ld]\n",*vector,*(vector+1),*(vector+2));
```

```c
        vector=p3(0,&start[0]);
        printf("p3[%ld%ld%ld]\n",*vector,*(vector+1),*(vector+2));
}
        long *p1(int i,long *comp)
        {
                static long a[]={0,0,0};
                int next;
             if(i==1)
              {
                    a[0]++;
                    if(*(comp+1)>a[1])
                          a[1]=*(comp+1);
                    if(*(comp+2)>a[2])
                          a[2]=*(comp+2);
                    next=random(2);
                    if(next==0)
                          p2(1,&a[0]);
                    else if(next==1)
                          p3(1,&a[0]);
                    return(&a[0]);
              }
              else
                    return(&a[0]);
        }
long *p2(int i,long *comp)
{
        static long b[]={0,0,0};
        int next;
        if(i==1)
        {
                b[i]++;
                if(*comp>b[0])
                      b[0]=*(comp);
                if(*(comp+2)>b[2])
                      b[2]=*(comp+2);
                next=random(2);
                if(next==0)
                      p1(1,&b[0]);
                else if(next==1)
                      p3(1,&b[0]);
                return &b[0];
        }
        else
                return &b[0];
}
long *p3(int i,long *comp)
{
        static long c[]={0,0,0};
        int next;
        if(i==1)
```

```c
        {
                c[2]++;
                if(*comp>c[0])
                        c[0]=*(comp);
                if(*(comp+1)>c[1])
                        c[1]=*(comp+1);
                next=random(2);
                if(next==0)
                        p1(1,&c[0]);
                return &c[0];
        }
        else
                return &c[0];
}
```

AIM: Write a program for simulating Lamport Logical Clock.

THEORY:

Introduction:
Lamport introduced a system of logical clocks in order to make the -> relation possible. It works like this: Each
process Pi in the system has its own clock Ci. Ci can be looked at as a function that assigns a number, Ci (a) to
an event a. This is the timestamp of the event a in process Pi. These numbers are not in any way related to
physical time -- that is why they are called logical clocks. These are generally implemented using counters, which
increase each time an event occurs. Generally, an event's timestamp is the value of the clock at that time it occurs.

Description:
Happened Before Relation (->). This relation captures causal dependencies between events, that is, whether or not events have a cause-and-effect relation.
This relation (->) is defined as follows:
- a -> b, if a and b are in the same process and a occurred before b.
- a -> b, if a is the event of sending a message and b is the receipt of that message by another process.
- If a -> b and b -> c, then a -> c - that is, the relation has the property of transitivity.
- Causally Related Events: If event a -> event b, then a causally affects b.
- Concurrent Events: Two distinct events a and b are concurrent (a || b) if (not) a -> b and (not) b > a. That is, the events have no causal relationship. This is equivalent to b || a.
- For any two events a and b in a system, only one of the following is true: a -> b, b -> a, or a || b. e11 → e12 , e12 → e22e21 → e13 , e14 || e24

Conditions Satisfied by the Logical Clock system:
For any events a and b, if a -> b, then C(a) < C(b). This is true if two conditions are met:
- If a occurs before b, then Ci(a) < Ci(b).
- If a is a message sent from Pi and b is the receipt of that same message in Pj, then Ci(a) < Cj(b).
- Implementation Rules Required:
- Clock Ci is incremented for each event: Ci := Ci + d (d > 0)
- if a is the event of sending a message from one process to another, then the receiver sets its clock to the max of its current clock and the sender's clock - that is, Cj := max(Cj, tm + d) (d> 0)

Limitation of Lamport's Clocks:
- If a → b then C(a) < C(b) but C(a) < C(b) does not necessarily imply a →b

CODE :

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int i,j,k; int x=0;
```

```c
    char a[10][10];
    int n,num[10],b[10][10];

    printf("Enter the no. of physical clocks: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nNo. of nodes 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("\n\n");
    for(i=0;i<n;i++)
    {
        printf("Physical Clock %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\n");
    }
    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("\nLogical Clock Timestamp for process
%c",a[j][k]); printf(" : %d ",x);
                    printf("\n");
                }
    return;
}
```

AIM: To implement 2-Phase Commit client-server.
Introduction and Theory

In a local database system, for committing a transaction, the transaction manager has to only convey the decision to commit to the recovery manager. However, in a distributed system, the transaction manager should convey the decision to commit to all the servers in the various sites where the transaction is being executed and uniformly enforce the decision. When processing is complete at each site, it reaches the partially committed transaction state and waits for all other transactions to reach their partially committed states. When it receives the message that all the sites are ready to commit, it starts to commit. In a distribu them does.

Figure 1 • The two-phase commit protocol

Distributed two-phase commit reduces the vulnerability of one phase commit protocols. The steps performed in the two phases are as follows −
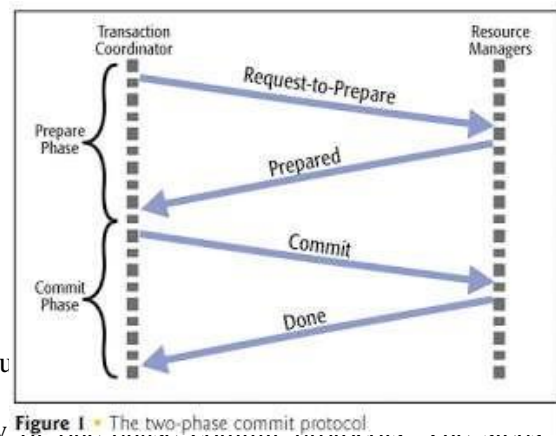
Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a "DONE" message to the controlling site. When the controlling site has received "DONE" message from all slaves, it sends a "Prepare" message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit,it sends a "Ready" message.
- A slave that does not want to commit sends a "Not Ready" message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received "Ready" message from all the slaves −
  - The controlling site sends a "Global Commit" message to the slaves.
  - The slaves apply the transaction and send a "Commit ACK" message to the controlling site.
  - When the controlling site receives "Commit ACK" message from all the slaves,it considers the transaction as committed.
- After the controlling site has received the first "Not Ready" message from any slave −
  - The controlling site sends a "Global Abort" message to the slaves.
  - The slaves abort the transaction and send a "Abort ACK" message to the controlling site.
  - When the controlling site receives "Abort ACK" message from all the slaves, it considers the transaction as aborted.

CODE:

```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>
#include <time.h>
#include <string.h>
#define MSG_CONFIRM 0


#define TRUE 1
#define FALSE 0
#define ML 1024
#define MPROC 32

typedef struct wireless_node
{
int priority;
int parent;
} wireless_node;

wireless_node w;

int max(int a, int b)
{
return a >= b? a:b;
}

int connect_to_port(int connect_to)
{
int sock_id;
int opt = 1;
struct sockaddr_in server;
if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
perror("unable to create a socket");
exit(EXIT_FAILURE);
}
setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void
*)&opt, sizeof(int));
memset(&server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(connect_to);
```

```c
    if (bind(sock_id, (const struct sockaddr *)&server,
    sizeof(server)) < 0)
    {
    perror("unable to bind to port");
    exit(EXIT_FAILURE);
    }
    return sock_id;
    }

    void send_to_id(int to, int from, char message[ML])
    {
    struct sockaddr_in cl;
    memset(&cl, 0, sizeof(cl));

    cl.sin_family = AF_INET;
    cl.sin_addr.s_addr = INADDR_ANY;
    cl.sin_port = htons(to);

    sendto(
    from, \
    (const char *)message, \
    strlen(message), \
    MSG_CONFIRM, \
    (const struct sockaddr *)&cl, \
    sizeof(cl));
    }

    void begin_commit(int id, int *procs, int num_procs)
    {
    int itr;
    char message[ML];
    sprintf(message, "%s", "SCMT");
    for (itr = 0; itr < num_procs; itr++)
    {
    printf("Sending begin commit to: %d\n", procs[itr]);
    send_to_id(procs[itr], id, message);
    }
    }

    void announce_action(int self, int *procs, int num_procs, char
    msg[ML])
    {
    int itr;

    for (itr = 0; itr < num_procs; itr++)
    {
    send_to_id(procs[itr], self, msg);
    }
    }
```

```c
int main(int argc, char* argv[])
{
int self = atoi(argv[1]);
int n_procs = atoi(argv[2]);
int procs[MPROC];
int sender, okcnt = 0, nocnt = 0, dncnt = 0;
int sock_id, coord_id;
int itr, len, n, start, ix;

char buffer[ML], flag[ML], p_id[ML], msg[256];

struct sockaddr_in from;

for(itr = 0; itr < n_procs; itr += 1)
procs[itr] = atoi(argv[3 + itr]);
printf("Creating node at %d\n", self);
sock_id = connect_to_port(self);
begin_commit(sock_id, procs, n_procs);
while(TRUE)
{
sleep(2);
memset(&from, 0, sizeof(from));
// printf("Tring read\n");
n = recvfrom(sock_id, (char *)buffer, ML, MSG_WAITALL,
(struct sockaddr *)&from, &len);
buffer[n] = '\0';
printf("Recieved: %s\n", buffer);

if (strcmp(buffer, "CMOK") == 0)
{
okcnt += 1;
}
else if (strcmp(buffer, "CMNO") == 0)
{
nocnt += 1;
}
if ((nocnt + okcnt) == n_procs)
{
printf("Recieved replies from all clients\n");
if (okcnt == n_procs)
{
printf("Announcing complete commit\n");
announce_action(sock_id, procs, n_procs, "CDON");
}
else
{
printf("Announcing abort commit\n");
announce_action(sock_id, procs, n_procs, "CABT");
```

```c
    }
    }
    if (strcmp(buffer, "DONE") == 0)
    {
    dncnt += 1;
    printf("clients confirmed commit\n");
    if (dncnt == n_procs)
    {
    printf("All process announced commit action\n");
    exit(EXIT_SUCCESS);
    }
    }
    // printf("Waiting\n");
    }
    return 0;
    }
```