# 414454: Lab Practice - V

**Table of Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **List of Assignments** | **Performed**  **Date** | **Completed**  **Date** | **Page No** |
| 1 | Implement multi-threaded client/server Process communication using RMI. |  |  | 02 |
| 2 | Develop any distributed application using CORBA to demonstrate object brokering.  (Calculator or String operations). |  |  | 10 |
| 3 | Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. |  |  | 18 |
| 4 | Implement Berkeley algorithm for clock synchronization. |  |  | 24 |
| 5 | Implement token ring based mutual exclusion algorithm. |  |  | 27 |
| 6 | Implement Bully and Ring algorithm for leader election. |  |  | 30 |
| 7 | Create a simple web service and write any distributed application to consume the web service. |  |  | 47 |
| 8 | Mini Project (In group): A Distributed Application for Interactive Multiplayer Games |  |  | 54 |

1

(fill this into printed index given by college)

# Assignment No. 1

**Aim: Implement multi-threaded client/server Process communication using RMI.**

**Objectives:** To study multi-threaded client/server Process communication

**Course Outcome (CO1):** To understand & Demonstrate knowledge of the core concepts and techniques in distributed systems

# Software Used: Eclipse, Java 8, rmiregistry, Ubuntu Operating System

# Theory:

# REMOTE METHOD INVOCATION (RIM)

# • Remote Method Invocation (RIM) is an API which allows an object to invoke a method of an object hat exists in another address space, which could be on the same machine or on a remote machine.

# • Through RMI, object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side).

# •RMI creates a public remote server object that enables client and server side communications through simple method calls on the server object. The communication between client and server is handled by using two intermediate objects:Stub object (on client side) and Skeleton object (on server side).

# • Stub Object:The stub object on the client machine builds an information block and sends this information to the ever.

# • The block consists of An identifier of the remote object to be used Method name which is to be invoked Parameters to the remote JVM.

# 

# RIM IMPLEMENTATION

# • Skeleton Object

# • The skeleton object passes the request from the stub object to the remote object. It performs following tasks:

# -It calls the desired method on the real object present on the server.

# -It forwards the parameters received from the stub object to the method

# RIM IMPLEMENTATION:

# Steps to implement RMI:

# 1. Defining a remote interface

# 2. Implementing the remote interface

# 3. Creating Stub and Skeleton objects from the implementation class using rmic

# (rmi complier)

# 4. Start the rmiregistry

# 5. Create and execute the server application program

# 6. Create and execute the client application program.

# • Remote interfaces: Every remote object has a remote interface that specifies which of its

# methods can be invoked remotely.

# RIM IMPLEMENTATION:

# • Step 1: Defining the remote interface:

# • To create an interface which will provide the description of the methods that can be invoked by remote clients.

# • This interface should extend the Remote interface and the method prototype within the interface should throw the RemoteException.

# // Creating a Search interface (Search.java)

# import java.rmi.\*;

# public interface Search extends Remote {

# // Declaring the method prototype

# public String query(String search) throws

# RemoteException;

# }

# Step 2: Implementing the remote interface

# • To implement the remote interface, the class should extend to RemoteObject class of java.rmi package.

# // Java program to implement the Search interface (SearchQuery.java)

# import java.rmi.\*;

# import java.rmi.server.\*;

# public class SearchQuery extends RemoteObject

# implements Search

# {

# // Implementation of the query interface

# public String query(String search)

# throws RemoteException

# {

# String result;

# if (search.equals("Reflection in Java"))

# result = "Found";

# else

# result = "Not Found";

# return result; } }

# •Step 3: Creating Stub and Skeleton objects from the implementation class using rmic

# The rmic tool is used to invoke the rmi compiler that creates the Stub and Skeleton objects.

# Its prototype is rmic classname.

# The command need to be executed at the command prompt

# # rmic SearchQuery

# STEP 4: Start the rmiregistry

# Start the registry service by issuing the command at the command prompt :

# # rmiregistry &

# STEP 5: Create and execute the server application program

# • To create the server application program and execute it on a separate command prompt.

# • The server program uses createRegistry method of LocateRegistry class to create

# rmiregistry within the server JVM with the port number passed as argument.

# • The rebind method of Naming class is used to bind the remote object to the new name.

# // A Java program for a Server

# import java.net.\*;

# import java.io.\*;

# public class Server

# {

# //initialize socket and input stream

# private Socket socket = null;

# private ServerSocket server = null;

# private DataInputStream in = null;

# // constructor with port

# public Server(int port)

# {

# // starts server and waits for a connection

# try

# {

# server = new ServerSocket(port);

# System.out.println("Server started");

# System.out.println("Waiting for a client ...");

# socket = server.accept();

# System.out.println("Client accepted");

# // takes input from the client socket

# in = new DataInputStream(

# new BufferedInputStream(socket.getInputStream()));

# String line = "";

# // reads message from client until "Over" is sent

# while (!line.equals("Over"))

# {

# try

# {

# line = in.readUTF();

# System.out.println(line);

# }

# catch(IOException i)

# {

# System.out.println(i);

# }

# }

# System.out.println("Closing connection");

# // close connection

# socket.close();

# in.close();

# }

# catch(IOException i)

# {

# System.out.println(i);

# }

# }

# public static void main(String args[])

# {

# Server server = new Server(5000);

# }

# }

# Step 6: Create and execute the client application program

# • The last step is to create the client application program and execute it on a separate

# command prompt .

# • The lookup method of Naming class is used to get the reference of the Stub object.

# // A Java program for a Client

# import java.io.\*;

# import java.net.\*;

# public class Client {

# // initialize socket and input output streams

# private Socket socket = null;

# private BufferedReader d = null;

# private InputStream input = null;

# private DataOutputStream out = null;

# // constructor to put ip address and port

# public Client(String address, int port)

# {

# // establish a connection

# try {

# socket = new Socket(address, port);

# System.out.println("Connected");

# System.out.println("Done with 1st program of DS");

# // takes input from terminal

# d = new BufferedReader(new InputStreamReader(System.in));

# // sends output to the socket

# out = new DataOutputStream(

# socket.getOutputStream());

# }

# catch (UnknownHostException u) {

# System.out.println(u);

# return;

# }

# catch (IOException i) {

# System.out.println(i);

# return;

# }

# // string to read message from input

# String line = "";

# // keep reading until "Over" is input

# while (!line.equals("Over")) {

# try {

# line = d.readLine();

# out.writeUTF(line);

# }

# catch (IOException i) {

# System.out.println(i);

# }

# }

# // close the connection

# try {

# input.close();

# out.close();

# socket.close();

# }

# catch (IOException i) {

# System.out.println(i);

# }

# }

# public static void main(String args[])

# {

# Client client = new Client("127.0.0.1", 5000);

# }

# }

# Step 7: Compile and execute application programs

# On console-1:

# #javac Search.java

# #javac SearchQuery.java

# #rmic SearchQuery

# #rmiregistry on console

# On console2:

# Compile Server Application:

# #javac Server.java

# #java Server

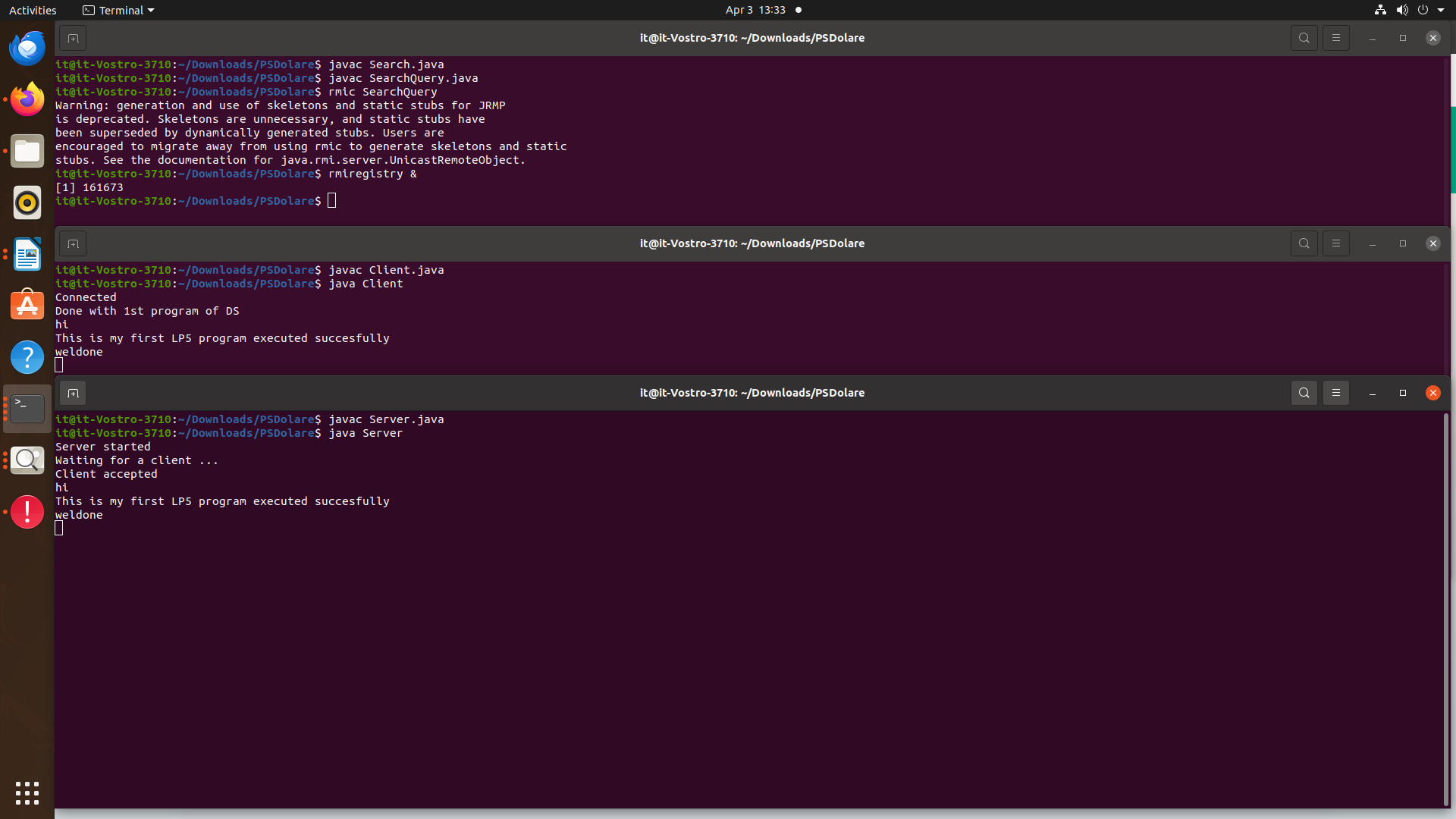
# On console-3:

# Compile Client Application:

# #javac Client.java

# #java Client

# Output:



**Conclusion:** Here we have studied implementation of multi threaded Client /Server process communication using RIM.

# Assignment No. 2

**Title: Develop any distributed application using CORBA to demonstrate object brokering.**

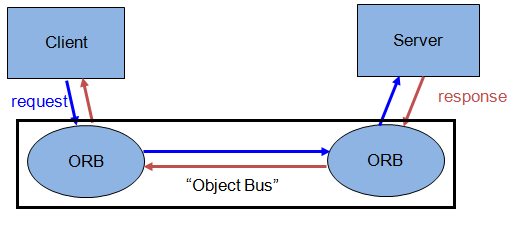
**Aim:** Using CORBA Implement and to demonstrate object brokering. (Calculator or String operations).

**Course Outcome (CO1):** Demonstrate knowledge of the core concepts and techniques in distributed systems.

**Tools : Java 8 with idlj compiler.**

**Theory: CORBA** ( Common Object Request Broker Architecture)

It is a specification for creating distributed objects and NOT a programming language. It promotes design of applications as a set of cooperating objects. Clients are isolated from servers by interface. CORBA objects run on any platform, can be located anywhere on the network and can be written in any language that has IDL mapping.



**1.Define the remote interface:**

Define the interface for the remote object using Interface Definition Langauge (IDL). Use IDL instead of the Java language because the idlj compiler automatically maps from IDL, generating all Java language stub and skeleton source files, along with the infrastructure code for connecting to the ORB.

BUILDING A CORBA DISTRIBUTED APPLICATION

USING JAVA IDL

**1.1.1** Writing Calc.idl

• Create a directory named Calc for this application.

• Create a file named Calc.idl in this directory.

**1.1.2** Understanding the IDL file : Perform 3 steps to write IDL file as follows:

• Declaring the CORBA IDL module: When you compile the IDL, the module statement will

generate a package statement in the Java code.

**[Save Below code as Calc.idl]**

**module CalcApp**

**{**

**interface Calc**

**{**

**exception DivisionByZero {};**

**float sum(in float a, in float b);**

**float div(in float a, in float b) raises (DivisionByZero);**

**float mul(in float a, in float b);**

**float sub(in float a, in float b);**

**};**

**};**

**2. Compile the remote interface:** When you run the idlj compiler the interface definition file, it generates the Java version of the interface, as well as the class code files for the stubs and skeletons that enable your applications to hook into the ORB.

**2.1.** Mapping  **Calc.idl** to Java:

• The tool idlj reads IDL files and creates the required Java files. The idlj compiler defaults to generating only the client-side bindings. If you need both client-side bindings and server-side skeletons, use the -fall option when running the idlj compiler.

• Enter compiler command on command line prompt having path to the java/bin directory:

idlj -fall  **Calc.idl**

• If you list the contents of the directory, you will see that six files are created.

**2.2** Understanding the idlj Compiler Output

• The files generated by the idlj compiler for  **Calc.idl** are:

i.  **Calc**POA.java : This abstract class is the skeleton, providing basic CORBA functionality for the server. The server class  **Calc**Impl extends **Calc**POA.

An object adapter is the mechanism that connects a request using an object reference with the proper code to service that request.

ii. \_**Calc**Stub.java : This class is the client stub providing CORBA functionality for the client.

It implements the **Calc.**java interface.

iii.**Calc**.java : This interface contains the Java version of IDL interface. The Hello.java interface extends org.omg.CORBA.Object, providing standard

CORBA object functionality.

IV. **Calc**Helper.java : This class provides additional functionality , the narrow()

method required to cast CORBA object references to their proper types. The Helper

class is responsible for reading and writing the data type to CORBA streams. The

Holder class uses the methods in the Helper class for reading and writing.

V.**Calc**Holder.java: It provides operations for OutputStream and InputStream. It

provides operations for out and inout arguments, which CORBA allows, but which do

not map easily to Java‘s semantics.

VI. **Calc**Operations.java : This operations interface contains the single methods

SayHello(). The IDL-to-Java mapping puts all of the operations defined on the IDL

interface into this file.

**3. Implement the Server:** Once you run the idlj compiler, you can use the skeletons it

generates to put together your server application. In addition to implementing the

methods of the remote interface, the server code includes a mechanism to start the ORB

and wait for invocation from a remote client.

**[Save Below code as CalcServer.java]**

**import CalcApp.\*;**

**import CalcApp.CalcPackage.DivisionByZero;**

**import org.omg.CosNaming.\*;**

**import org.omg.CosNaming.NamingContextPackage.\*;**

**import org.omg.CORBA.\*;**

**import org.omg.PortableServer.\*;**

**import java.util.Properties;**

**class CalcImpl extends CalcPOA {**

**@Override**

**public float sum(float a, float b) {**

**return a + b;**

**}**

**@Override**

**public float div(float a, float b) throws DivisionByZero {**

**if (b == 0) {**

**throw new CalcApp.CalcPackage.DivisionByZero();**

**} else {**

**return a / b;**

**}**

**}**

**@Override**

**public float mul(float a, float b) {**

**return a \* b;**

**}**

**@Override**

**public float sub(float a, float b) {**

**return a - b;**

**}**

**private ORB orb;**

**public void setORB(ORB orb\_val) {**

**orb = orb\_val;**

**}**

**}**

**public class CalcServer {**

**public static void main(String args[]) {**

**try {**

**// create and initialize the ORB**

**ORB orb = ORB.init(args, null);**

**// get reference to rootpoa & activate the POAManager**

**POA rootpoa = POAHelper.narrow(orb.resolve\_initial\_references("RootPOA"));**

**rootpoa.the\_POAManager().activate();**

**// create servant and register it with the ORB**

**CalcImpl helloImpl = new CalcImpl();**

**helloImpl.setORB(orb);**

**// get object reference from the servant**

**org.omg.CORBA.Object ref = rootpoa.servant\_to\_reference(helloImpl);**

**Calc href = CalcHelper.narrow(ref);**

**// get the root naming context**

**// NameService invokes the name service**

**org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");**

**// Use NamingContextExt which is part of the Interoperable**

**// Naming Service (INS) specification.**

**NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);**

**// bind the Object Reference in Naming**

**String name = "Calc";**

**NameComponent path[] = ncRef.to\_name(name);**

**ncRef.rebind(path, href);**

**System.out.println("Ready..");**

**// wait for invocations from clients**

**orb.run();**

**} catch (Exception e) {**

**System.err.println("ERROR: " + e);**

**e.printStackTrace(System.out);**

**}**

**System.out.println("Exiting ...");**

**}**

**}**

**4. Developing a Client Application:**Use the stubs generated by the idlj compiler as the basis of client application. The client code builds on the stubs to start its ORB, look up the server using the name service provided with Java IDL, obtain a reference for the remote object, and call its method.

**[Save Below code as CalcClient.java]**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import CalcApp.\*;

import CalcApp.CalcPackage.DivisionByZero;

import org.omg.CosNaming.\*;

import org.omg.CosNaming.NamingContextPackage.\*;

import org.omg.CORBA.\*;

import static java.lang.System.out;

public class CalcClient {

static Calc calcImpl;

static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

public static void main(String args[]) {

try {

// create and initialize the ORB

ORB orb = ORB.init(args, null);

// get the root naming context

org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");

// Use NamingContextExt instead of NamingContext. This is

// part of the Interoperable naming Service.

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

// resolve the Object Reference in Naming

String name = "Calc";

calcImpl = CalcHelper.narrow(ncRef.resolve\_str(name));

// System.out.println(calcImpl);

while (true) {

out.println("1. Sum");

out.println("2. Sub");

out.println("3. Mul");

out.println("4. Div");

out.println("5. exit");

out.println("--");

out.println("choice: ");

try {

String opt = br.readLine();

if (opt.equals("5")) {

break;

} else if (opt.equals("1")) {

out.println("a+b= " + calcImpl.sum(getFloat("a"), getFloat("b")));

} else if (opt.equals("2")) {

out.println("a-b= " + calcImpl.sub(getFloat("a"), getFloat("b")));

} else if (opt.equals("3")) {

out.println("a\*b= " + calcImpl.mul(getFloat("a"), getFloat("b")));

} else if (opt.equals("4")) {

try {

out.println("a/b= " + calcImpl.div(getFloat("a"), getFloat("b")));

} catch (DivisionByZero de) {

out.println("Division by zero!!!");

}

}

} catch (Exception e) {

out.println("===");

out.println("Error with numbers");

out.println("===");

}

out.println("");

}

//calcImpl.shutdown();

} catch (Exception e) {

System.out.println("ERROR : " + e);

e.printStackTrace(System.out);

}

}

static float getFloat(String number) throws Exception {

out.print(number + ": ");

return Float.parseFloat(br.readLine());

}

}

**At 1st Console**

1) idlj -fall Calc.idl

2) javac \***.**java CalcApp**/\***.java or javac -Xlint **.**java CalcApp**/**.java

3) orbd -ORBInitialPort 1050&

**At 2nd Console**

1) javac CalcServer.java

2) java CalcServer -ORBInitialPort 1050&

**At 3rd Console**

1) javac CalcClient.java

2) java CalcClient -ORBInitialPort 1050

# Output:

# 

**Conclusion:** Here we have studied Implementation object brokering using CORBA .

# Assignment No.3

**Title**: Develop a distributed system, to find sum of N elements in an array by distributing N/n

elements to n number of processors MPI or OpenMP.

**Aim:** Develop a distributed system, demonstrate by displaying the intermediate sums calculated at different processors MPI .

**Course Outcome (CO1):** Demonstrate knowledge of the core concepts and techniques in distributed systems.

# Theory:

# MPI stands for Message Passing Interface. It is used to make communication between different processes in the same machine or across different machines in a network in a distributed environment. So that the task can be parallelized and work can be done faster. The MPI has made the parallelizing tasks very easy. It is available in the form of a standard library. The parallelism between different processes can be achieved with help of a rank.

The MPI library assigns the different processes with a unique number called a rank. The rank starts from 0 to n-1. If there are 4 processes then each process will be assigned a unique rank ranging from 0,1,2, and 3. These ranks can be used to differentiate and communicate between the different processes.

**Installation of OPENMPI:**

1. Download openmpi-4.1.4.tar.bz2 from http://www.open-mpi.org in a folder say LP5.

# 2. Goto the terminal (Command prompt)

# 3. update using sudo apt-get update

# sudo apt install gcc {if not already installed}

# 4. Goto the directory which contains the downloaded file

# 5. Extract the files using tar -jxf openmpi-4.1.4.tar.bz2

# 6. The directory openmpi-4.1.4 is created

# 7. Configure, compile and install by executing the following commands

# ./configure --prefix=$HOME/opt/openmpi

# make all

# make install

# 8. Now openmpi folder is created in ‘opt‘ folder of Home directory.

# 9. Now the folder LP5 can be deleted (optional)

# 10. Update the PATH and LD\_LIBRARY\_PATH environment variable using

# echo "export PATH=\$PATH:\$HOME/opt/openmpi/bin" >> $HOME/.bashrc

# echo "export LD\_LIBRARY\_PATH=\$LD\_LIBRARY\_PATH:\$HOME/opt/openmpi/lib">>$HOME/.bashrc

# 11. Compile the program using

# mpicc name of the program

# 12. Execute the program using

# mpirun -np N ./a.outHello world program

# nllabc2d22@nllabc2d-22:~/opt/openmpi/bin$ gedit hello.c

# #include <stdio.h>

# #include "mpi.h"

# int main(int argc, char\* argv[])

# {

# int rank, size, len;

# MPI\_Init(&argc, &argv);

# MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

# MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

# printf("Hello, world, I am %d of %d\n",rank, size);

# MPI\_Finalize();

# return 0;

# }

# Compile the program

# nllabc2d22@nllabc2d-22:~/opt/openmpi/bin$ mpicc hello.c

# Execute the program using 2 cores

# nllabc2d22@nllabc2d-22:~/opt/openmpi/bin$ mpirun -np 2 ./a.out

# Hello, world, I am 0 of 2

# Hello, world, I am 1 of 2

# Execute the program using 4 cores

# nllabc2d22@nllabc2d-22:~/opt/openmpi/bin$ mpirun -np 4 ./a.out

# Hello, world, I am 0 of 4

# Hello, world, I am 3 of 4

# Hello, world, I am 1 of 4

# Hello, world, I am 2 of 4

# Program to transfer data from core 0 to core 1.

# #include <stdio.h>

# #include "mpi.h"

# int main(int argc, char\* argv[])

# {

# int rank, size, len;

# int num=10;

# MPI\_Init(&argc, &argv);

# MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

# MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

# if(rank == 0)

# {

# printf("Sending message containing: %d from rank %d\n", num,rank);

# MPI\_Send(&num, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);

# }

# else

# {

# printf(" at rank %d\n",rank);

# MPI\_Recv(&num, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

# printf("Received message containing: %d at rank %d\n", num,rank);

# }

# MPI\_Finalize();

# return 0;

# }

# Sending message containing: 10 from rank 0

# at rank 1

# at rank 3

# Received message containing: 10 at rank 1

# at rank 2

# /\*\*\*\*\*\* The cores 2 and will be in waiting mode … Press Ctrl+z to end the execution

# \*\*\*\*\*\*\*/Assignment program: Add 20 numbers in an array using 4 cores

# #include <stdio.h>

# #include "mpi.h"

# int main(int argc, char\* argv[])

# {

# int rank, size;

# int num[20]; //N=20, n=4

# MPI\_Init(&argc, &argv);

# MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

# MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

# for(int i=0;i<20;i++)

# num[i]=i+1;

# if(rank == 0){

# int s[4];

# printf("Distribution at rank %d \n", rank);

# for(int i=1;i<4;i++)

# MPI\_Send(&num[i\*5], 5, MPI\_INT, i, 1, MPI\_COMM\_WORLD); //N/n i.e. 20/4=5

# int sum=0, local\_sum=0;

# for(int i=0;i<5;i++)

# {

# local\_sum=local\_sum+num[i];

# }

# for(int i=1;i<4;i++)

# {

# MPI\_Recv(&s[i], 1, MPI\_INT, i, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

# }

# printf("local sum at rank %d is %d\n", rank,local\_sum);

# sum=local\_sum;

# for(int i=1;i<4;i++)

# sum=sum+s[i];

# printf("final sum = %d\n\n",sum);

# } else {

# int k[5];

# MPI\_Recv(k, 5, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

# int local\_sum=0;

# for(int i=0;i<5;i++)

# {

# local\_sum=local\_sum+k[i];

# }

# printf("local sum at rank %d is %d\n", rank, local\_sum);

# MPI\_Send(&local\_sum, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);

# }

# MPI\_Finalize();

# return 0;

# }

# Distribution at rank 0

# local sum at rank 1 is 40

# local sum at rank 2 is 65

# local sum at rank 3 is 90

# local sum at rank 0 is 15

# final sum = 210

# Output:

# 

# Conclusion:Here we studied to develop a distributed system, displaying the intermediate sums calculated at different processors .

**Assignment No. 4**

**Aim**: Write a program to implement Berkeley Clock Synchronization.

**Objective:**Demonstrate knowledge of the core concepts and techniques in distributed systems.

**Course Outcome (CO2):** Learn how to apply principles of state-of-the-Art Distributed systems in practical application.

**Theory:**

Berkeley’s Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn’t have an accurate time source or doesn’t possess an UTC server.

Algorithm: An individual node is chosen as the master node from a pool nodes in the network. This node is the main node in the network which acts as a master and rest of the nodes act as slaves. Master node is chosen using a election process/leader election algorithm. Master node periodically pings slaves nodes and fetches clock time at them using Cristian’s algorithm. Master node calculates average time difference between all the clock times received and the clock time given by master’s system clock itself. This average time difference is added to the current time at master’s system clock and broadcasted over the network.

Scope of Improvement : Improvision in accuracy of cristian’s algorithm. Ignoring significant outliers in calculation of average time difference In case master node fails/corrupts, a secondary leader must be ready/pre-chosen to take the place of the master node to reduce downtime caused due to master’s unavailability. Instead of sending the synchronized time, master broadcasts relative inverse time difference, which leads to decrease in latency induced by traversal time in the network while time of calculation at slave node.

24

**[Save Below code as BerkeleyClockSync.java]**

import java.util.ArrayList;

public class BerkeleyClockSync {

public static void main(String[] args) {

// Initialize the system clocks

int[] systemClocks = { 10, 12, 13, 11, 14 };

int masterClock = 0;

// Print the initial system clocks

System.out.print("System clocks: ");

for (int clock : systemClocks) {

System.out.print(clock + " ");

}

System.out.println();

// Calculate the average system clock

int sum = 0;

for (int clock : systemClocks) {

sum += clock;

}

int averageClock = sum / systemClocks.length;

// Calculate the time difference for each system clock

ArrayList<Integer> timeDifferences = new ArrayList<>();

for (int clock : systemClocks) {

timeDifferences.add(averageClock - clock);

}

// Calculate the time adjustment for the master clock

int timeAdjustment = 0;

for (int difference : timeDifferences) {

timeAdjustment += difference;

}

timeAdjustment /= timeDifferences.size();

// Update the master clock

masterClock = averageClock - timeAdjustment;

// Print the updated system clocks and master clock

System.out.print("Updated system clocks: ");

for (int clock : systemClocks) {

System.out.print((clock - timeAdjustment) + " ");

}

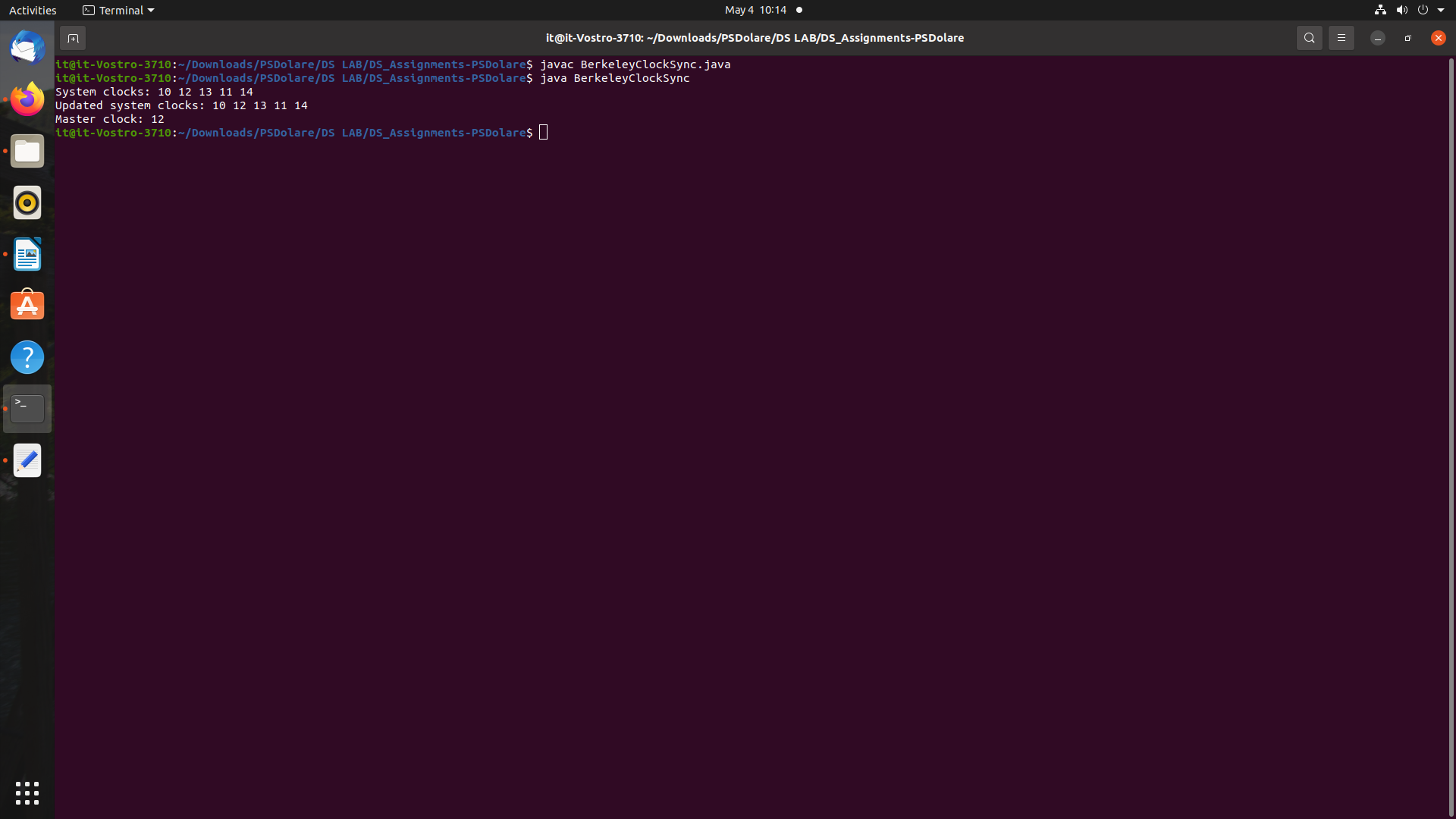
System.out.println();

System.out.println("Master clock: " + masterClock);

}

}

**Output:**



# Conclusion:Here we studied to develop a distributed system for Berkeley Clock Synchronization.

26

**Assignment No. 5**

**Aim**: Implement Mutual Exclusion using Token Ring.

**Objective:** Demonstrate knowledge of the core concepts and techniques in distributed systems.

**Course Outcome (CO3):** Design, build and test application programs on distributed systems

**Theory:**

Token Ring algorithm achieves mutual exclusion in a distributed system by creating a bus network of processes. A logical ring is constructed with these processes and each process is assigned a position in the ring. Each process knows who is next in line after itself. When them ring is initialized, process 0 is given a token. The token circulates around the ring. When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a critical

region. If so, the process enters the region, does all the work it needs to, and leaves the region.

After it has exited, it passes the token to the next process in the ring. It is not allowed to enter

the critical region again using the same token. If a process is handed the token by its neighbor

and is not interested in entering a critical region, it just passes the token along to the next process.

• Advantages:

o The correctness of this algorithm is evident. Only one process has the token at any instant, so only one process can be in a CS

o Since the token circulates among processes in a well-defined order, starvation cannot occur.

• Disadvantages

o Once a process decides it wants to enter a CS, at worst it will have to wait for every other process to enter and leave one critical region.

o If the token is ever lost, it must be regenerated. In fact, detecting that it is lost is difficult, since the amount of time between successive appearances of the token on the network is not a constant. The fact that the token has not been spotted for an hour does not mean that it has been lost; some process may still be using it.

o The algorithm also runs into trouble if a process crashes, but recovery is easier than in the other cases. If we require a process receiving the token to acknowledge receipt, a dead process will be detected when its neighbor tries to give it the token and fails. At that point the dead process can be removed from the group, and the token holder can pass the token to the next member down the line

27

## Code (save this code as tokenring.java)

import java.io.\*;

import java.util.\*;

class tokenring {

public static void main(String args[]) throws Throwable {

Scanner scan = new Scanner(System.in);

System.out.println("Enter the num of nodes:");

int n = scan.nextInt();

int m = n - 1;

// Decides the number of nodes forming the ring

int token = 0;

int ch = 0, flag = 0;

for (int i = 0; i < n; i++) {

System.out.print(" " + i);

}

System.out.println(" " + 0);

do{

System.out.println("Enter sender:");

int s = scan.nextInt();

System.out.println("Enter receiver:");

int r = scan.nextInt();

System.out.println("Enter Data:");

int a;

a = scan.nextInt();

System.out.print("Token passing:");

for (int i = token, j = token; (i % n) != s; i++, j = (j + 1) % n) {

System.out.print(" " + j + "->");

}

System.out.println(" " + s);

System.out.println("Sender " + s + " sending data: " + a);

for (int i = s + 1; i != r; i = (i + 1) % n) {

System.out.println("data " + a + " forwarded by " + i);

}

System.out.println("Receiver " + r + " received data: " + a +"\n");

token = s;

do{

try {

if( flag == 1)

System.out.print("Invalid Input!!...");

System.out.print("Do you want to send again?? enter 1 for Yes and 0 for No : ");

ch = scan.nextInt();

if( ch != 1 && ch != 0 )

flag = 1;

else

flag = 0;

} catch (InputMismatchException e){

System.out.println("Invalid Input");

}

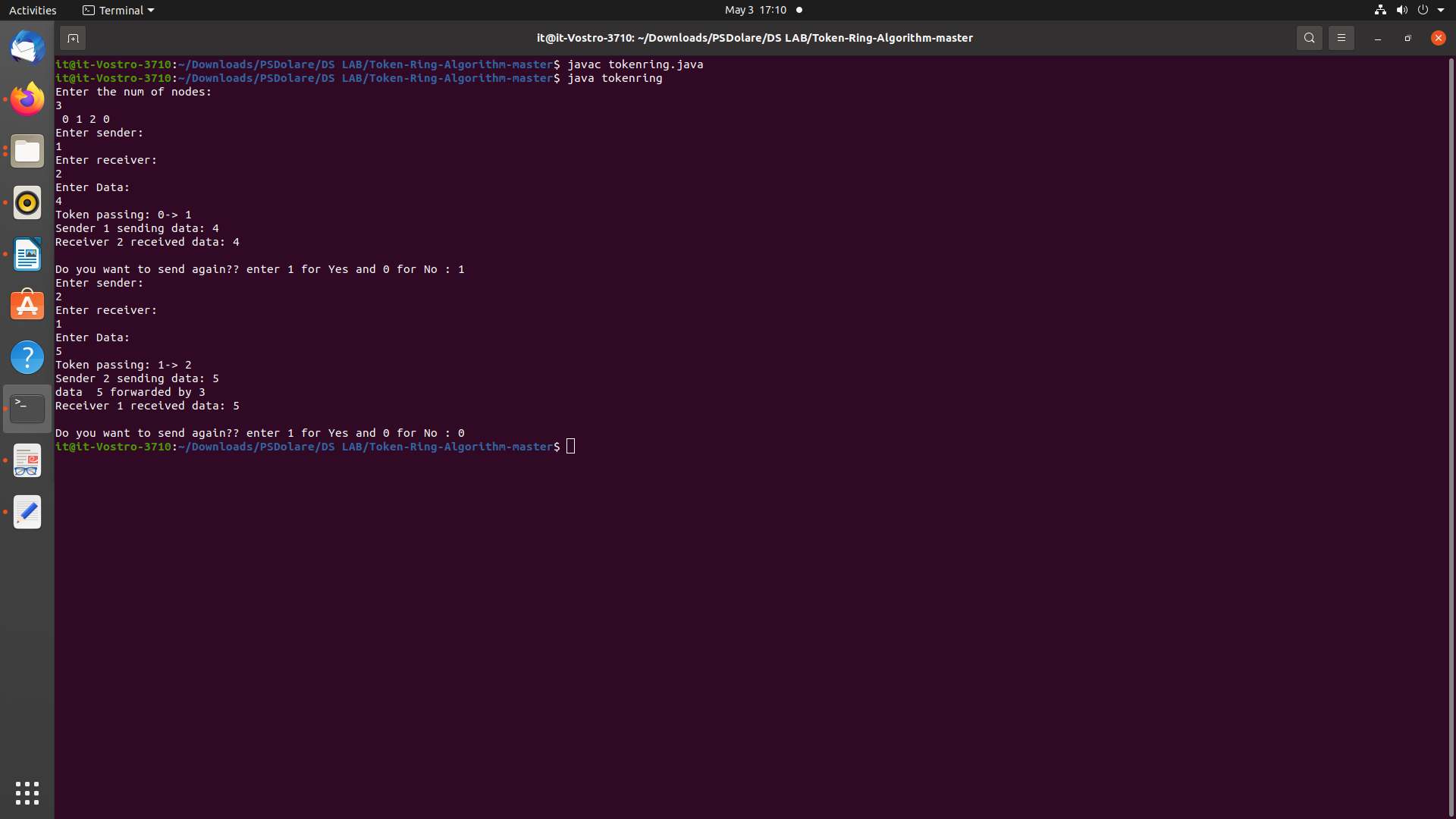
}while( ch != 1 && ch != 0 );

}while( ch == 1 );

}

}

**OutPut:**



# Conclusion:Here we studied to develop a distributed system Mutual Exclusion using Token Ring.

30

**Assignment No. 6**

**Aim**: Write a program to implement Bully Election Algorithm.

**Objective:** Demonstrate knowledge of the core concepts and techniques in distributed systems.

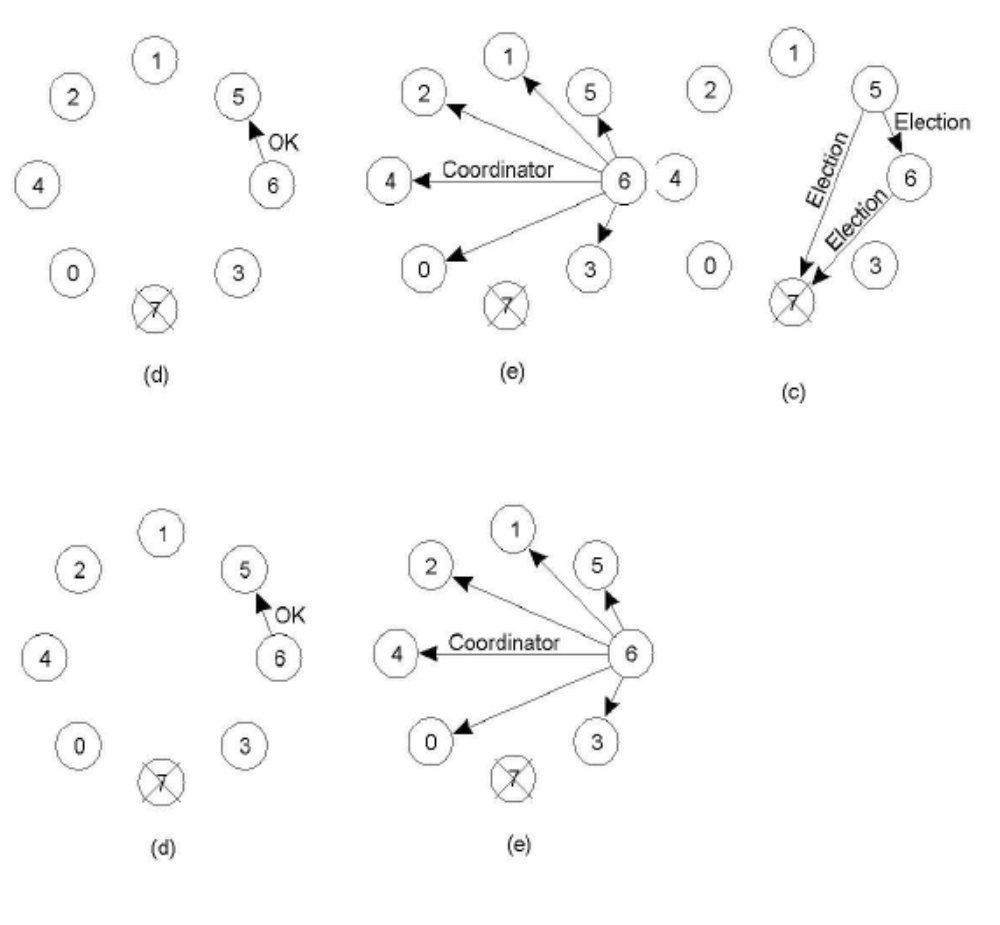
**Course Outcome (CO2):** Learn how to apply principles of state-of-the-Art Distributed systems in practical application.

## Theory: Election Algorithms

Election algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Election algorithm basically determines where a new copy of coordinator should be restarted. Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then, this number is sent to every active process in the distributed system.

**Ring Election Algorithm**

Another election algorithm is based on the use of a ring, but without a token. We assume that the processes are physically or logically ordered, so that each process knows who its successor is. When any process notices that the coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down, the sender skips over the successor and goes to the next member along the ring, or the one after that, until a running process is located. At each step, the sender adds its own process number to the list in the message. Eventually, the message gets back to the process that started it all. That process recognizes this event whe it receives an incoming message containing its own process number. At that point, the message type is changed to COORDINATOR and circulated once again, this time to inform everyone else who the coordinator is (the list member with the highest number) and who the members of the new ring are. When this message has circulated once, it is removed and everyone goes back to work.

Figure 1: Bully Election

## Code (save this code as Bully.java)

## import java.io.InputStream;

import java.io.PrintStream;

import java.util.Scanner;

public class Bully

{

static boolean[] state = new boolean[5];

int coordinator;

public static void up(int up)//4

{

if (state[up - 1])// 0 1 2 3 4

{

System.out.println("process" + up + "is already up");

}

else

{

int i;

Bully.state[up - 1] = true;

System.out.println("process " + up + "held election");

for (i = up; i < 5; ++i)

{

System.out.println("election message sent from process" + up + "to process" + (i + 1));

}

for (i = up + 1; i <= 5; ++i)

{

if (!state[i - 1]) continue;

System.out.println("alive message send from process" + i + "to process" + up);

break;

}

}

}

public static void down(int down)

{

if (!state[down - 1])

{

System.out.println("process " + down + "is already dowm.");

}

else

{

Bully.state[down - 1] = false;

}

}

public static void mess(int mess)

{

if (state[mess - 1])

{

if (state[4])

{

System.out.println("0K");

}

else if (!state[4])

{

int i;

System.out.println("process" + mess + "election");

for (i = mess; i < 5; ++i)

{

System.out.println("election send from process" + mess + "to process " + (i + 1));

}

for (i = 5; i >= mess; --i)

{

if (!state[i - 1]) continue;

System.out.println("Coordinator message send from process" + i + "to all");

break;

}

}

}

else

{

System.out.println("Prccess" + mess + "is down");

}

}

public static void main(String[] args)

{

int choice;

Scanner sc = new Scanner(System.in);

for (int i = 0; i < 5; ++i)

{

Bully.state[i] = true;

}

System.out.println("5 active process are:");

System.out.println("Process up = p1 p2 p3 p4 p5");

System.out.println("Process 5 is coordinator");

do

{

System.out.println(".........");

System.out.println("1 up a process.");

System.out.println("2.down a process");

System.out.println("3 send a message");

System.out.println("4.Exit");

choice = sc.nextInt();

switch (choice)

{

case 1:

{

System.out.println("bring proces up");

int up = sc.nextInt();

if (up == 5)

{

System.out.println("process 5 is co-ordinator");

Bully.state[4] = true;

break;

}

Bully.up(up);

break;

}

case 2:

{

System.out.println("bring down any process.");

int down = sc.nextInt();

Bully.down(down);

break;

}

case 3:

{

System.out.println("which process will send message");

int mess = sc.nextInt();

Bully.mess(mess);

}

}

} while (choice != 4);

}

}

## Code (save this code as Ring.java)

import java.util.Scanner;

public class Ring

{

public static void main(String[] args)

{

// TODO Auto-generated method stub

int temp, i, j;

char str[] = new char[10];

Rr proc[] = new Rr[10];

// object initialisation

for (i = 0; i < proc.length; i++)

proc[i] = new Rr();

// scanner used for getting input from console

Scanner in = new Scanner(System.in);

System.out.println("Enter the number of process : ");

int num = in.nextInt();

// getting input from users

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

{

proc[i].index = i;

System.out.println("Enter the id of process : ");

proc[i].id = in.nextInt();

proc[i].state = "active";

proc[i].f = 0;

}

// sorting the processes from on the basis of id

for (i = 0; i < num - 1; i++)

{

for (j = 0; j < num - 1; j++)

{

if (proc[j].id > proc[j + 1].id)

{

temp = proc[j].id;

proc[j].id = proc[j + 1].id;

proc[j + 1].id = temp;

}

}

}

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

{

System.out.print(" [" + i + "]" + " " + proc[i].id);

}

int init;

int ch;

int temp1;

int temp2;

int ch1;

int arr[] = new int[10];

proc[num - 1].state = "inactive";

System.out.println("\n process " + proc[num - 1].id + "select as co-ordinator");

while (true)

{

System.out.println("\n 1.election 2.quit ");

ch = in.nextInt();

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

{

proc[i].f = 0;

}

switch (ch)

{

case 1:

System.out.println("\n Enter the Process number who initialsied election : ");

init = in.nextInt();

temp2 = init;

temp1 = init + 1;

i = 0;

while (temp2 != temp1)

{

if ("active".equals(proc[temp1].state) && proc[temp1].f == 0)

{

System.out.println("\nProcess " + proc[init].id + " send message to " + proc[temp1].id);

proc[temp1].f = 1;

init = temp1;

arr[i] = proc[temp1].id;

i++;

}

if (temp1 == num)

{

temp1 = 0;

}

else

{

temp1++;

}

}

System.out.println("\nProcess " + proc[init].id + " send message to " + proc[temp1].id);

arr[i] = proc[temp1].id;

i++;

int max = -1;

// finding maximum for co-ordinator selection

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

{

if (max < arr[j])

{

max = arr[j];

}

}

// co-ordinator is found then printing on console

System.out.println("\n process " + max + "select as co-ordinator");

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

{

if (proc[i].id == max)

{

proc[i].state = "inactive";

}

}

break;

case 2:

System.out.println("Program terminated ...");

return ;

default:

System.out.println("\n invalid response \n");

break;

}

}

}

}

class Rr

{

public int index; // to store the index of process

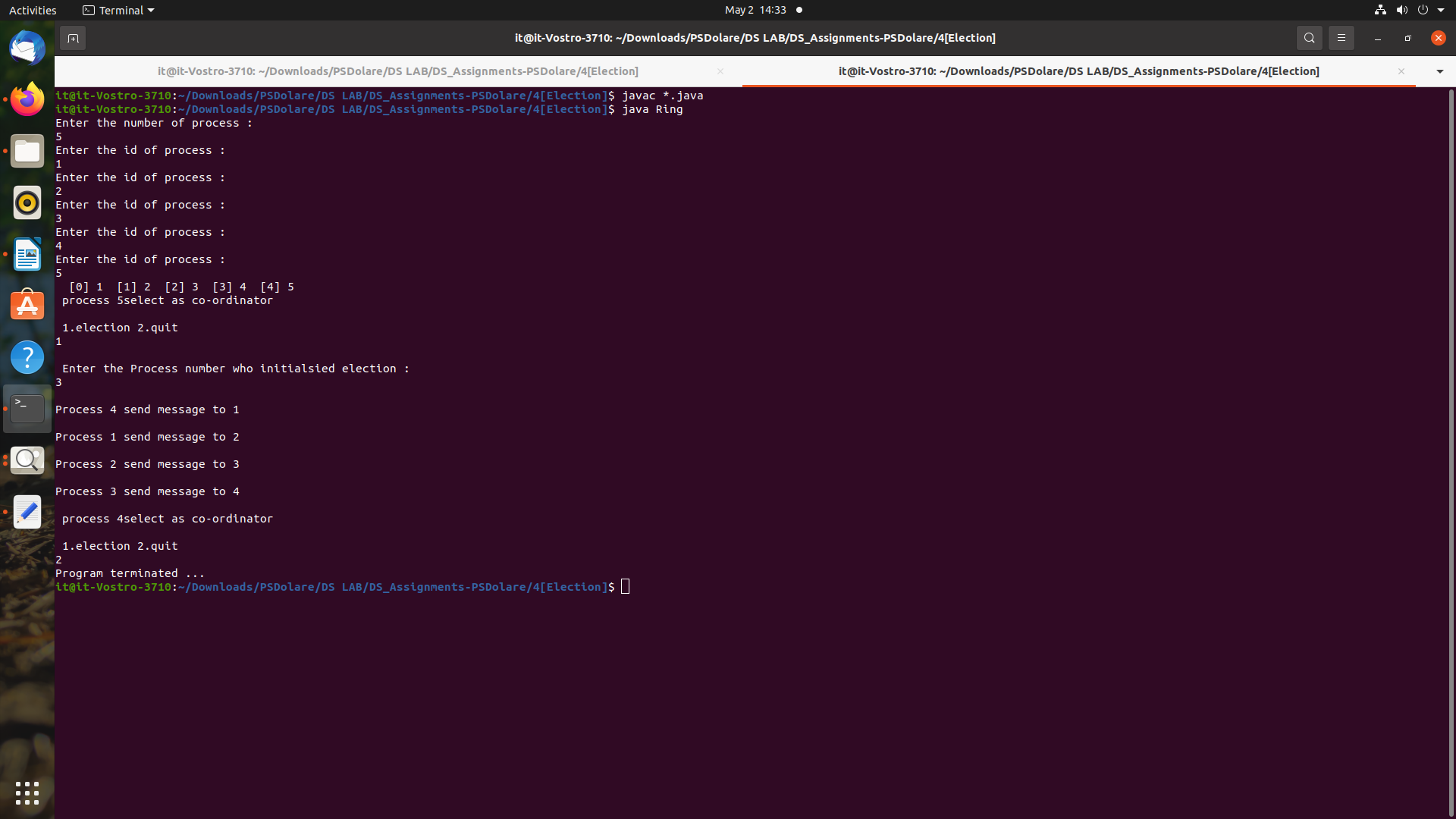
public int id; // to store id/name of process

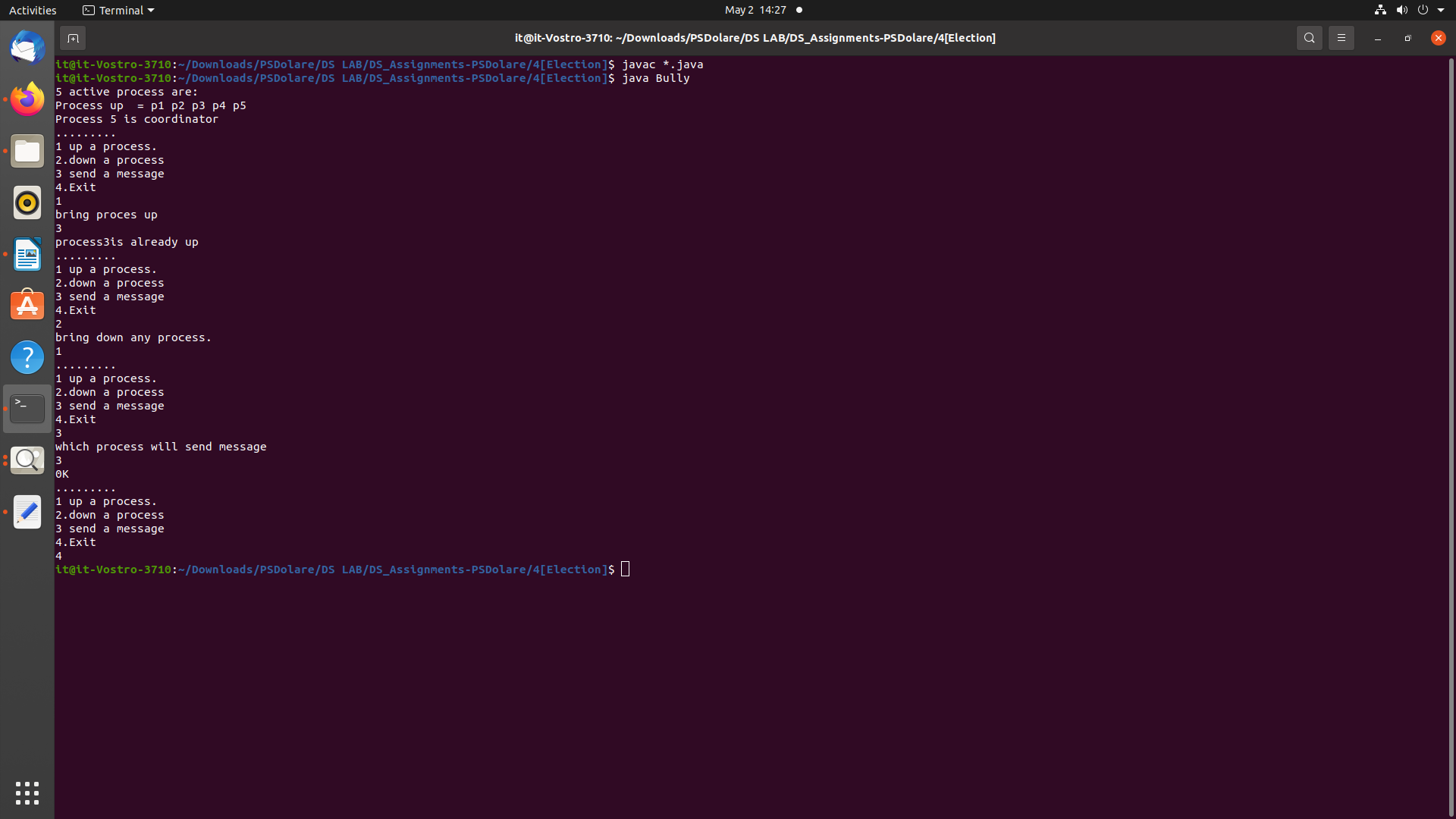
public int f;

String state; // indiactes whether active or inactive state of node

}

**OutPut:**





**Conclusion:** Here we studied Bully Ring Election Algorithm implementation.