Meerut Institute of Engineering & Technology

Department of Computer Science & Engineering

Dr. A.P.J. Abdul Kalam Technical University, Lucknow

LAB MANUAL

Subject Code: RCS-701

Subject Name: Distributed System Lab

INDEX

Vision of the Institute

To be an outstanding institution in the country imparting technical education, providing need based, value based and career based programmes and producing self-reliant, self-sufficient technocrats, capable of meeting new challenges.

Mission of the Institute

To educate young aspirants in various technical fields to fulfill global requirement of human resources by providing sustainable quality education, training and invigorating environment, also molding them into skilled competent and socially responsible citizens who will lead the building of a powerful nation.

Vision of Department

To be an excellent department that imparts value based quality education and uplifts innovative research in the ever-changing field of technology.

Mission of Department

1. To fulfill the requirement of skilled human resources with focus on quality education.
2. To create globally competent and socially responsible technocrats by providing value and need based training.
3. To improve Industry-Institution Interaction and encourage the innovative research activities.

Program Educational Objectives

1. Students will have the successful careers in the field of computer science and allied sectors as an innovative engineer.
2. Students will continue to learn and advance their careers through participation in professional activities, attainment of professional certification and seeking advance studies.
3. Students will be able to demonstrate a commitment to life-long learning.
4. Students will be ready to serve society in any manner and become a responsible and aware citizen.
5. Establishing students in a leadership role in any field.

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

1. Ability to apply and analyze computational concepts in the areas related to algorithms, machine learning, cloud computing, web designing and web services.
2. Ability to apply standard practices and methodologies in software development and project management.
3. Ability to employ fundamental concepts and emerging technologies for innovative research activities, carrier opportunities & zeal for higher studies.

Course Outcomes

CO-1	To understand the principles & basic concepts of distributed systems.
CO-2	To understand the concepts of Fault Tolerance and failure recovery of resources in distributed system.
CO-3	To solve problems in distributed Mutual Exclusion using various algorithms and methods.
CO-4	To analyze different Protocols in Distributed Systems.
CO-5	To analyze different distributed system transactions and concurrency controls.

List of Experiments

Exp. No.	**Experiment Name**	**Course Outcome**
1	Simulate the functioning of Lamport's Logical Clock in 'C'.	CO1
2	Simulate the Distributed Mutual Exclusion in 'C'.	CO2
3	Implement a Distributed Chat Server using TCP Sockets in 'C'.	CO3
4	Implement RPC mechanism for a file transfer across a network in 'C'	CO3
5	Implement 'Java RMI' mechanism for accessing methods of remote systems.	CO4
6	Simulate Balanced Sliding Window Protocol in 'C'.	CO4
7	Implement CORBA mechanism by using 'C++' program at one end and Java program on the other.	CO5
Value added Programs 8		
8	Write an algorithm for implementation of Round robin algorithm for CPU Scheduling.	CO2
9	Write an algorithm for implementation of Shortest Job First (SJF) for CPU Scheduling	CO2
10	Write an algorithm for implementation of Priority Scheduling for CPU Scheduling.	CO2

Lab Incharge   Head(CSE)

PRACTICAL No.-1

Objective: Write an algorithm to simulate the functioning of Lamport’s clocks.

Algorithm:

Lamport Algorithm/Time Stamp

Vector Time Stamp--- extension of Lamports approach

If two products do not interact them there is no need for synchronization of their logical clocks.

1. Lamport Algorithm/Time Stamp.

Concept in Logical clocks :

1. **Happens before relation** : a → b if one of the following is true

   a. If a and b are events in the same process and occurs a before b.
   b. If a is event of a message sending the message and b is the event of the message being receiving the message by another process a→b is also true.
   c. It implies that the message cannot be received before it is sent.

2. **Happens before is Transitive :** that means if a → b, b→C then a→C

3. **Concurrent :** If two events x and y are in two processes that never exchange messages than x and y are concurrent

Lamport Algorithm/Time Stamp.

For every event a, we can assign a time C(a) on which all process agrees.

If a →b than C(a) → C(b).

Alternatively:

1. If a and b are events in the same process and occurs a before b, (a→b) than C (a) < C (b).

2. If a is event sending the message and b is the event receiving the message is another process $C(a) < C(b)$.
3. for all distinctive events a & b, $C(a) \neq C(b)$

PRACTICAL No.-2

Objective: Simulate the Distributed Mutual Exclusion in 'C'.

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

Mutual exclusion in single computer system Vs. distributed system:

In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved.

In Distributed systems, we neither have shared memory nor a common physical clock and there for we can not solve mutual exclusion problem using shared variables. To eliminate the mutual exclusion problem in distributed system approach based on message passing is used.

A site in distributed system do not have complete information of state of the system due to lack of shared memory and a common physical clock.

Requirements of Mutual exclusion Algorithm:

No Deadlock:

Two or more site should not endlessly wait for any message that will never arrive.

No Starvation:

Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section

Fairness:

Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.

Fault Tolerance:

In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Solution to distributed mutual exclusion:

As we know shared variables or a local kernel can not be used to implement mutual exclusion in distributed systems. Message passing is a way to implement mutual exclusion. Below are the three approaches based on message passing to implement mutual exclusion in distributed systems:

Token Based Algorithm:

A unique token is shared among all the sites.

If a site possesses the unique token, it is allowed to enter its critical section

This approach uses sequence number to order requests for the critical section.

Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.

This approach insures Mutual exclusion as the token is unique

Example:

Suzuki-Kasami’s Broadcast Algorithm

Non-token based approach:

A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successive round of messages among sites.

This approach use timestamps instead of sequence number to order requests for the critical section.

When ever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.

All algorithm which follows non-token based approach maintains a logical clock. Logical clocks get updated according to Lamport's scheme

Example:

Lamport's algorithm, Ricart–Agrawala algorithm

Quorum based approach:

Instead of requesting permission to execute the critical section from all other sites, Each site requests only a subset of sites which is called a quorum.

Any two subsets of sites or Quorum contains a common site.

This common site is responsible to ensure mutual exclusion

Example:

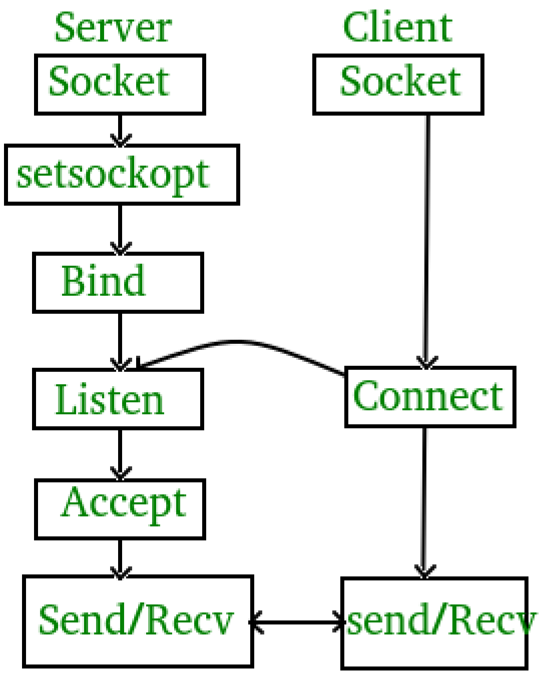
Maekawa's Algorithm

PRACTICAL No.-3

Objective: Implement a Distributed Chat Server using TCP Sockets in ‘C’.

If we are creating a connection between client and server using TCP then it has few functionality like, TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

The entire process can be broken down into following steps:



The entire process can be broken down into following steps:

TCP Server –

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.

3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3.

PRACTICAL No.-3

Objective: Implement a Distributed Chat Server using TCP Sockets in ‘C’.

Program : ►SOURCE CODE

```
#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 512
int
read_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)
{
perror ("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;
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)
```

```
{
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)
{
perror ("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;
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)
```

PRACTICAL No.-4

Objective: Implement RPC mechanism for a file transfer across a network in ‘C’

PRACTICAL No.-5

Objective: 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.

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.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);
} } }
```

P ost-ExperimentQuestions:

Q1. What is the use of RMI registry?
Q2. What is meant by distributed garbage collection?
Q3. Explain the use of Reflection in RMI?

PRACTICAL No.-6

Objective: Simulate Balanced Sliding Window Protocol in ‘C’.

```
#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 user to enter file name and size of 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 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;
}

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 user to enter file name and size of 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 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;
}

//This loop is used to fill the characters read from opened file to char array data which
//is a memeber of structure frame.
//we have to reseve a byte for \0 which is used to identify the end of the data array.
//that means each time we only read datasize -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 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.
//display some informaiton of 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
}
//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 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 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”, “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.

Post-ExperimentQuestions:

Q1. Define sliding window protocol ‘
Q2.Explain working of a sliding window protocol .

PRACTICAL No.-7

Objective: Implement CORBA mechanism by using „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;
}

// 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;
}

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;
}
```

►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 -OAport 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

PRACTICAL No.-8

Objective: Write an algorithm for implementation of Ricart Agarwala algorithm for distributed mutual exclusion.

Theory: The Ricart- Agrawala Algorithm is an algorithm for mutual exclusion on a distributed system.This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm, by removing the need for release messages. It was developed by Glenn Ricart and Ashok Agrawala.

Steps of an algorithm:

1. Process requesting the Critical Section (CS):

When a process wants to enter the CS, it sends a time stamped request to all other processes.

2. When a process receives a request:

If it is neither requesting nor executing the CS, it returns a reply (not time stamped).

If it is requesting the CS, but the timestamp on the incoming request is smaller than the timestamp on its own request, it returns a reply which means the other process requested first.

Otherwise, it defers answering the request.

3. Process executing the CS:

A process enters the CS when it has received a reply from all other processes in the system.

4. Process releasing the CS:

When a process leaves the CS, it:

- Sends a reply message to all the deferred requests.
- Process with next earliest request will now received its last reply message and enter the CS.

Algorithm:

On initialization

state:= RELEASED;

```
To enter the section
state:= WANTED;
Multicast request to all processes;       processing of incoming requests deferred here
T:= request’s timestamp;
Wait until (number of replies received = (N– 1));
state:= HELD;
On receipt of a request <Ti, pi>  at pj (i ≤ j)
if (state= HELD or (state= WANTED and (T, pj) < (Ti, pi) ))
then
queue request from pi without replying;
else
reply immediately to pi;
end if
To exit the critical section
state:= RELEASED;
reply to any queued requests;
```

VALUE ADDED PRACTICALS:

PRACTICAL No.-8

Objective: An algorithm for implementation of Round robin algorithm for CPU Scheduling

Theory: Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is preemptive as processes are assigned CPU only for a fixed slice of time at most. The disadvantage of it is more overhead of context switching.

Algorithm:

```
        let (time) quantum be 2 (sec.) ---time slice

        if countQueue(CPUQUEUE) equals 1 THEN --- is a process (tran) running ?

        if headqueue[CPUQUEUE] PCB_run equals 0 THEN --- process has completed take
process off CPU

        if countQueue(READYQUEUE) greater than 0 THEN issue_process onto CPU
decrement time    quantum  counter

        else --- process has not completed

        if headqueue[CPUQUEUE] time quantum equals 0 THEN --- time's up!

        if countQueue(READYQUEUE) greater than 0 THEN remove process from CPUQ and
insert into READYQUEUE reset time quantum counter issue next available process onto CPUQ
decrement time quantum counter

        else ----no processes (trans) in ready queuereset time quantum counterrun for a
tickdecrement time quantum counter

            else ---quantum has not expiredrun for a tickdecrement time quantum counter

                else ---CPU is idle
```

```
                    if (countQueue(READYQUEUE) greater than 0 THEN issue process onto
CPU decrement time quantum counter
                              end_if
                         end_if
                     end_if
                 end_if
           end_if
 end
```

PRACTICAL No.-9

Objective: Write an algorithm for implementation of Shortest Job First (SJF) for CPU Scheduling.

Theory: SJF is a **Shortest Job First Scheduling Algorithm** that assigns to each process the length of its next CPU burst/execution time. CPU is then given to the process with the minimal CPU burst from the waiting queue. *SJF* is provably optimal, in that for a given set of processes and their CPU bursts/execution times it gives the least average waiting time for each process.

Algorithm:

1. Firstly start process.

2. Declare array size ie. A[10].

3. Select process which has shortest burst time among all process will execute first.

4. If process have same burst time length then FCFS (First come First Serve) scheduling algorithm used.

5. Make average waiting time length of next process.

6. Start with first process, selection as above and other processes are to be in queue.

7. Calculates Burst total number of time.

8. Display the related values.

9. Display now close/Stop process.

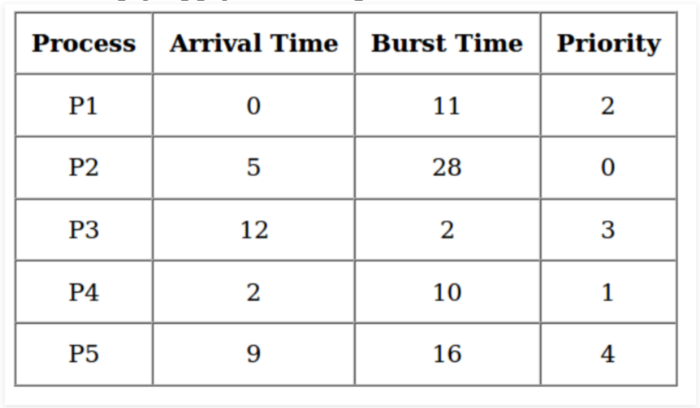
PRACTICAL No.-10

Objective: Write an algorithm of Priority Scheduling for CPU Scheduling.

Theory: Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned first arrival time (less arrival time process first) if two processes have same arrival time, then compare to priorities (highest process first). Also, if two processes have same priority then compare to process number (less process number first). This process is repeated while all process get executed.

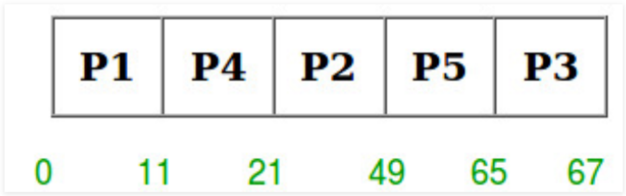
Algorithm:

1. First input the processes with their arrival time, burst time and priority.
2. Sort the processes, according to arrival time if two process arrival time is same then sort according process priority if two process priority are same then sort according to process number.
3. Now simply apply FCFS algorithm.



Process	Arrival Time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

Gantt Chart –



Examples –

```
Input :
process no-> 1 2 3 4 5
arrival time-> 0 1 3 2 4
burst time-> 3 6 1 2 4
priority-> 3 4 9 7 8
Output :
Process_no Start_time Complete_time Trun_Around_Time Wating_Time
1          0          3             3                0
2          3          9             8                2
4          9          11            9                7
3          11         12            9                8
5          12         16            12               8
Average Wating Time is : 5.0
Average Trun Around time is : 8.2
```