EXPERIMENT No. 1

Date of Performance:

Date of Submission:

Aim: To implement Inter process Communication.

Theory: Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:
1. Shared Memory
2. Message passing

Interprocess communication in distributed environment provides both Datagram and stream communication.
Examples Of Interprocess Communication:

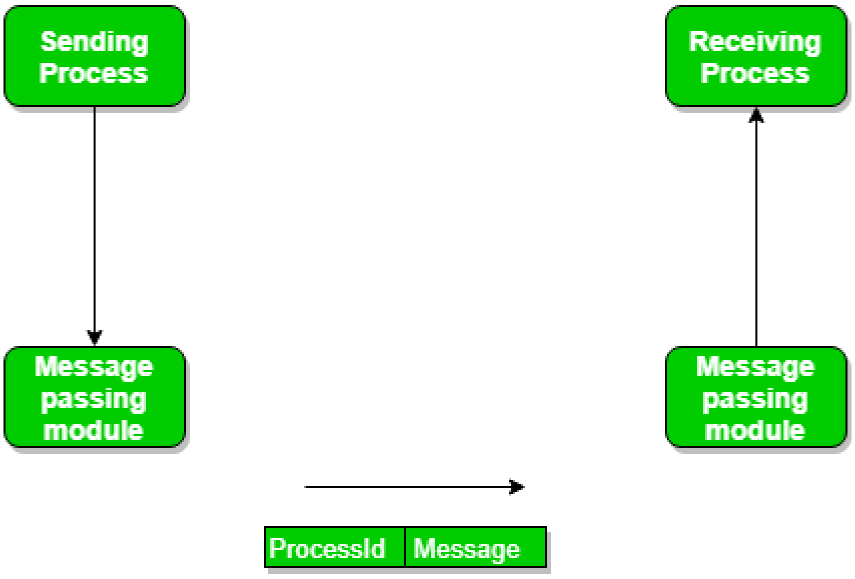
1. N number of applications can communicate with the X server through network protocols.
2. Servers like Apache spawn child processes to handle requests.
3. Pipes are a form of IPC: grep foo

file | sort It has two functions:

1. **Synchronization:**
   Exchange of data is done synchronously which means it has a single clock pulse.
2. **Message Passing:**
   When processes wish to exchange information. Message passing takes several forms such as: pipes, FIFO, Shared Memory, and Message Queues.

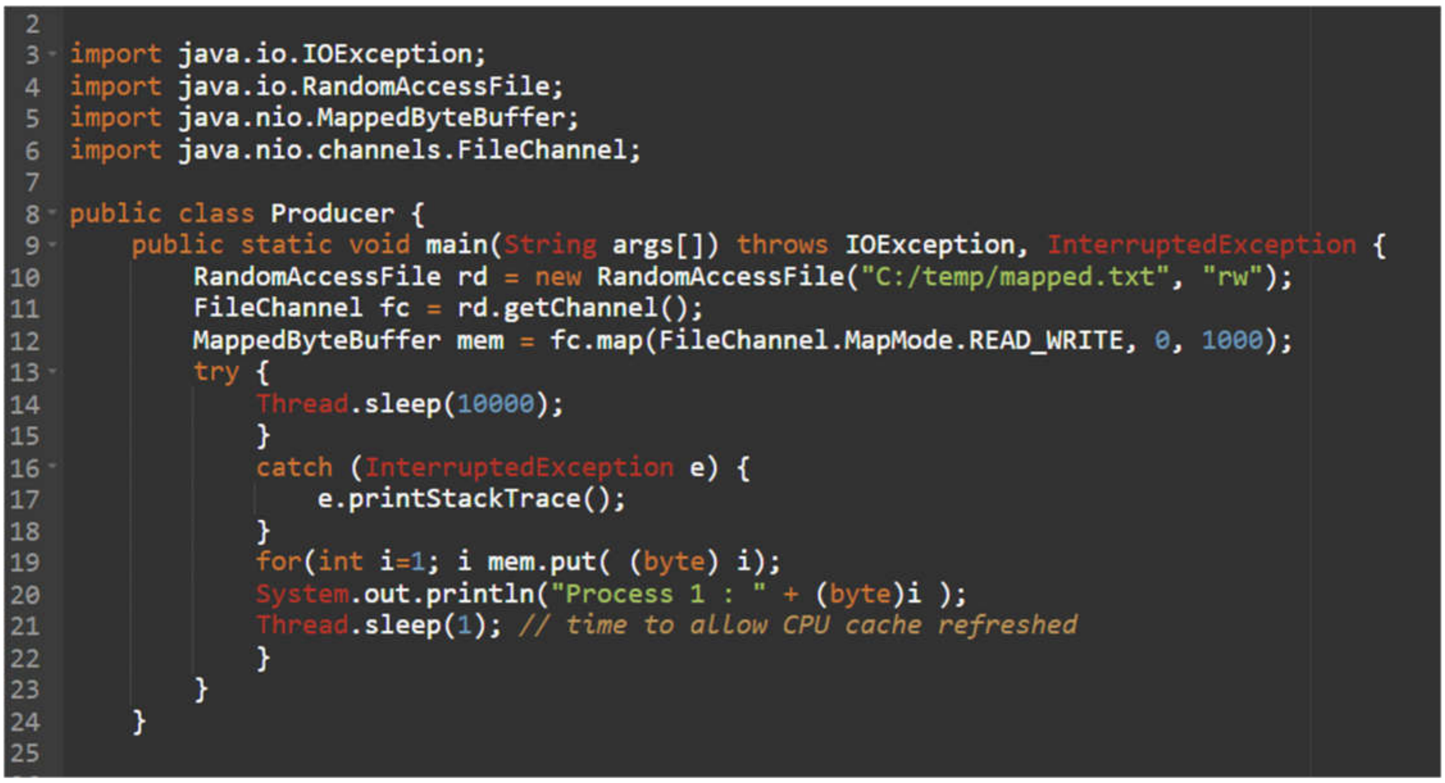
Messaging Passing Method:

3. If two processes p1 and p2 want to communicate with each other, they proceed as follow:
4. Establish a communication link (if a link already exists, no need to establish it again.)
5. Start exchanging messages using basic primitives. We need at least two primitives:
   – send (message, destination) or send(message)
   – receive (message, host) or receive(message)
6. The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for the OS designer but complicated for programmers and if it is of variable size then it is easy for programmers but complicated for the OS designer. A standard message can have two parts: header and body.
   The header part is used for storing Message type, destination id, source id, message length and control information. The control information contains information like what to do if it runs out of buffer space, sequence number, priority.



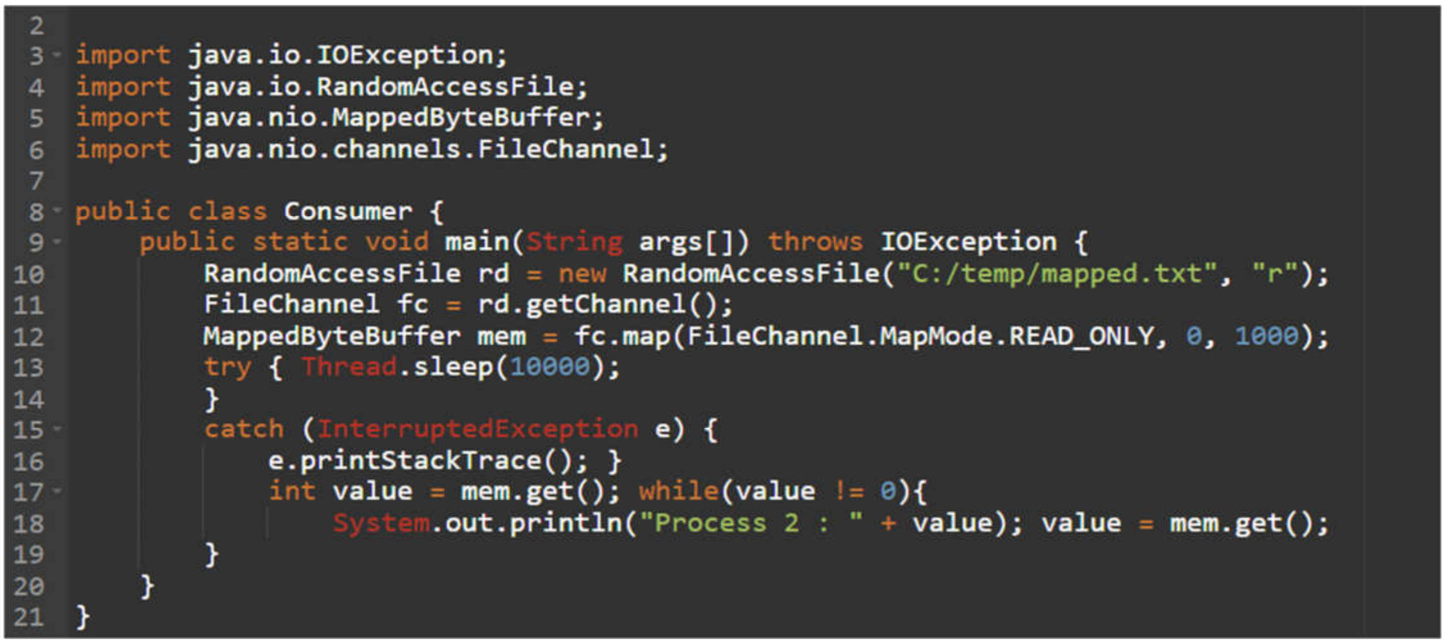
CODE:

Process 1 which is producing data:



```

import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class Producer {
    public static void main(String args[]) throws IOException, InterruptedException {
        RandomAccessFile rd = new RandomAccessFile("C:/temp/mapped.txt", "rw");
        FileChannel fc = rd.getChannel();
        MappedByteBuffer mem = fc.map(FileChannel.MapMode.READ_WRITE, 0, 1000);
        try {
            Thread.sleep(10000);
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
            for(int i=1; i mem.put( (byte) i);
            System.out.println("Process 1 : " + (byte)i );
            Thread.sleep(1); // time to allow CPU cache refreshed
            }
        }
    }

```

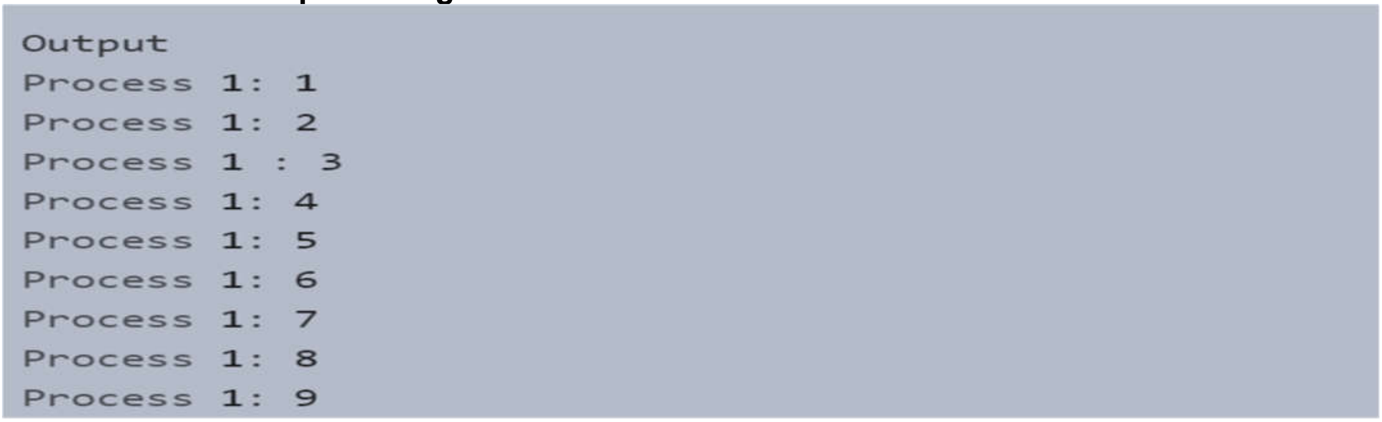
Process 2, which is consuming data



```java

import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class Consumer {
    public static void main(String args[]) throws IOException {
        RandomAccessFile rd = new RandomAccessFile("C:/temp/mapped.txt", "r");
        FileChannel fc = rd.getChannel();
        MappedByteBuffer mem = fc.map(FileChannel.MapMode.READ_ONLY, 0, 1000);
        try { Thread.sleep(10000);
        }
        catch (InterruptedException e) {
            e.printStackTrace(); }
            int value = mem.get(); while(value != 0){
                System.out.println("Process 2 : " + value); value = mem.get();
        }
    }
}
```

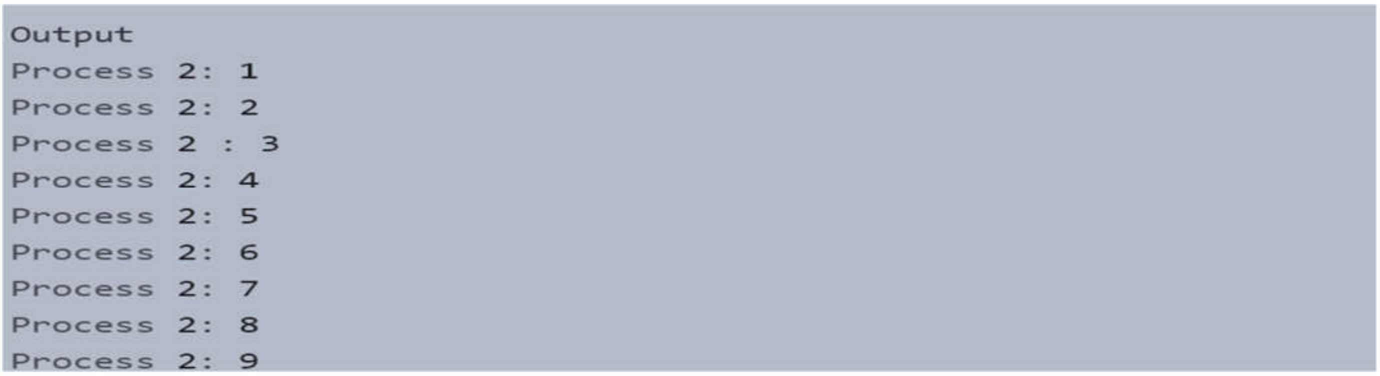
Output:

Process 1 which is producing data:



```
Output
Process 1: 1
Process 1: 2
Process 1 : 3
Process 1: 4
Process 1: 5
Process 1: 6
Process 1: 7
Process 1: 8
Process 1: 9
```

Process 2, which is consuming data



```
Output
Process 2: 1
Process 2: 2
Process 2 : 3
Process 2: 4
Process 2: 5
Process 2: 6
Process 2: 7
Process 2: 8
Process 2: 9
```

OBSERVATION AND FINDING:

- IPC (Inter-Process Communication) is a communication mechanism used in distributed systems to allow processes running on the same or different machines to communicate with each other. IPC can take different forms, such as pipes, sockets, message queues, shared memory, and semaphores.
- One key observation is that IPC enables different processes to work together and share resources, leading to increased efficiency and flexibility in distributed systems. IPC can also facilitate communication between processes running on different platforms and programming languages.
- Another important observation is that the choice of IPC mechanism can impact the overall performance, reliability, and security of the system. Different IPC mechanisms have different trade-offs in terms of performance, scalability, ease of use, and robustness, and developers need to carefully choose the right mechanism based on their requirements.

CONCLUSION:

IPC is a critical component of distributed systems, enabling different processes to work together and share resources. It provides flexibility, efficiency, and scalability to distributed systems, allowing them to communicate and work together seamlessly. However, the choice of IPC mechanism can impact the overall performance, reliability, and security of the system. Therefore, developers need to choose the right IPC mechanism based on their requirements and take into consideration the trade-offs of different mechanisms to ensure optimal system performance and security.

SIGN AND REMARK

DATE

R1 **(4 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**R4** **(3 Mark)**	**Total** **(15 Marks)**	**Signature**

EXPERIMENT NO: 2

Date of Performance:

Date of Submission:

AIM: Client/server using RPC/RMI.

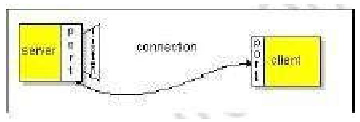
Theory:

The client-server model is one of the most used communication paradigms in networked systems. The server accepts the connection from the client, binds a new socket to the same local port, and sets its remote endpoint to the client's address and port. It needs a new socket so that it can continue to listen to the original socket for connection requests when the attention needs for the connected client.

Creating a server program:

The Echo Server example creates a server socket and waits for a client request. When it receives

aclient request, the server connects to the client and responds to it.



Following are the steps to create echo server application

- **Create and open a server socket.**
  Server Socket server Socket = new Server Socket (port Number);

The port Number argument is the logical address through which the application communicatesover the network. It's the port on which the server is running. You must provide the port number through which the server can listen to the client. Don't select port numbers between 0and 1,023 because they're reserved for privileged users (that is, super user or root). Add the server socket inside the try with- resources block.

- **Wait for the client request.**
  Socket client Socket = serverSocket.accept();

The accept () method waits until a client starts and requests a connection on the host and port ofthis server. When a connection is requested and successfully established, the accept () method returns a new Socket object. It's bound to the same local port, and its remote address and remote port are set to match the client's. The server can communicate with the client over this new object and listen for client connection requests

- **Open an input stream and an output stream to the client.**
  out = new Print Writer (clientSocket.getOutputStream(), true);
  in = new BufferedReader (new InputStreamReader (clientSocket.getInputStream()));

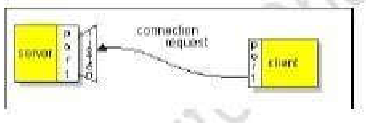
Communicate with the client.

Receive data from the client: (input Line =in.readLine()) Send data to the client: out.println(inputLine);

- **Close the streams and then close the socket.**

Creating a client program:

The client knows the host name of the machine on which the server is running. It also knows the port number on which the server is listening. To make a connection request, the client tries to connect with the server on the server's machine and port. Because the client also needs to identify itself to the server, it binds to a local port number that it will use during this connection. The system typically assigns the port number.



Following are the various steps to create client.

□**Create and open a client socket.**

Socket echoSocket = new Socket(hostName, portNumber);

The hostName argument is the machine where you are trying to open a connection, and portNumber is the port on which the server is running. Don't select port numbers between 0 and 1,023 because they're reserved for privileged users (that is, super user or root).

□**Open an input stream and an output stream to the socket.**

PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true); BufferedReader in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));

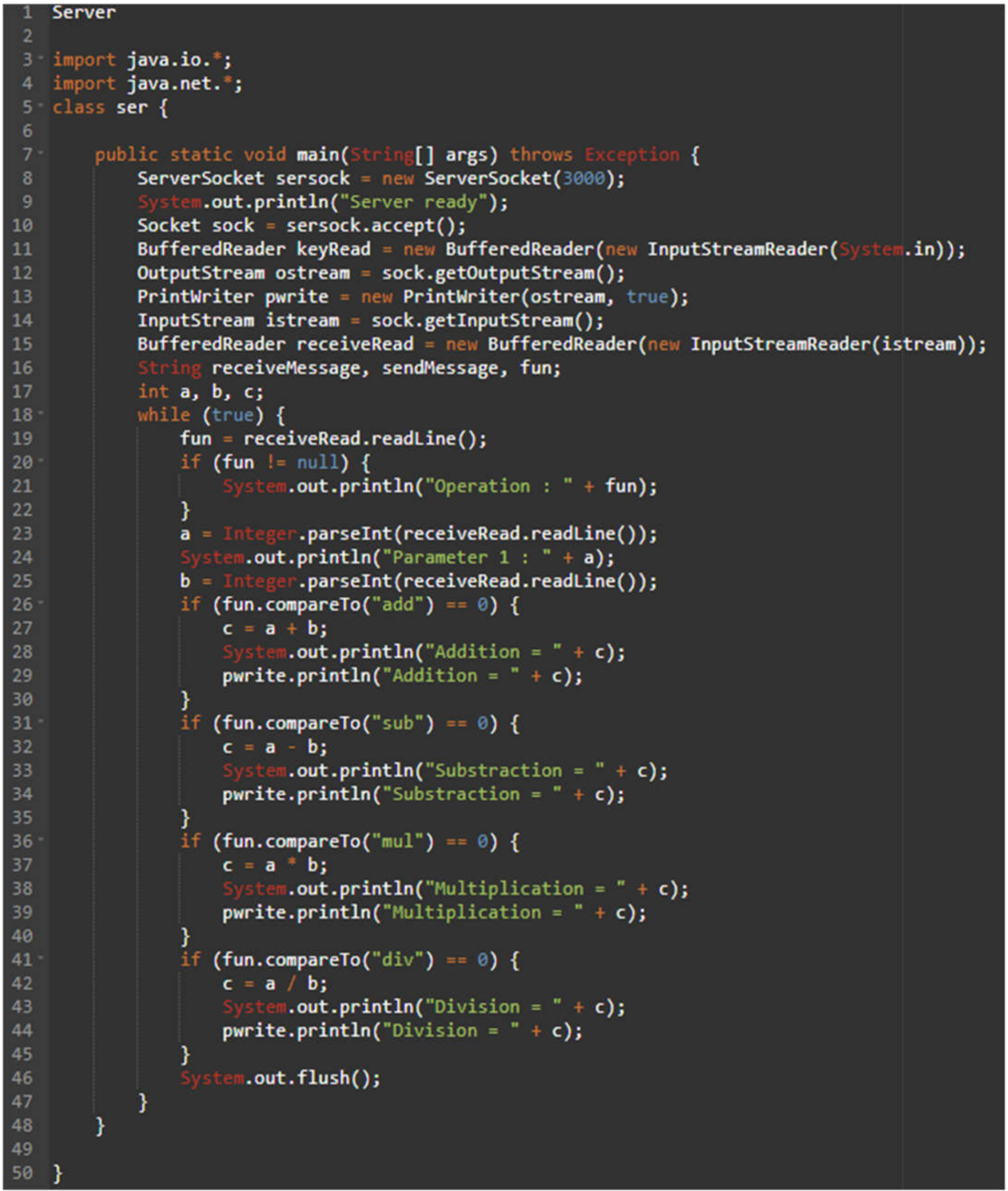
□**Read from and write to the stream according to the server's protocol.**

Receive data from the server: (userInput = stdIn.readLine())

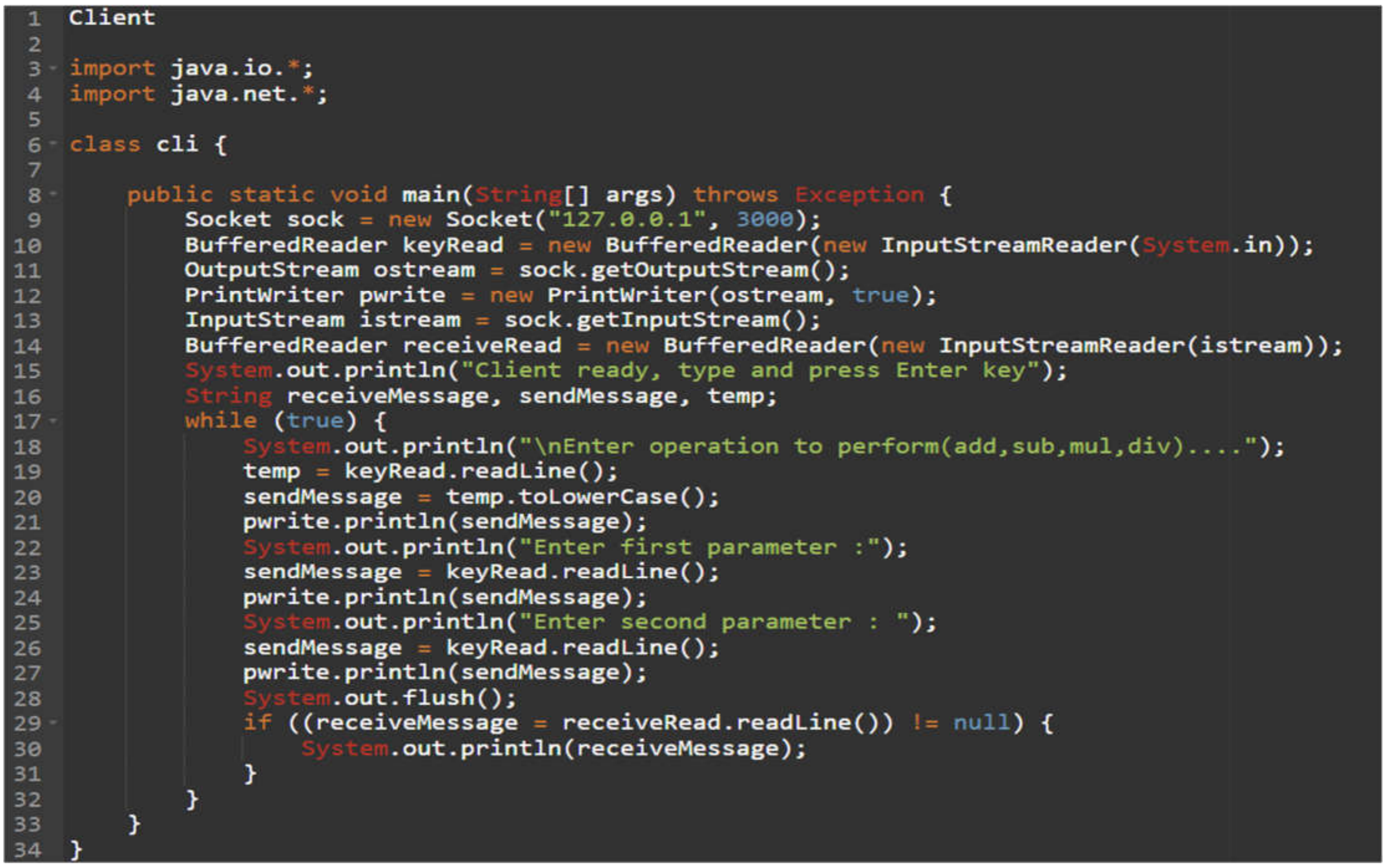
Send data to the server: out.println(userInput);

□**Close the streams and then close the socket.**

CODE:

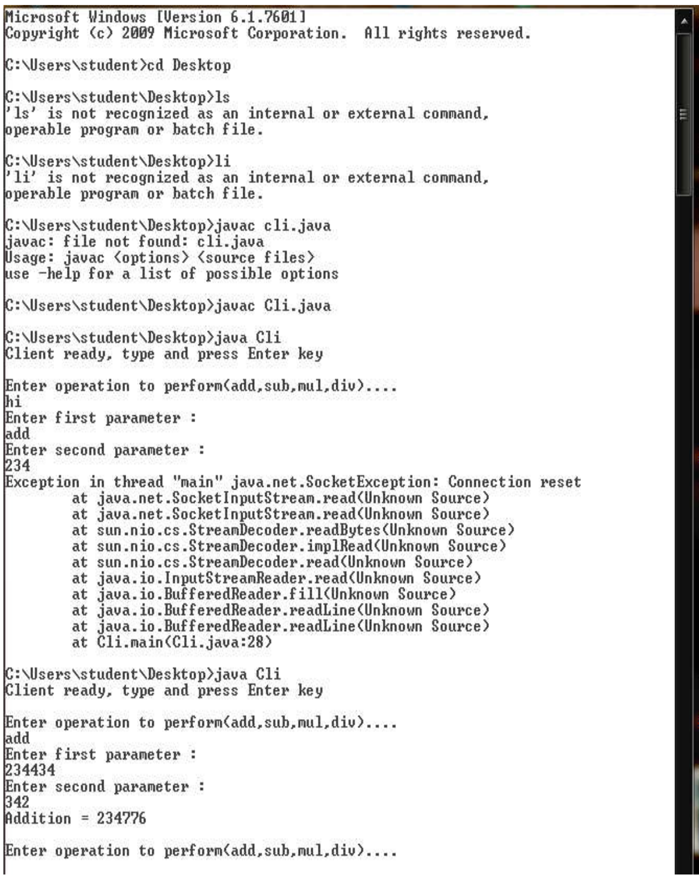


```
Server

import java.io.*;
import java.net.*;
class ser {

    public static void main(String[] args) throws Exception {
        ServerSocket sersock = new ServerSocket(3000);
        System.out.println("Server ready");
        Socket sock = sersock.accept();
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);
        InputStream istream = sock.getInputStream();
        BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
        String receiveMessage, sendMessage, fun;
        int a, b, c;
        while (true) {
            fun = receiveRead.readLine();
            if (fun != null) {
                System.out.println("Operation : " + fun);
            }
            a = Integer.parseInt(receiveRead.readLine());
            System.out.println("Parameter 1 : " + a);
            b = Integer.parseInt(receiveRead.readLine());
            if (fun.compareTo("add") == 0) {
                c = a + b;
                System.out.println("Addition = " + c);
                pwrite.println("Addition = " + c);
            }
            if (fun.compareTo("sub") == 0) {
                c = a - b;
                System.out.println("Substraction = " + c);
                pwrite.println("Substraction = " + c);
            }
            if (fun.compareTo("mul") == 0) {
                c = a * b;
                System.out.println("Multiplication = " + c);
                pwrite.println("Multiplication = " + c);
            }
            if (fun.compareTo("div") == 0) {
                c = a / b;
                System.out.println("Division = " + c);
                pwrite.println("Division = " + c);
            }
            System.out.flush();
        }
    }

}
```

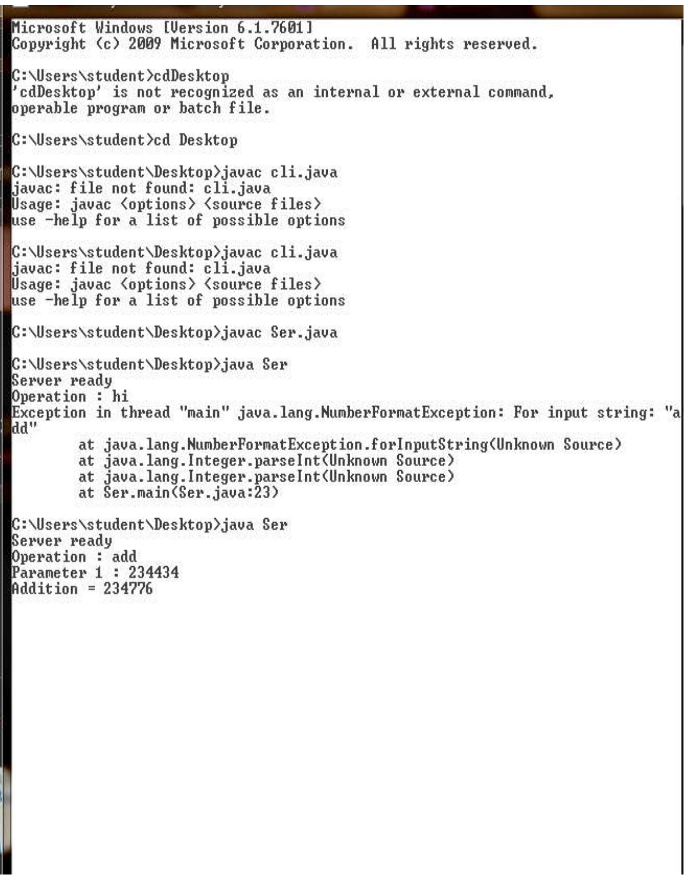


```java
Client

import java.io.*;
import java.net.*;

class cli {

    public static void main(String[] args) throws Exception {
        Socket sock = new Socket("127.0.0.1", 3000);
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);
        InputStream istream = sock.getInputStream();
        BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
        System.out.println("Client ready, type and press Enter key");
        String receiveMessage, sendMessage, temp;
        while (true) {
            System.out.println("\nEnter operation to perform(add,sub,mul,div)....");
            temp = keyRead.readLine();
            sendMessage = temp.toLowerCase();
            pwrite.println(sendMessage);
            System.out.println("Enter first parameter :");
            sendMessage = keyRead.readLine();
            pwrite.println(sendMessage);
            System.out.println("Enter second parameter : ");
            sendMessage = keyRead.readLine();
            pwrite.println(sendMessage);
            System.out.flush();
            if ((receiveMessage = receiveRead.readLine()) != null) {
                System.out.println(receiveMessage);
            }
        }
    }
}
```

Output:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\student>cd Desktop

C:\Users\student\Desktop>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\student\Desktop>li
'li' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\student\Desktop>javac cli.java
javac: file not found: cli.java
Usage: javac <options> <source files>
use -help for a list of possible options

C:\Users\student\Desktop>javac Cli.java

C:\Users\student\Desktop>java Cli
Client ready, type and press Enter key

Enter operation to perform(add,sub,mul,div)....
hi
Enter first parameter :
add
Enter second parameter :
234
Exception in thread "main" java.net.SocketException: Connection reset
        at java.net.SocketInputStream.read(Unknown Source)
        at java.net.SocketInputStream.read(Unknown Source)
        at sun.nio.cs.StreamDecoder.readBytes(Unknown Source)
        at sun.nio.cs.StreamDecoder.implRead(Unknown Source)
        at sun.nio.cs.StreamDecoder.read(Unknown Source)
        at java.io.InputStreamReader.read(Unknown Source)
        at java.io.BufferedReader.fill(Unknown Source)
        at java.io.BufferedReader.readLine(Unknown Source)
        at java.io.BufferedReader.readLine(Unknown Source)
        at Cli.main(Cli.java:28)

C:\Users\student\Desktop>java Cli
Client ready, type and press Enter key

Enter operation to perform(add,sub,mul,div)....
add
Enter first parameter :
234434
Enter second parameter :
342
Addition = 234776

Enter operation to perform(add,sub,mul,div)....
```



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\student>cdDesktop
'cdDesktop' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\student>cd Desktop

C:\Users\student\Desktop>javac cli.java
javac: file not found: cli.java
Usage: javac <options> <source files>
use -help for a list of possible options

C:\Users\student\Desktop>javac cli.java
javac: file not found: cli.java
Usage: javac <options> <source files>
use -help for a list of possible options

C:\Users\student\Desktop>javac Ser.java

C:\Users\student\Desktop>java Ser
Server ready
Operation : hi
Exception in thread "main" java.lang.NumberFormatException: For input string: "a
dd"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at Ser.main(Ser.java:23)

C:\Users\student\Desktop>java Ser
Server ready
Operation : add
Parameter 1 : 234434
Addition = 234776
```

OBSERVATION AND FINDING:

- RPC (Remote Procedure Call) is a communication mechanism used in distributed systems to allow one process to execute a function or method on another process on a remote machine. RPC involves the use of a client-server model, where the client sends a request to the server, and the server sends a response back to the client.
- One key observation is that RPC simplifies the development of distributed systems by providing a mechanism for remote procedure calls that abstracts away the details of the underlying network communication. This simplifies the programming model and can help reduce the complexity of the system.
- Another important observation is that RPC performance can be impacted by network latency, bandwidth, and reliability. These factors can impact the overall performance and responsiveness of the system, and developers need to take them into consideration when designing and implementing RPC-based applications.

CONCLUSION:

RPC is a powerful mechanism for building distributed systems. It simplifies the development process by providing a mechanism for remote procedure calls that abstracts away the details of the underlying network communication. However, RPC performance can be affected by network conditions, so developers need to be aware of this and take steps to optimize the system for better performance. Overall, RPC is a valuable tool for building distributed applications and can help simplify the development process while providing robust communication between distributed processes.

SIGN AND REMARK

DATE

R1 (4 Marks)	**R2** (4 Marks)	**R3** (4 Marks)	**R4** (3 Mark)	**Total** (15 Marks)	**Signature**

EXPERIMENT No. 2A

Date of Performance :

Date of Submission :

AIM : Implement a program to illustrate the concept of Remote Method Invocation (RMI) Problem : Design a distributed application using remote method invocation where client submit two strings to server and returns concatenation of given string.

OBJECTIVES:

- To study RMI concepts.
- To know how to invoke a method running in different JVM.
- To study how to write an application using RMI methodology.

THEORY/ALGORITHM:

The main objective of RMI is to provide the facility of invoking method on server. This is done by creating a RMI Client and RMI Server. RMI Client invokes a method defined on the RMI Server. In this article, we will learn:

RMI terminology:

Following figure shows the terminology of Remote Method Invocation.

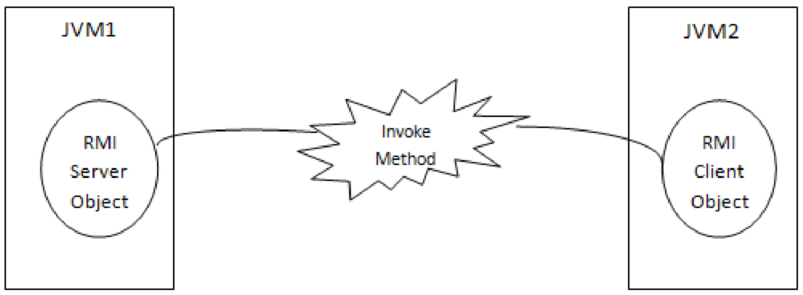


Figure : Method is invoked by client on the server using RMI

RMI is used to communicate between two running java applications on different JVM (Java Virtual Machines). The main motive behind RMI implementation is to invoke one method running on different JVM. The JVM Application, in which an invoked method is running out, is called RMI Server, where as the invoking application running on the different JVM is called RMI Client. **RMI Client**: It is an object that invokes remote methods on an RMI Server.
Stub: A stub is proxy that stands for RMI Server on client side and handles remote method invocation on behalf of RMI Client.
Skeleton: It is a proxy that stands for RMI client on server side and handles remote method invocation on RMI Server on behalf of client.

Registry Service: A Registry Service is an application that provides the facility of registration & lookup of Remote stub. A Registry Service provides location transparency of Remote Object to RMI Client.

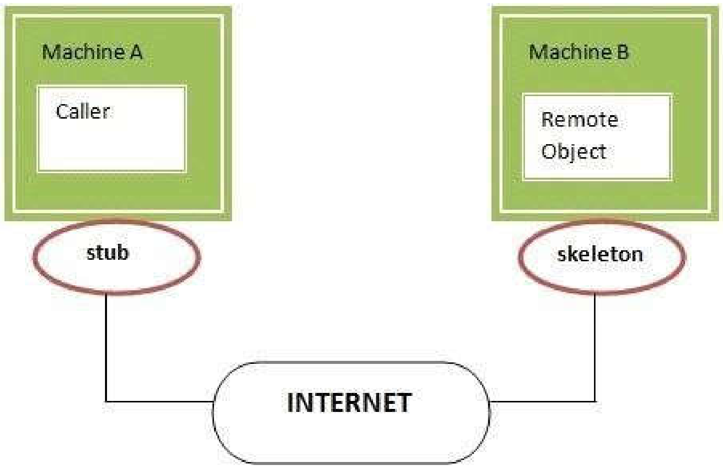


Figure : Role of stub and skeleton in RMI

Algorithm:
Step 1: Define Remote Interface
A remote interface specifies the methods that can be invoked remotely by a client. Clients program communicate to remote interfaces, not to classes implementing it. To be a remote interface, an interface must extend the **Remote** interface of **java.rmi** package.

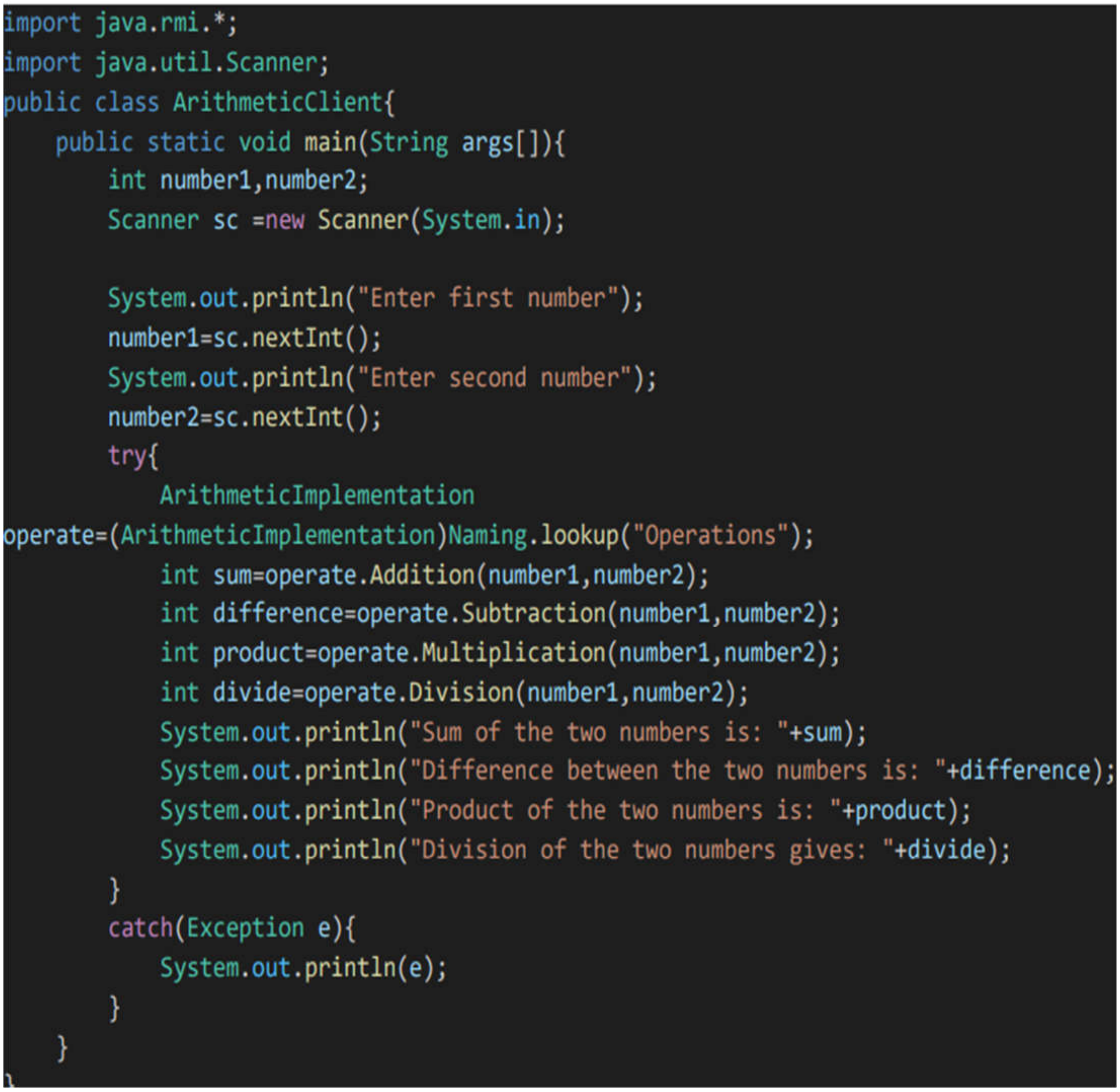
Step 2: Implementation of remote interface
For implementation of a remote interface, a class must either extend UnicastRemoteObject or use exportObject() method of the UnicastRemoteObject class.

Step 3: Create AddServer and host rmi service
You need to create a server application and host rmi service Adder in it. This is done using rebind() method of java.rmi.Naming class. rebind() method take two arguments, first represent the name of the object reference and second argument is reference to instance of Adder

Step 4: Create client application
Client application contains a java program that invokes the lookup() method of the Naming class. This method accepts one argument, the rmi URL and returns a reference to an object of type AddServerInterface. All remote method invocation is done on this object.

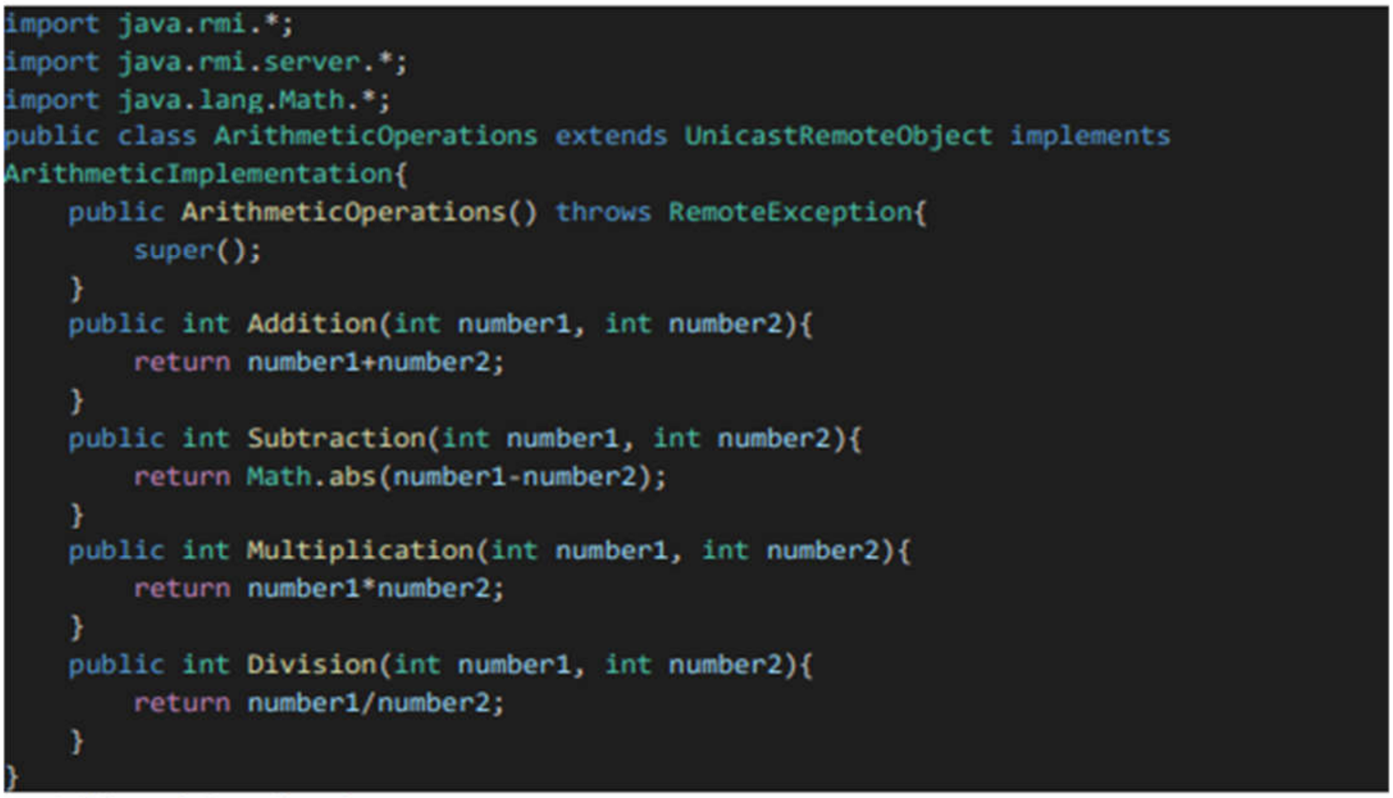
CODE:

ArithmeticClient.java:



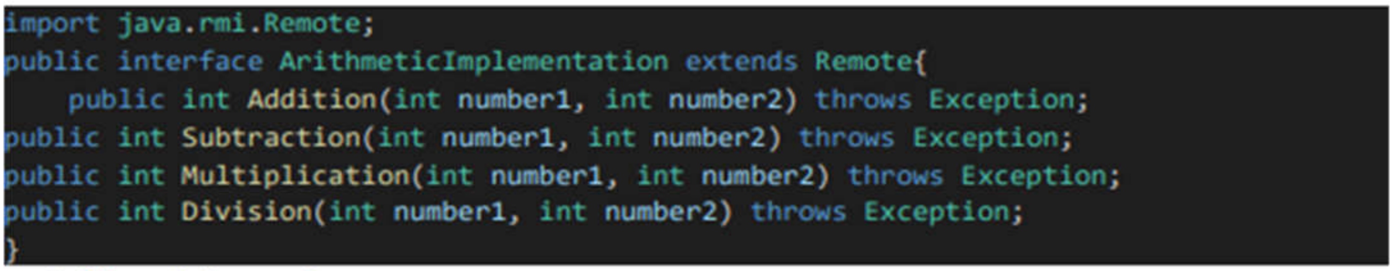
```
import java.rmi.*;
import java.util.Scanner;
public class ArithmeticClient{
    public static void main(String args[]){
        int number1,number2;
        Scanner sc =new Scanner(System.in);

        System.out.println("Enter first number");
        number1=sc.nextInt();
        System.out.println("Enter second number");
        number2=sc.nextInt();
        try{
            ArithmeticImplementation
operate=(ArithmeticImplementation)Naming.lookup("Operations");
            int sum=operate.Addition(number1,number2);
            int difference=operate.Subtraction(number1,number2);
            int product=operate.Multiplication(number1,number2);
            int divide=operate.Division(number1,number2);
            System.out.println("Sum of the two numbers is: "+sum);
            System.out.println("Difference between the two numbers is: "+difference);
            System.out.println("Product of the two numbers is: "+product);
            System.out.println("Division of the two numbers gives: "+divide);
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

ArithmeticOperations.java:



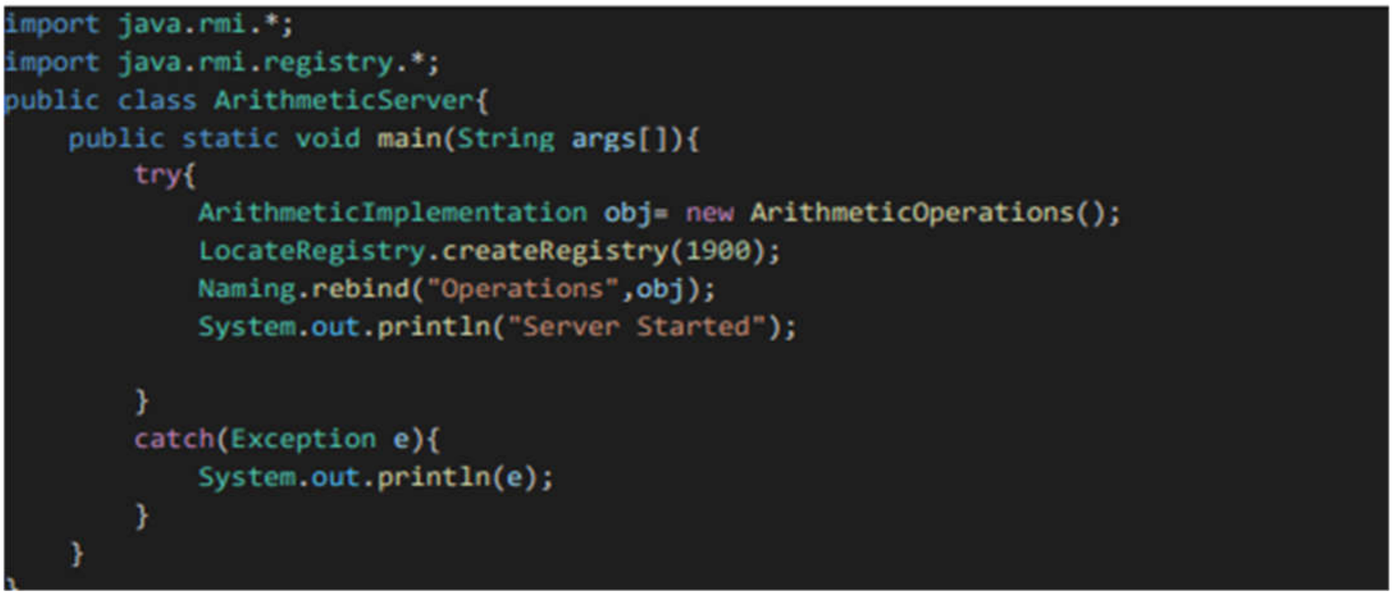
```java
import java.rmi.*;
import java.rmi.server.*;
import java.lang.Math.*;
public class ArithmeticOperations extends UnicastRemoteObject implements
ArithmeticImplementation{
    public ArithmeticOperations() throws RemoteException{
        super();
    }
    public int Addition(int number1, int number2){
        return number1+number2;
    }
    public int Subtraction(int number1, int number2){
        return Math.abs(number1-number2);
    }
    public int Multiplication(int number1, int number2){
        return number1*number2;
    }
    public int Division(int number1, int number2){
        return number1/number2;
    }
}
```

ArithmeticInterface.java:



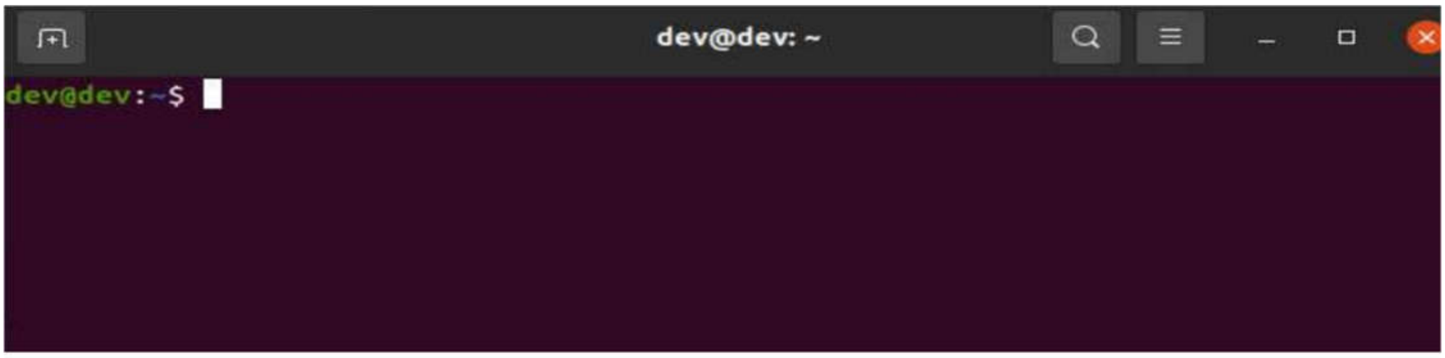
```java
import java.rmi.Remote;
public interface ArithmeticImplementation extends Remote{
    public int Addition(int number1, int number2) throws Exception;
public int Subtraction(int number1, int number2) throws Exception;
public int Multiplication(int number1, int number2) throws Exception;
public int Division(int number1, int number2) throws Exception;
}
```

ArithmeticServer.java:

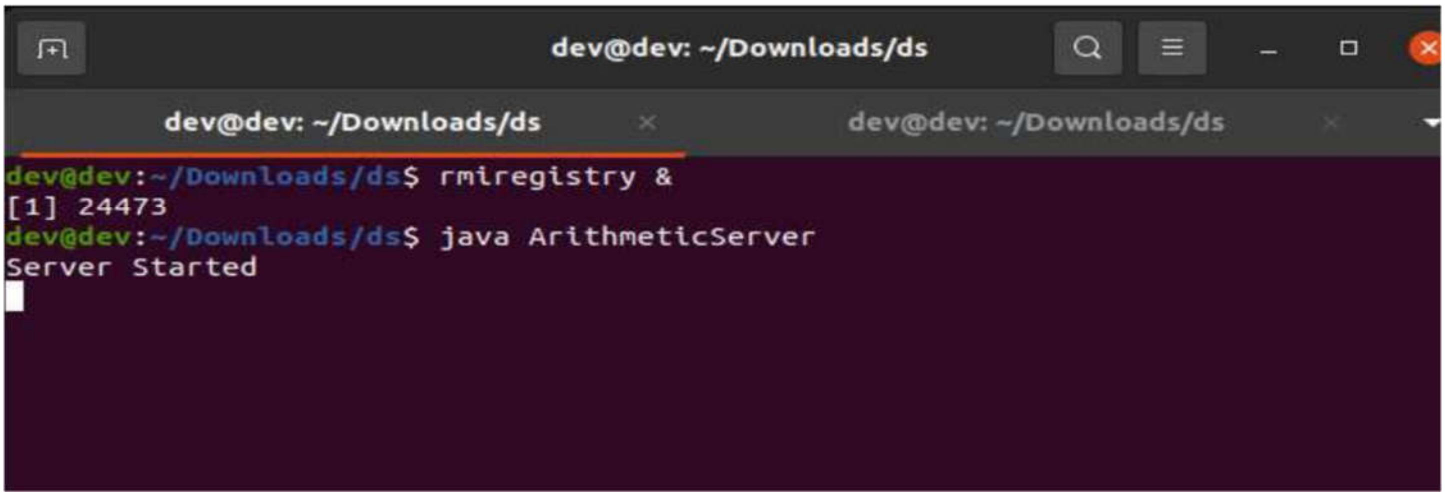


```java
import java.rmi.*;
import java.rmi.registry.*;
public class ArithmeticServer{
    public static void main(String args[]){
        try{
            ArithmeticImplementation obj= new ArithmeticOperations();
            LocateRegistry.createRegistry(1900);
            Naming.rebind("Operations",obj);
            System.out.println("Server Started");

        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

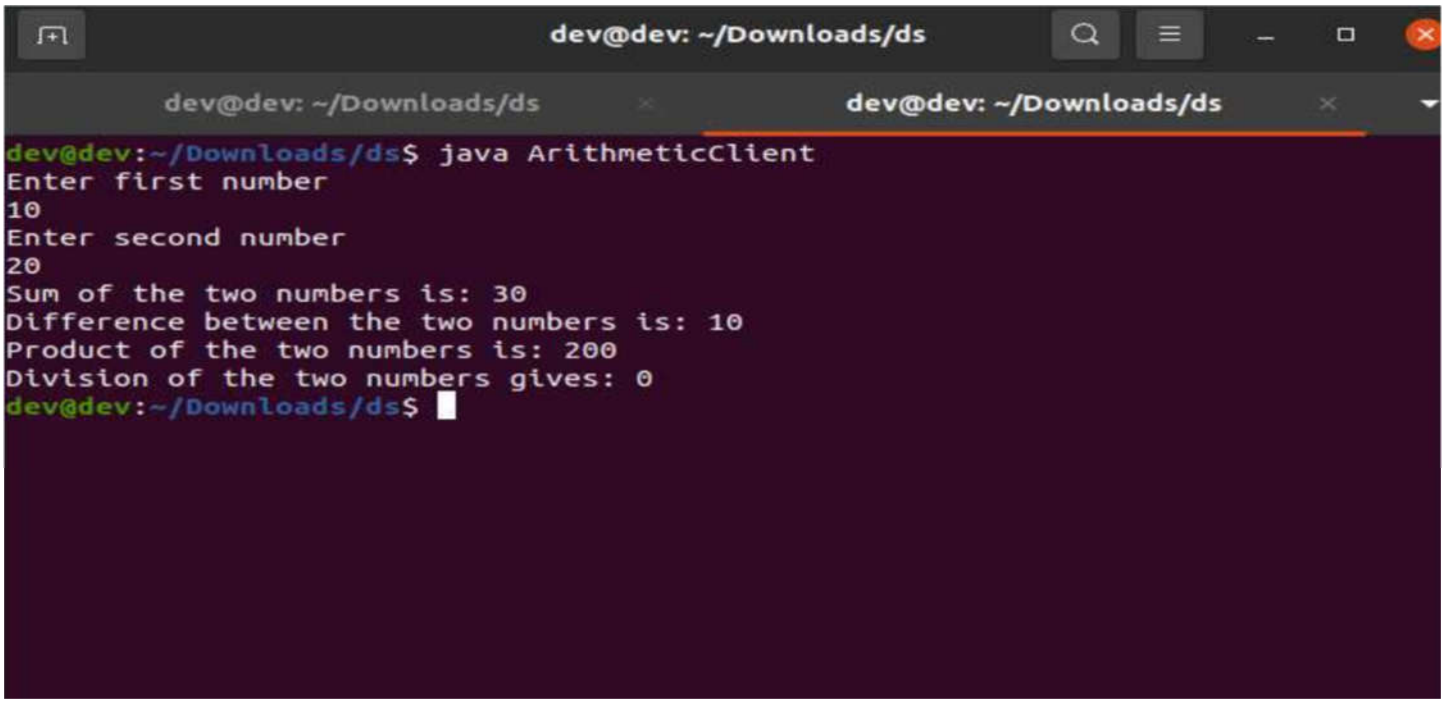
Output:



```
dev@dev:~$
```



```
dev@dev:~/Downloads/ds$ rmiregistry &
[1] 24473
dev@dev:~/Downloads/ds$ java ArithmeticServer
Server Started
```



```
dev@dev:~/Downloads/ds$ java ArithmeticClient
Enter first number
10
Enter second number
20
Sum of the two numbers is: 30
Difference between the two numbers is: 10
Product of the two numbers is: 200
Division of the two numbers gives: 0
dev@dev:~/Downloads/ds$
```

OBSERVATION AND FINDING:

- RMI (Remote Method Invocation) is a communication mechanism used in distributed systems to allow objects in one JVM (Java Virtual Machine) to invoke methods on objects in another JVM. RMI involves the use of stubs and skeletons to facilitate the communication between the two JVMs.
- One key observation is that RMI simplifies the development of distributed systems by providing a transparent communication mechanism that allows developers to interact with remote objects as if they were local objects. This simplifies the programming model and can help reduce the complexity of the system.
- Another important observation is that RMI performance can be impacted by network latency, bandwidth, and reliability. These factors can impact the overall performance and responsiveness of the system, and developers need to take them into consideration when designing and implementing RMI-based applications.

CONCLUSION:

RMI is a powerful mechanism for building distributed systems in Java. It simplifies the development process and allows developers to interact with remote objects as if they were local objects. However, RMI performance can be affected by network conditions, so developers need to be aware of this and take steps to optimize the system for better performance. Overall, RMI is a valuable tool for building distributed applications and can help simplify the development process while providing robust communication between distributed objects.

SIGN AND REMARK

DATE

R1 **(4 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**R4** **(3 Mark)**	**Total** **(15 Marks)**	**Signature**

EXPERIMENT No. 3

Date of Performance :

Date of Submission :

Aim: To implement group communication

Objectives/ Requirements:

1) Process groups may be dynamic. New groups can be created and old groups can be destroyed.
2) A process can join a group or leave one during system operation.
3) A process can be a member of several groups at the same time.
4) Consequently, mechanisms are needed for managing groups and group membership.
5) Ordering of the messeges can be considered.
6) Security of the message can also be added.

Theory: Communication in consists of three or more people/ processes who share a common goal and communicate collectively to achieve it
The key property that all groups have is that when a message is sent to the group itself, all members of the group receive it.
The purpose of introducing groups is to allow processes to deal with collections of processes as a single abstraction. Thus a process can send a message to a group of servers without having to know who they are or how many there are or where they are, which may change from one call to the next.

Types of Group:

Flat Groups versus Hierarchical Groups

1) Flat Groups : All the processes are equal. No one is boss and all decisions are made collectively. The flat group is symmetrical and has no single point of failure. If one of the processes crashes, the group simply becomes smaller, but can otherwise continue. A disadvantage is that decision making is more complicated. For example, to decide anything, a vote often has to be taken, incurring some delay and overhead.

2) Hierarchical Groups : Some kind of hierarchy exists. For example, one process is the coordinator and all the others are workers. In this model, when a request for work is generated, either by an external client or by one of the workers, it is sent to the coordinator. The coordinator then decides which worker is best suited to carry it out, and forwards it there. More complex hierarchies are also possible, of course

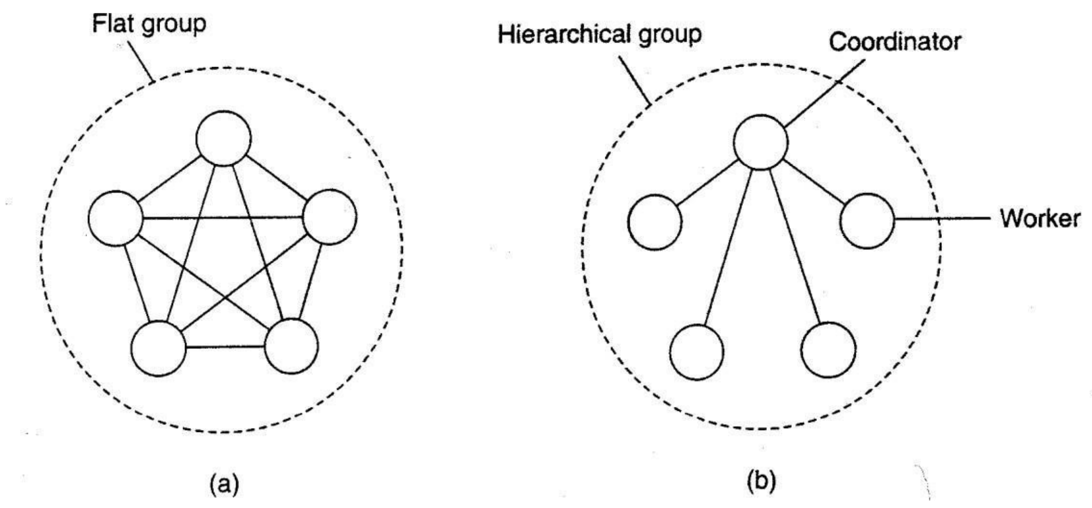


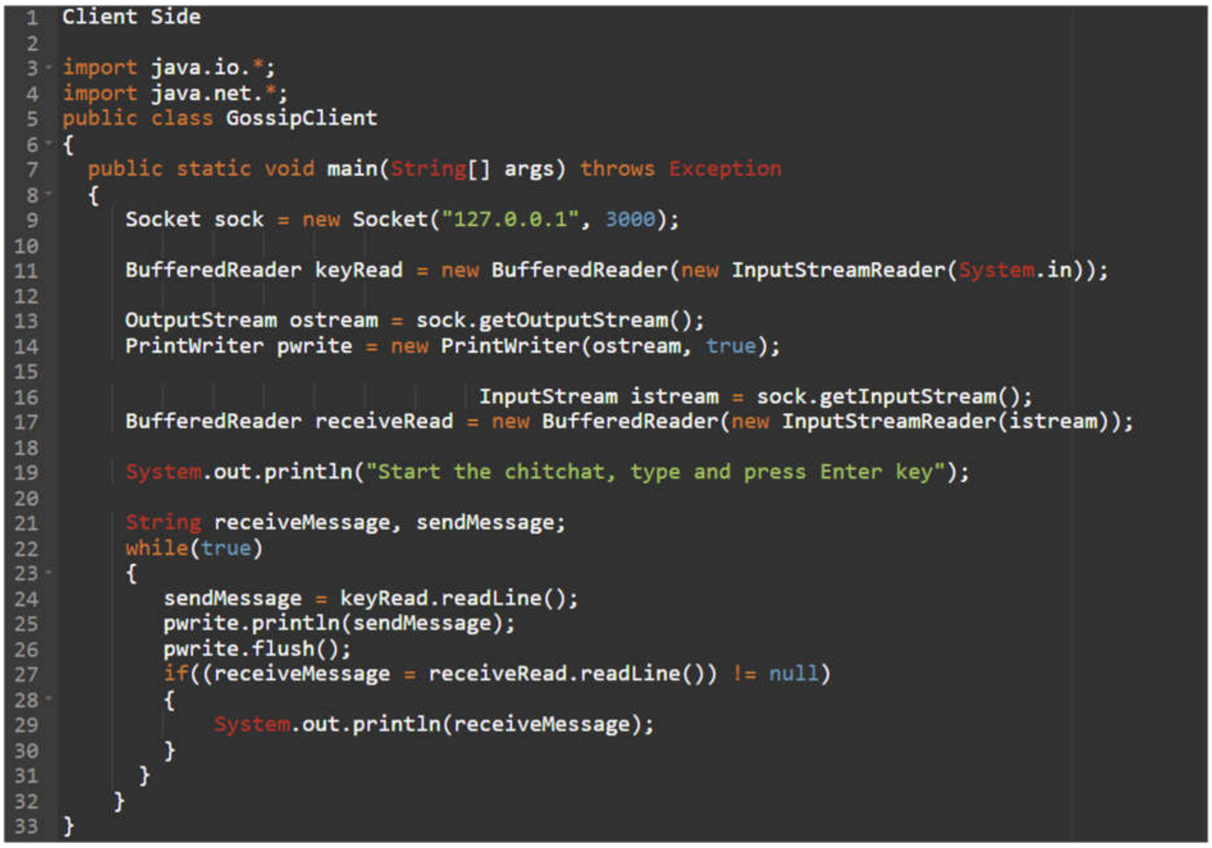
Figure (a) Communication in a flat group. (b) Communication in a simple hierarchical group.

Group Membership:

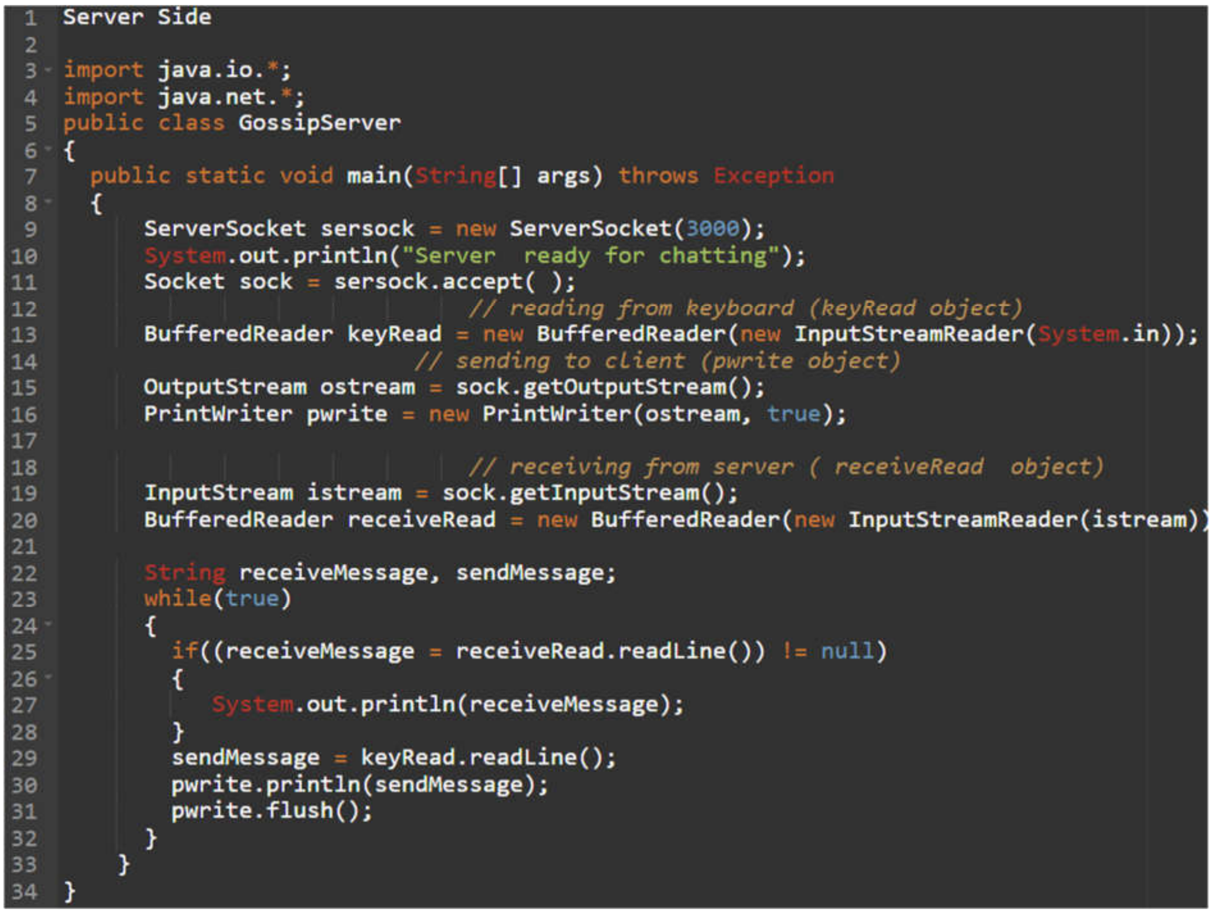
1) When group communication is present, some method is needed for creating and deleting groups, as well as for allowing processes to join and leave groups.
2) Member join: Either co-coordinator or by broadcasting
3)Member leaving : simply by sending message to group or co-coordinator removing.
4) Leaving and joining have to be synchronous with data messages being sent.

A process has joined a group, it must receive all messages sent to that group. Similarly, as soon as a process has left a group, it must not receive any more messages from the group, and the other members must not receive any more messages from it.
Co-coordinator crash: Make new co-ordinator.

CODE:



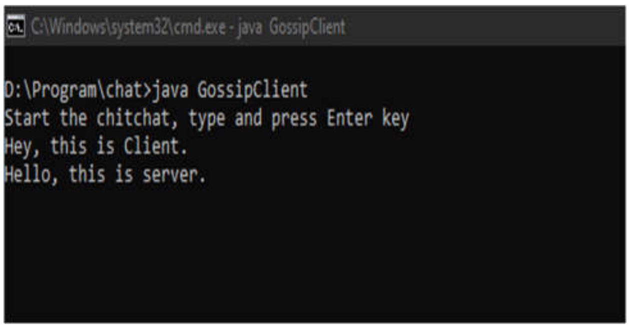
```
Client Side

import java.io.*;
import java.net.*;
public class GossipClient
{
  public static void main(String[] args) throws Exception
  {
     Socket sock = new Socket("127.0.0.1", 3000);

     BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));

     OutputStream ostream = sock.getOutputStream();
     PrintWriter pwrite = new PrintWriter(ostream, true);

                                 InputStream istream = sock.getInputStream();
     BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));

     System.out.println("Start the chitchat, type and press Enter key");

     String receiveMessage, sendMessage;
     while(true)
     {
        sendMessage = keyRead.readLine();
        pwrite.println(sendMessage);
        pwrite.flush();
        if((receiveMessage = receiveRead.readLine()) != null)
        {
            System.out.println(receiveMessage);
        }
      }
    }
}
```



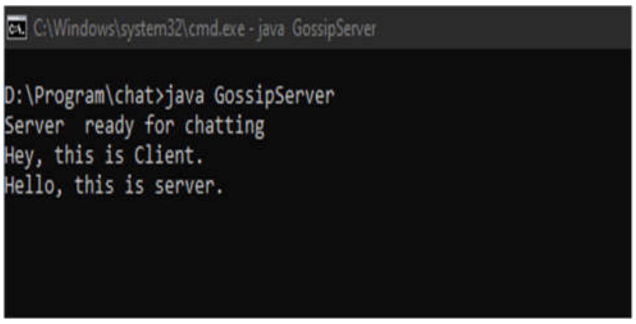
```
Server Side

import java.io.*;
import java.net.*;
public class GossipServer
{
  public static void main(String[] args) throws Exception
  {
      ServerSocket sersock = new ServerSocket(3000);
      System.out.println("Server  ready for chatting");
      Socket sock = sersock.accept( );
                              // reading from keyboard (keyRead object)
      BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
                          // sending to client (pwrite object)
      OutputStream ostream = sock.getOutputStream();
      PrintWriter pwrite = new PrintWriter(ostream, true);

                              // receiving from server ( receiveRead  object)
      InputStream istream = sock.getInputStream();
      BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream))

      String receiveMessage, sendMessage;
      while(true)
      {
        if((receiveMessage = receiveRead.readLine()) != null)
        {
           System.out.println(receiveMessage);
        }
        sendMessage = keyRead.readLine();
        pwrite.println(sendMessage);
        pwrite.flush();
      }
    }
}
```

Output:

Client **Server**



```
D:\Program\chat>java GossipClient
Start the chitchat, type and press Enter key
Hey, this is Client.
Hello, this is server.
```



```
D:\Program\chat>java GossipServer
Server  ready for chatting
Hey, this is Client.
Hello, this is server.
```

Observations and Findings:

- Group communication in distributed systems involves the exchange of information between multiple processes or nodes within a network. This communication can take place either through one-to-many or many-to-many communication protocols.
- One key observation is that in a distributed system, communication between nodes can be affected by various factors such as network latency, bandwidth, and reliability. These factors can impact the effectiveness of group communication and may result in delays, dropped messages, or even message loss.
- Another important observation is that group communication in distributed systems often involves the use of complex algorithms and protocols for ensuring reliability, consistency, and fault tolerance. These algorithms and protocols can have a significant impact on the performance and scalability of the system.

Conclusion:

Group communication is an essential aspect of distributed systems, and its effectiveness can impact the overall performance and reliability of the system. To ensure effective group communication, it is important to consider factors such as network latency, bandwidth, and reliability, and use appropriate algorithms and protocols for achieving reliability, consistency, and fault tolerance.

SIGN AND REMARK

DATE

R1 (4 Marks)	R2 (4 Marks)	R3 (4 Marks)	R4 (3 Mark)	Total (15 Marks)	Signature

EXPERIMENT No. 4

Date of Performance:

Date of Submission:

Aim: To implement an election algorithm.

Objectives/ Requirements:

1) Create a dynamic group
2) Create number of processes in the group
3) Calculate number of processes in the group.
4) Decide on the election algorithm (Optional)
5) Elect coordinator.

Theory:

Principle: Many distributed algorithms require that some process acts as a coordinator. The question is how to select this special process dynamically. Note: In many systems the coordinator is chosen by hand (e.g., file servers, DNS servers). This leads to centralized solutions => single point of failure.

– Doesn't matter which process does the job, just need to pick one

– Example: pick a master in Berkeley clock synchronization algorithm

• Election algorithms: technique to pick a unique coordinator

– Assumption: each process has a unique ID

– Goal: find the non-crashed process with the highest ID

Types of election algorithm:

1) Bully's algorithm

Assumptions

– Each process knows the ID and address of every other process

– Communication is reliable

– Need consistent result

• A process initiates an election if it just recovered from failure or it notices that the coordinatorhas failed

• Three types of messages: Election, OK, Coordinator.

• Several processes can initiate an election simultaneously.

Algorithm:

• Any process P can initiate an election

• P sends Election messages to all process with higher IDs and awaits OK messages

– If no OK messages, P becomes coordinator and sends Coordinator messages to allprocess's ith lower IDs

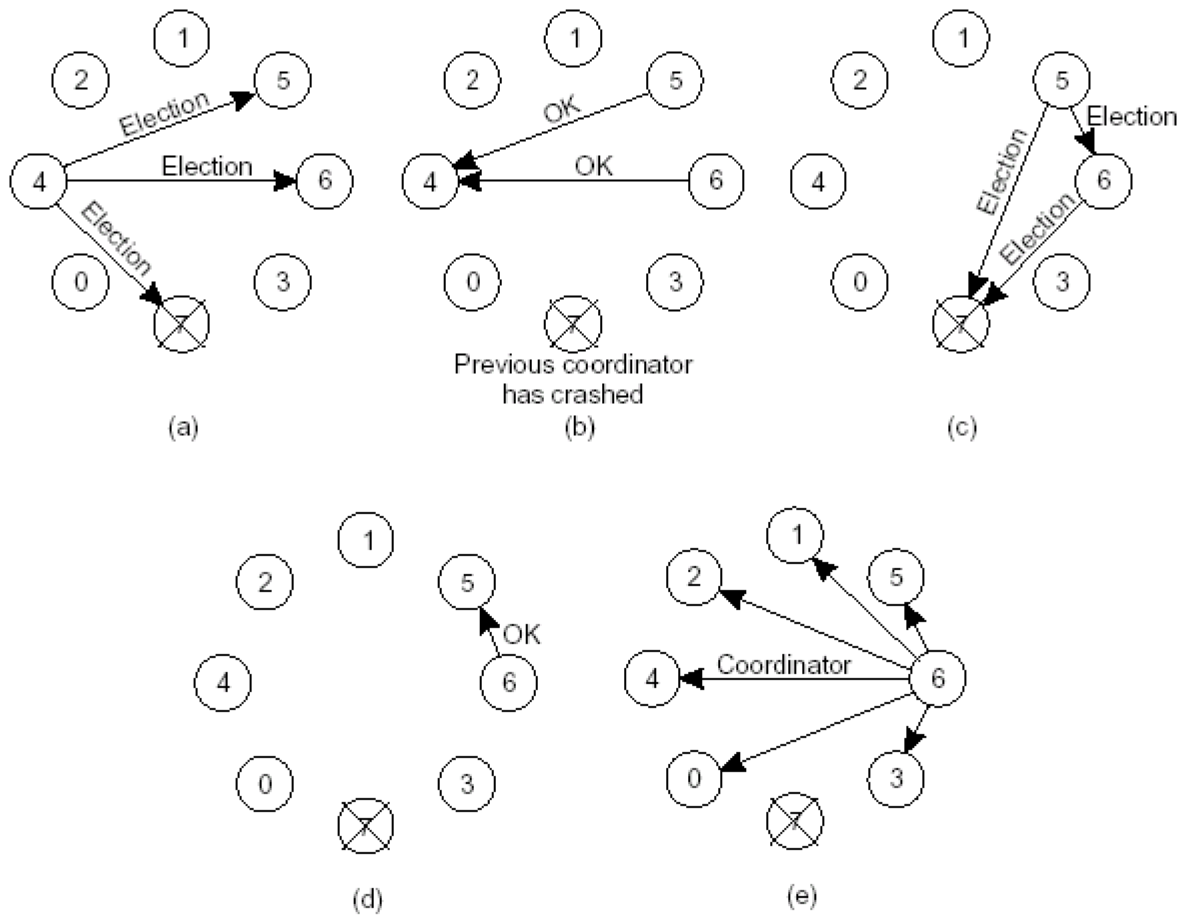
– If it receives an OK, it drops out and waits for a Coordinator message

• If a process receives an Election message

– Immediately sends Coordinator message if it is the process with highest ID

– Otherwise, returns an OK and starts an election

• If a process receives a Coordinator message, it treats sender as the coordinator.



2) **Ring Algorithm:**

Processes are arranged in a logical ring, each process knows the structure of the ring

• A process initiates an election if it just recovered from failure or it notices that the coordinatorhas failed

• Initiator sends Election message to closest downstream node that is alive

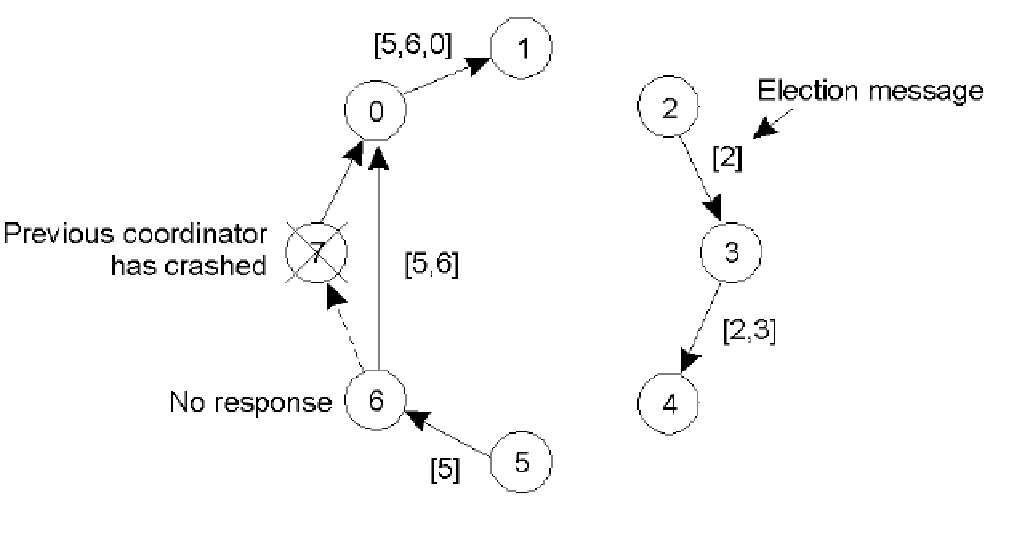
– Election message is forwarded around the ring

– Each process adds its own ID to the Election message

• When Election message comes back, initiator picks node with highest ID and sends aCoordinator message specifying the winner of the election

– Coordinator message is removed when it has circulated once.

• Multiple elections can be in progress



Assume n processes and one election in progress

• Bully algorithm

– Worst case: initiator is node with lowest ID

• Triggers n-2 elections at higher ranked nodes: O(n2) messages

– Best case: initiator is node with highest ID

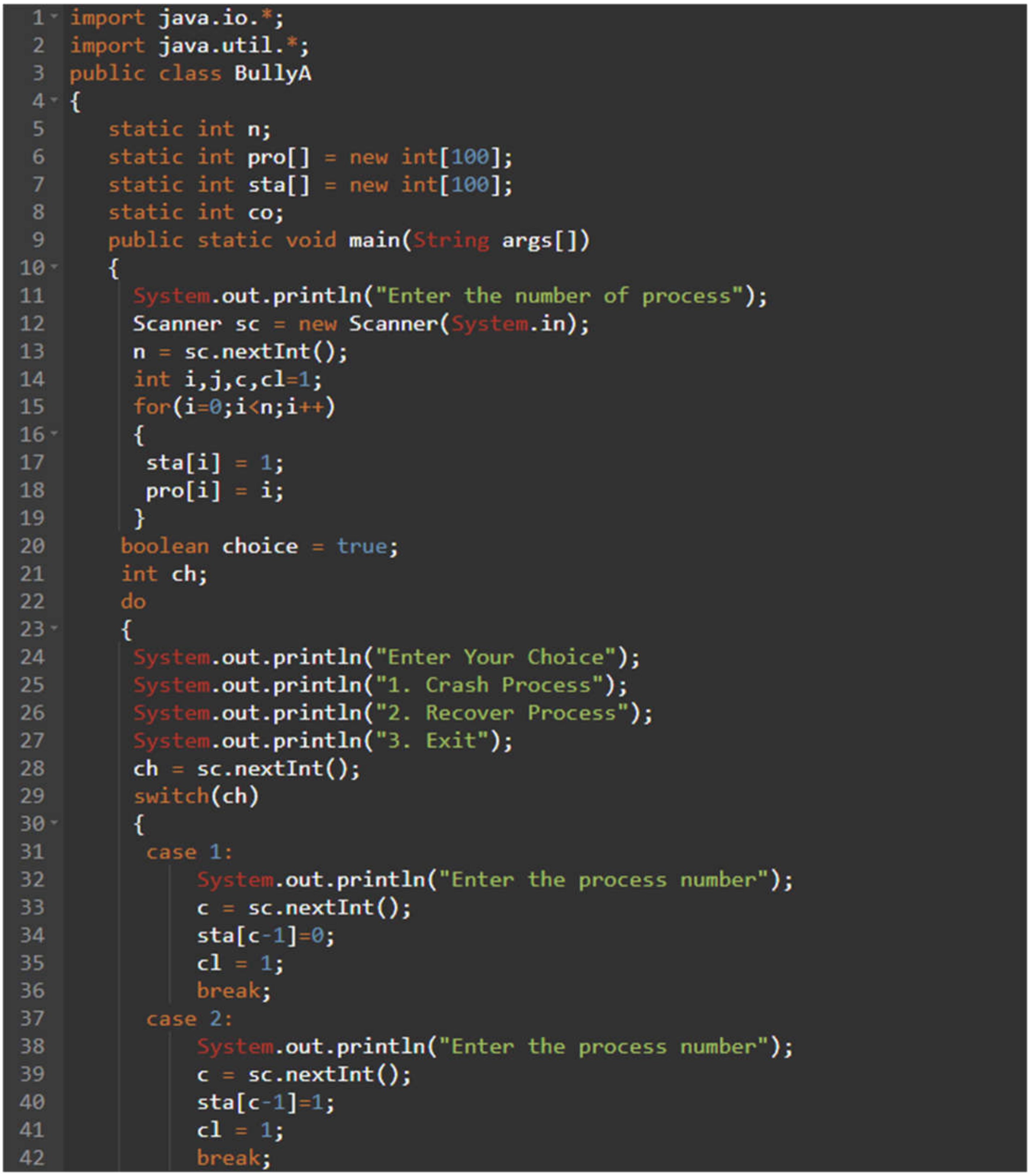
• Immediate election: n-1 messages

• Ring algorithm

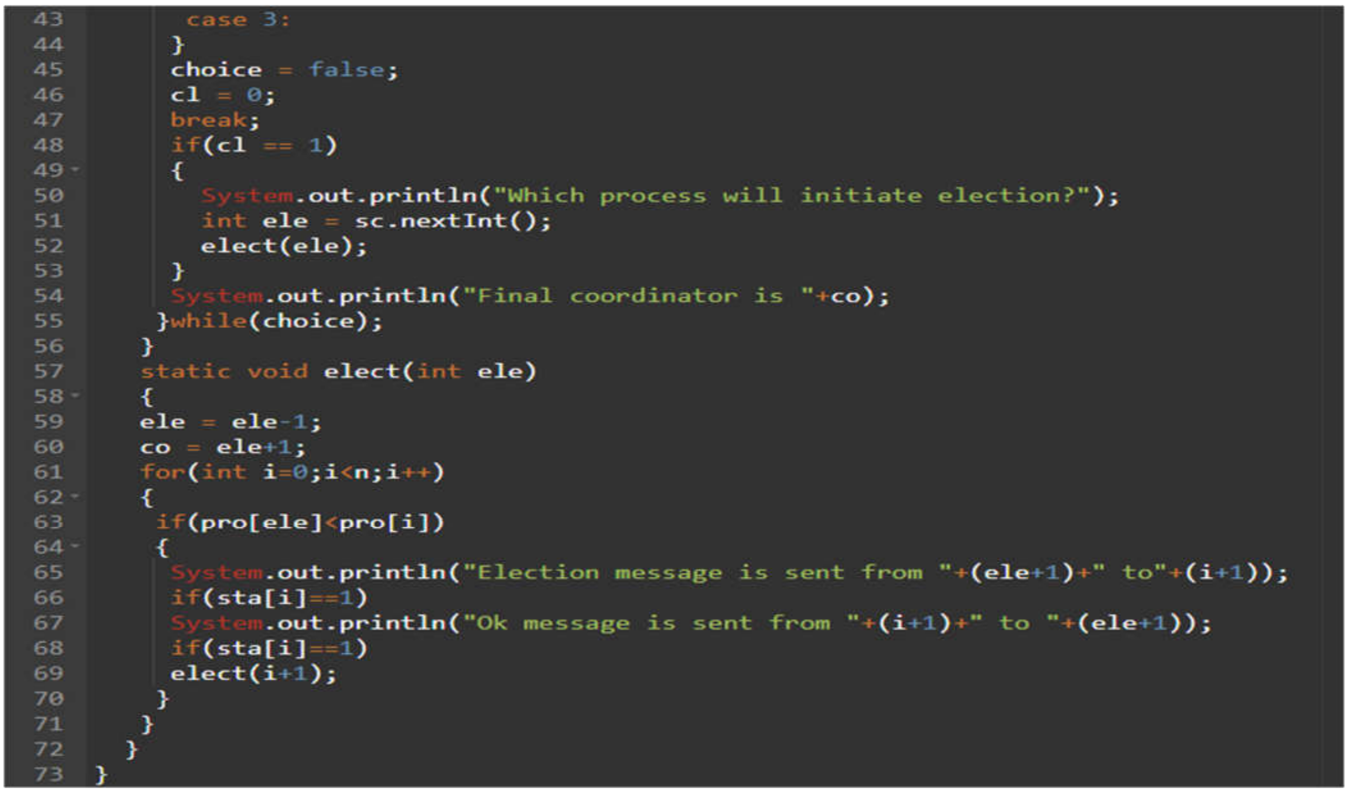
– 2n messages always

Bully's algorithm

CODE:

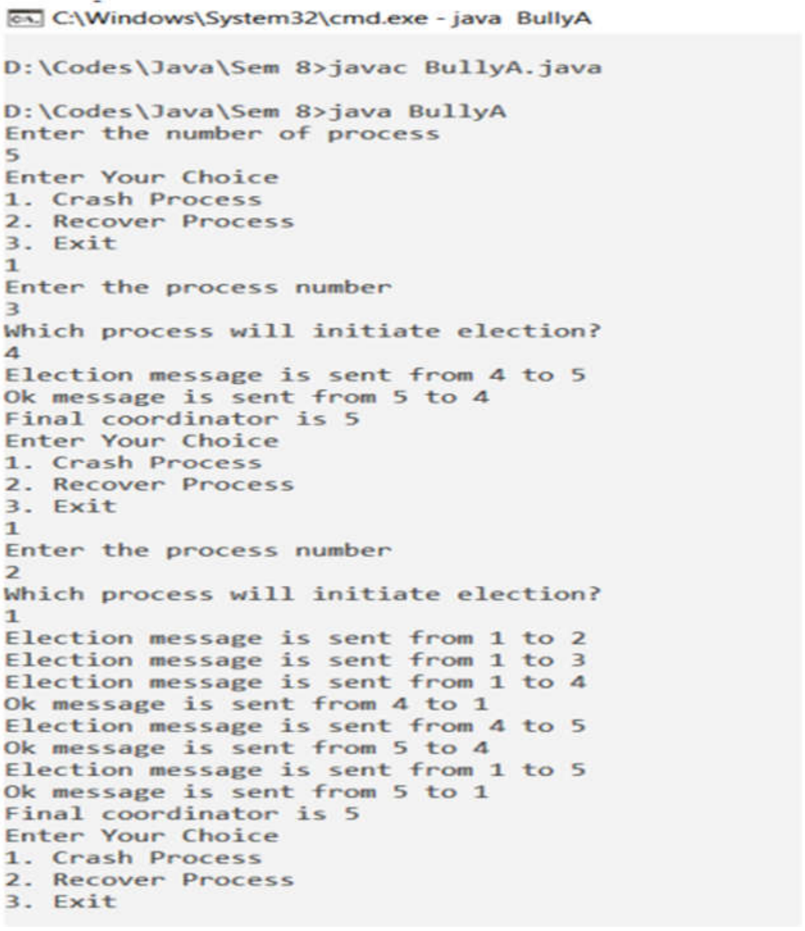


```java
import java.io.*;
import java.util.*;
public class BullyA
{
    static int n;
    static int pro[] = new int[100];
    static int sta[] = new int[100];
    static int co;
    public static void main(String args[])
    {
      System.out.println("Enter the number of process");
      Scanner sc = new Scanner(System.in);
      n = sc.nextInt();
      int i,j,c,cl=1;
      for(i=0;i<n;i++)
      {
       sta[i] = 1;
       pro[i] = i;
      }
     boolean choice = true;
     int ch;
     do
     {
      System.out.println("Enter Your Choice");
      System.out.println("1. Crash Process");
      System.out.println("2. Recover Process");
      System.out.println("3. Exit");
      ch = sc.nextInt();
      switch(ch)
      {
       case 1:
           System.out.println("Enter the process number");
           c = sc.nextInt();
           sta[c-1]=0;
           cl = 1;
           break;
       case 2:
           System.out.println("Enter the process number");
           c = sc.nextInt();
           sta[c-1]=1;
           cl = 1;
           break;
```



```
        case 3:
      }
      choice = false;
      cl = 0;
      break;
      if(cl == 1)
      {
        System.out.println("Which process will initiate election?");
        int ele = sc.nextInt();
        elect(ele);
      }
      System.out.println("Final coordinator is "+co);
    }while(choice);
  }
  static void elect(int ele)
  {
  ele = ele-1;
  co = ele+1;
  for(int i=0;i<n;i++)
  {
   if(pro[ele]<pro[i])
   {
    System.out.println("Election message is sent from "+(ele+1)+" to"+(i+1));
    if(sta[i]==1)
    System.out.println("Ok message is sent from "+(i+1)+" to "+(ele+1));
    if(sta[i]==1)
    elect(i+1);
   }
  }
 }
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - java  BullyA

D:\Codes\Java\Sem 8>javac BullyA.java

D:\Codes\Java\Sem 8>java BullyA
Enter the number of process
5
Enter Your Choice
1. Crash Process
2. Recover Process
3. Exit
1
Enter the process number
3
Which process will initiate election?
4
Election message is sent from 4 to 5
Ok message is sent from 5 to 4
Final coordinator is 5
Enter Your Choice
1. Crash Process
2. Recover Process
3. Exit
1
Enter the process number
2
Which process will initiate election?
1
Election message is sent from 1 to 2
Election message is sent from 1 to 3
Election message is sent from 1 to 4
Ok message is sent from 4 to 1
Election message is sent from 4 to 5
Ok message is sent from 5 to 4
Election message is sent from 1 to 5
Ok message is sent from 5 to 1
Final coordinator is 5
Enter Your Choice
1. Crash Process
2. Recover Process
3. Exit
```

Observations and Findings:

- Election algorithms are used in distributed systems to elect a leader or coordinator among a group of processes.
- Election algorithms can be categorized into centralized, decentralized, and hybrid algorithms.
- Centralized algorithms rely on a central coordinator to elect the leader, while decentralized algorithms distribute the election process across all processes in the system.
- Hybrid algorithms combine elements of both centralized and decentralized algorithms to achieve a balance between efficiency and fault-tolerance.
- The main goal of election algorithms is to ensure that a single leader is elected and can coordinate the activities of the system.

Conclusion:

- Election algorithms are an essential aspect of distributed systems, allowing processes to elect a leader or coordinator in a fair and reliable way.
- The choice of election algorithm depends on the requirements of the system, such as fault-tolerance, efficiency, and scalability.
- Centralized algorithms are efficient but vulnerable to a single point of failure, while decentralized algorithms are more fault-tolerant but can suffer from higher communication overhead.

SIGN AND REMARK

DATE

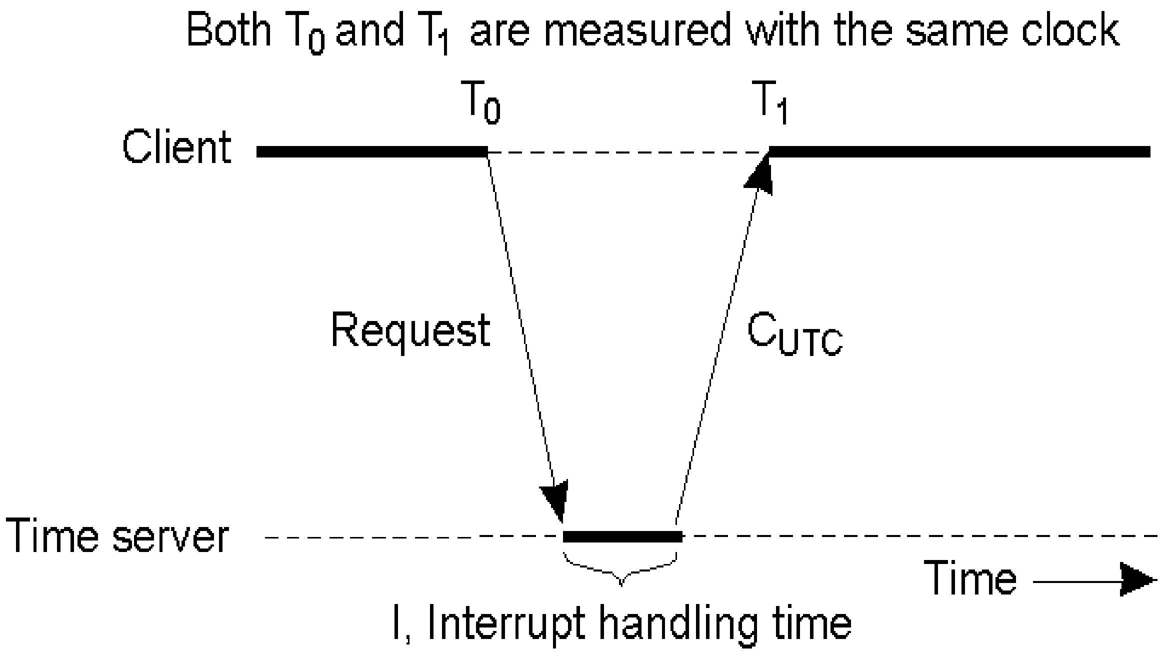
R1 (4 Marks)	R2 (4 Marks)	R3 (4 Marks)	R4 (3 Mark)	Total (15 Marks)	Signature

EXPERIMENT No. 5

Date of Performance :

Date of Submission :

Aim: To implement Christian's clock synchronization Algorithm
PROBLEM: Construct a program to demonstrate the concept of clock synchronization in distributed environment .
OBJECTIVES:
To study clock synchronization issues in distributed environment.
THEORY/ALGORITHM:
Distributed system is a collection of computer that are interconnected via somecommunication networks. Basically all the computers in distributed system are physically separatedand they may be located apart from each other. Therefore all the process that interact with each other need to be synchronized in the context of time to achieve some goal.
There are certain limitations of distributed systems that leave impact on the design of distributed systems
Following are some inherent limitations :
No global clock available
No shared Memory
In distributed system, there is no common or global clock available. Since in some situation, the programs on different computers need to coordinate their actions by exchanging message. Due to theunavailability of notion of a global clock there are limits on the accuracy of the output. Fo thatpurpose temporal ordering of events is required for scheduling processes and it becomes verydifficult for a distributed system.


Both T_0 and T_1 are measured with the same clock
T_0
T_1
Client
Request
C_{UTC}
Time server
Time
I, Interrupt handling time

Christian's algorithm is a method for clock synchronization which can be used in many fields of distributive computer science but is primarily used in low-latency intranets. Christian observed that this simple algorithm is probabilistic, in that it only achieves synchronization if the round-trip time (RTT) of the request is short compared to required accuracy.

Algorithm: -

Christian's algorithm works between a process P, and a time server S connected to a source of UTC(Coordinated Universal Time).

1. P requests the time from S
2. After receiving the request from P, S prepares a response and appends the time T from its own clock.
3. P then sets its time to be T +RTT/2

This method assumes that the RTT is split equally between request and response, which may not always be the case but is a reasonable assumption on a LAN connection.

Further accuracy can be gained by making multiple requests to S and using the response with the shortest RTT.

Let *min* be the minimum time to transmit a message one-way. The earliest point at which S could have placed the time T, was *min* after P sent its request. Therefore, the time at S, when the message by P is received, is in the range (T + *min*) to (T + RTT - *min*). The width of this range is (RTT - 2**min*). This gives an accuracy of (RTT/2 - *min*).

Server Side:

1. Open asocket.
2. Wait for request from client
3. Open an input stream and output stream to socket
4. Process the input with processing function and reply time.
5. Close program
6. Close socket

Client side

1. Open a socket.
2. Request server for system time
3. While requesting save client system time$T_{0.}$
4. When request served note client system time$T_{1.}$
5. ($T_{0.}$- T_1)/2 Compare Server time.
6. Add a difference to client system.
7. Display result
8. Close stream.
9. Close connection.

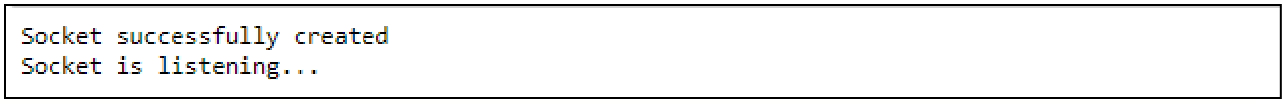
CODE:

Code below is used to initiate a prototype of a clock server on local machine:

```
# Python3 program imitating a clock server
import socket
import datetime

# function used to initiate the Clock Server
def initiateClockServer():
```

```
          s = socket.socket()
          print("Socket successfully created")
          port = 8000                  # Server port
          s.bind(('', port))
          s.listen(5)                  # Start listening to requests
          print("Socket is listening...")
          # Clock Server Running forever
          while True:
          connection, address = s.accept()          # Establish connection with client
           print('Server connected to', address)
          # Respond the client with server clock time
          connection.send(str(datetime.datetime.now()).encode())
          # Close the connection with the client process
          connection.close()
# Driver function
if __name__ == '__main__':
          initiateClockServer()      # Trigger the Clock Server
```

Output:



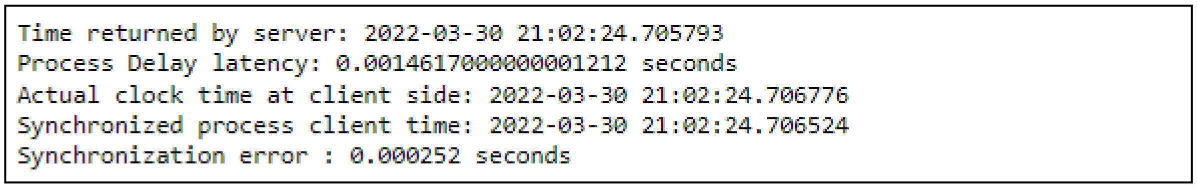
```
Socket successfully created
Socket is listening...
```

Code below is used to initiate a prototype of a client process on the local machine:

```
# Python3 program imitating a client process

import socket
import datetime
from dateutil import parser
from timeit import default_timer as timer
# function used to Synchronize client process time
def synchronizeTime():
   s = socket.socket()
          port = 8000           # Server port
          s.connect(('127.0.0.1', port))    # connect to the clock server on local computer
   request_time = timer()
   # receive data from the server
          server_time = parser.parse(s.recv(1024).decode())
          response_time = timer()
          actual_time = datetime.datetime.now()
   print("Time returned by server: " + str(server_time))
   process_delay_latency = response_time - request_time
   print("Process Delay latency: " \+ str(process_delay_latency) \ + " seconds")
   print("Actual clock time at client side: " \+ str(actual_time))
   # synchronize process client clock time
          client_time = server_time \+ datetime.timedelta(seconds = \(process_delay_latency) / 2)
    print("Synchronized process client time: " \+ str(client_time))
   # calculate synchronization error
          error = actual_time - client_time
```

```
        print("Synchronization error : "+ str(error.total_seconds()) + " seconds")
s.close()
# Driver function
if __name__ == '__main__':
    synchronizeTime()# synchronize time using clock server
```

OUTPUT:



```
Time returned by server: 2022-03-30 21:02:24.705793
Process Delay latency: 0.0014617000000001212 seconds
Actual clock time at client side: 2022-03-30 21:02:24.706776
Synchronized process client time: 2022-03-30 21:02:24.706524
Synchronization error : 0.000252 seconds
```

Observations and Findings:

Attempt to compensate for network delays

- Remember local time just before sending: T0
- Server gets request, and puts Ts into response
- When client receives reply, notes local time: T1
- Correct time is then approximately (Ts + (T1- T0) / 2) (assumes symmetric behaviour...)

Conclusion:

Uses a time server to synchronize clocks .Mainly designed for LAN . Time server keeps the reference time (say UTC) .A client asks the time server for time, the server responds with its current time, and the client uses the received value T to set its clock But network round-trip time introduces an error.

SIGN AND REMARK

DATE

R1 **(4 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**R4** **(3 Mark)**	**Total** **(15 Marks)**	**Signature**

EXPERIMENT No. 6

Date of Performance:

Date of Submission:

Aim: Build a program to implement Ricart– Agrawal mutual exclusion algorithm.

OBJECTIVES:

To Study the concept of mutual exclusion in distributed
environment. To study the working of Ricart-Agrawala algorithm

THEORY/ALGORITHM:

Mutual exclusion:

Concurrent access of processes to a shared resource or data is executed in 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.
Three basic approaches for distributed mutual exclusion:
1. Token based approach
2. Non-token-based approach
3. Quorum based approach

1. Token-based approach:

A unique token is shared among the sites.
A site is allowed to enter its CS if it possesses the
token. Mutual exclusion is ensured because the token is unique

2. 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.

3. 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

Ricart-Agrawala Algorithm

Optimization of Lamport's – no releases (merged with replies)

Requesting the critical section (CS):

When a processor wants to enter the CS, it:

Sends a timestamped request to all OTHER processors

When a processor receives a request:

If it is neither requesting nor executing the CS, it returns a reply(not timestamped) 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 Means the other processor requested first Otherwise, it defers answering the request.

Executing the CS:

A processor enters the CS when:

It has received a reply from all other processors in the system

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 receive its last reply message and enter the CS)

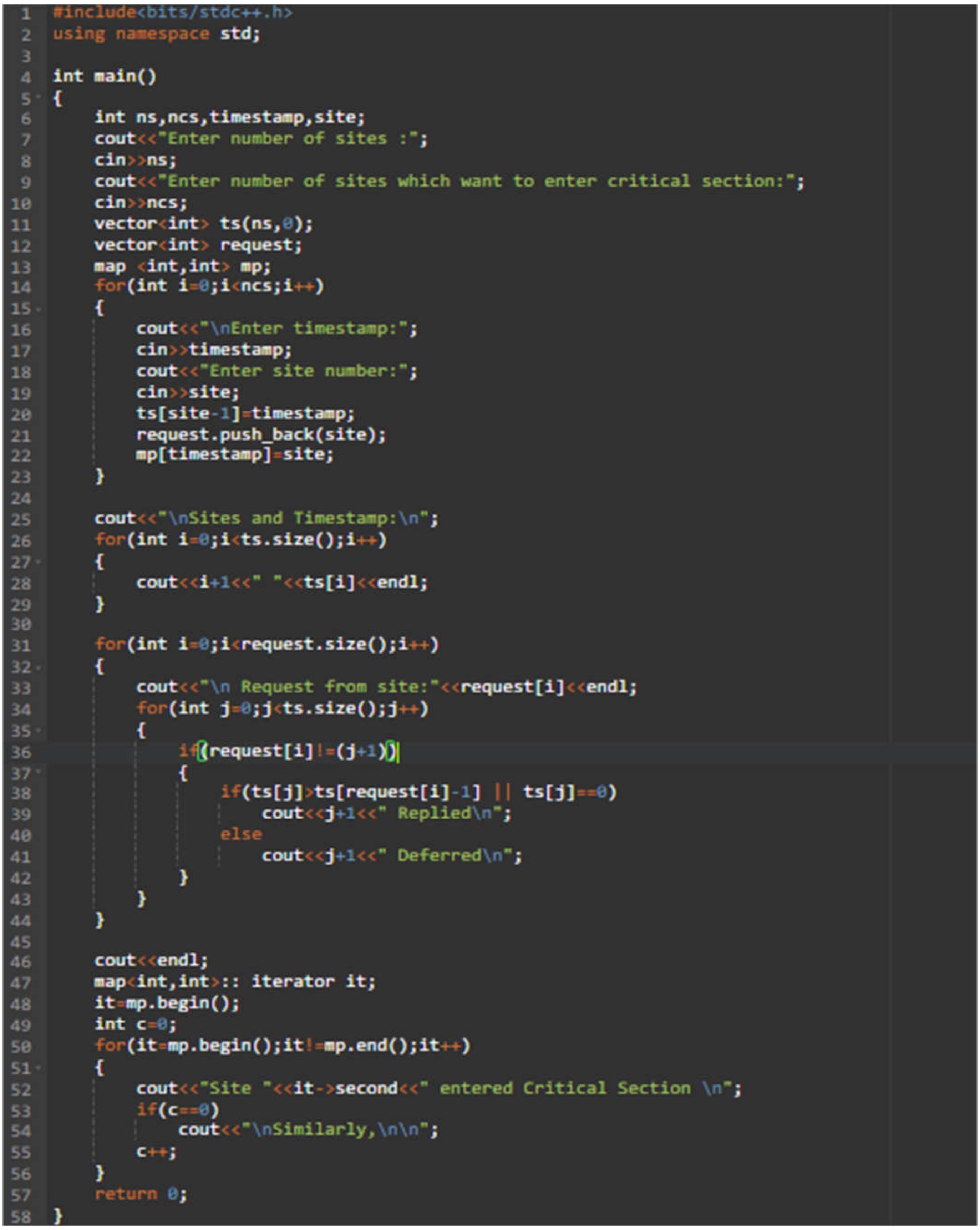
Evaluation:

message complexity - 2(N–1)

(N–1) reply, (N–1) request

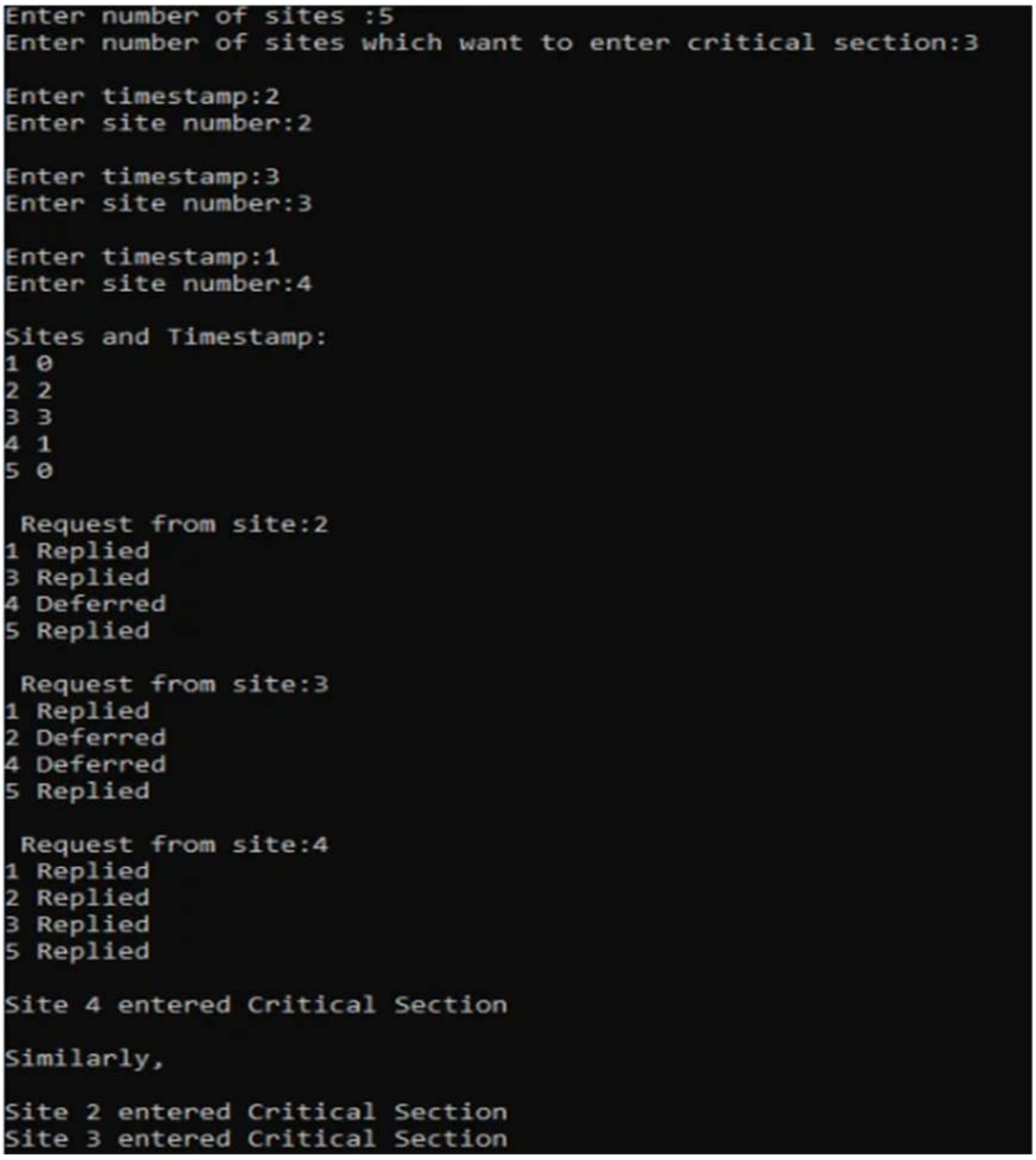
synchronization delay – 1 T

CODE:



```cpp
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int ns,ncs,timestamp,site;
    cout<<"Enter number of sites :";
    cin>>ns;
    cout<<"Enter number of sites which want to enter critical section:";
    cin>>ncs;
    vector<int> ts(ns,0);
    vector<int> request;
    map <int,int> mp;
    for(int i=0;i<ncs;i++)
    {
        cout<<"\nEnter timestamp:";
        cin>>timestamp;
        cout<<"Enter site number:";
        cin>>site;
        ts[site-1]=timestamp;
        request.push_back(site);
        mp[timestamp]=site;
    }

    cout<<"\nSites and Timestamp:\n";
    for(int i=0;i<ts.size();i++)
    {
        cout<<i+1<<" "<<ts[i]<<endl;
    }

    for(int i=0;i<request.size();i++)
    {
        cout<<"\n Request from site:"<<request[i]<<endl;
        for(int j=0;j<ts.size();j++)
        {
            if((request[i]!=(j+1)))
            {
                if(ts[j]>ts[request[i]-1] || ts[j]==0)
                    cout<<j+1<<" Replied\n";
                else
                    cout<<j+1<<" Deferred\n";
            }
        }
    }

    cout<<endl;
    map<int,int>:: iterator it;
    it=mp.begin();
    int c=0;
    for(it=mp.begin();it!=mp.end();it++)
    {
        cout<<"Site "<<it->second<<" entered Critical Section \n";
        if(c==0)
            cout<<"\nSimilarly,\n\n";
        c++;
    }
    return 0;
}
```

Output:



```
Enter number of sites :5
Enter number of sites which want to enter critical section:3

Enter timestamp:2
Enter site number:2

Enter timestamp:3
Enter site number:3

Enter timestamp:1
Enter site number:4

Sites and Timestamp:
1 0
2 2
3 3
4 1
5 0

 Request from site:2
1 Replied
3 Replied
4 Deferred
5 Replied

 Request from site:3
1 Replied
2 Deferred
4 Deferred
5 Replied

 Request from site:4
1 Replied
2 Replied
3 Replied
5 Replied

Site 4 entered Critical Section

Similarly,

Site 2 entered Critical Section
Site 3 entered Critical Section
```

Observations and Findings:

- The Ricart-Agrawala algorithm provides mutual exclusion in a distributed system by allowing processes to request access to a critical section.
- The algorithm ensures that no two processes can access the critical section simultaneously.
- Processes request access by sending a request message with a timestamp indicating when the request was made.
- Other processes defer their replies if their timestamp is greater than the requesting process's timestamp.
- Once a process has received replies from all other processes, it can enter the critical section.
- After a process has finished executing in the critical section, it sends a reply message to all other processes that deferred their replies.

Conclusion:

- The Ricart-Agrawala algorithm is an effective algorithm for mutual exclusion in distributedsystems.
- The algorithm ensures that no two processes can access a critical section simultaneously.
- The algorithm does not require a central server or coordinator, making it a decentralizedapproach.
- The algorithm can handle network failures and communication delays, making it suitable forreal-world distributed systems.
- However, the algorithm can suffer from a high overhead of message passing, especially when many processes are involved.
- The algorithm assumes that all processes have access to a global clock or a clock synchronization protocol, which may not always be the case in real-world systems.

SIGN AND REMARK

DATE

R1 **(4 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**R4** **(3 Mark)**	**Total** **(15 Marks)**	**Signature**

EXPERIMENT No. 7

AIM: Build a program to implement Edge Chasing deadlock detection algorithm.

OBJECTIVES:

- know the concept of deadlock.
- interpreted the concept of distributed deadlock. know the process of detecting
- deadlock.

THEORY/ALGORITHM:

A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function. The same conditions for deadlock in uniprocessors apply to distributed systems. Unfortunately, as in many other aspects of distributed systems, they are harder to detect, avoid, and prevent.

Chandy-Misra-Haas's distributed deadlock detection algorithm for AND model is based on edge-chasing. The algorithm uses a special message called probe, which is a triplet (i, j, k), denoting that it belongs to a deadlock detection initiated for process Pi and it is being sent by the home site of process Pj to the home site of process Pk. A probe message travels along the edges of the global TWF graph, and a deadlock is detected when a probe message returnsto the process that initiated it.

A process Pj is said to be dependent on another process Pk if there exists a sequence of processes Pj, Pi1, Pi2, …, Pim, Pk such that each process except Pk in the sequence is blocked and each process, except the Pj, holds a resource for which the previous process in the sequence is waiting. Process Pj is said to be locally dependent upon process Pk if Pj is dependent upon Pk and both the processes are on the same site. Eachprocess Pi maintains a Boolean array, dependent, where dependent i (j) is true only if Pi knows that Pj is dependent on it. Initially, dependent i(j) is false for all i and j.

Edge Chasing deadlock detection algorithm:

if *Pi* is locally dependent on
itself then declare a deadlock
else for all *Pj* and *Pk* such that
1 *Pi* is locally dependent upon *Pj*,
and 2 *Pj* is waiting on *Pk*, and
3 *Pj* and *Pk* are on different sites, send a probe (i, j, k) to the home site of *Pk*

On the receipt of a probe (i, j, k), the site takes the following
actions: if

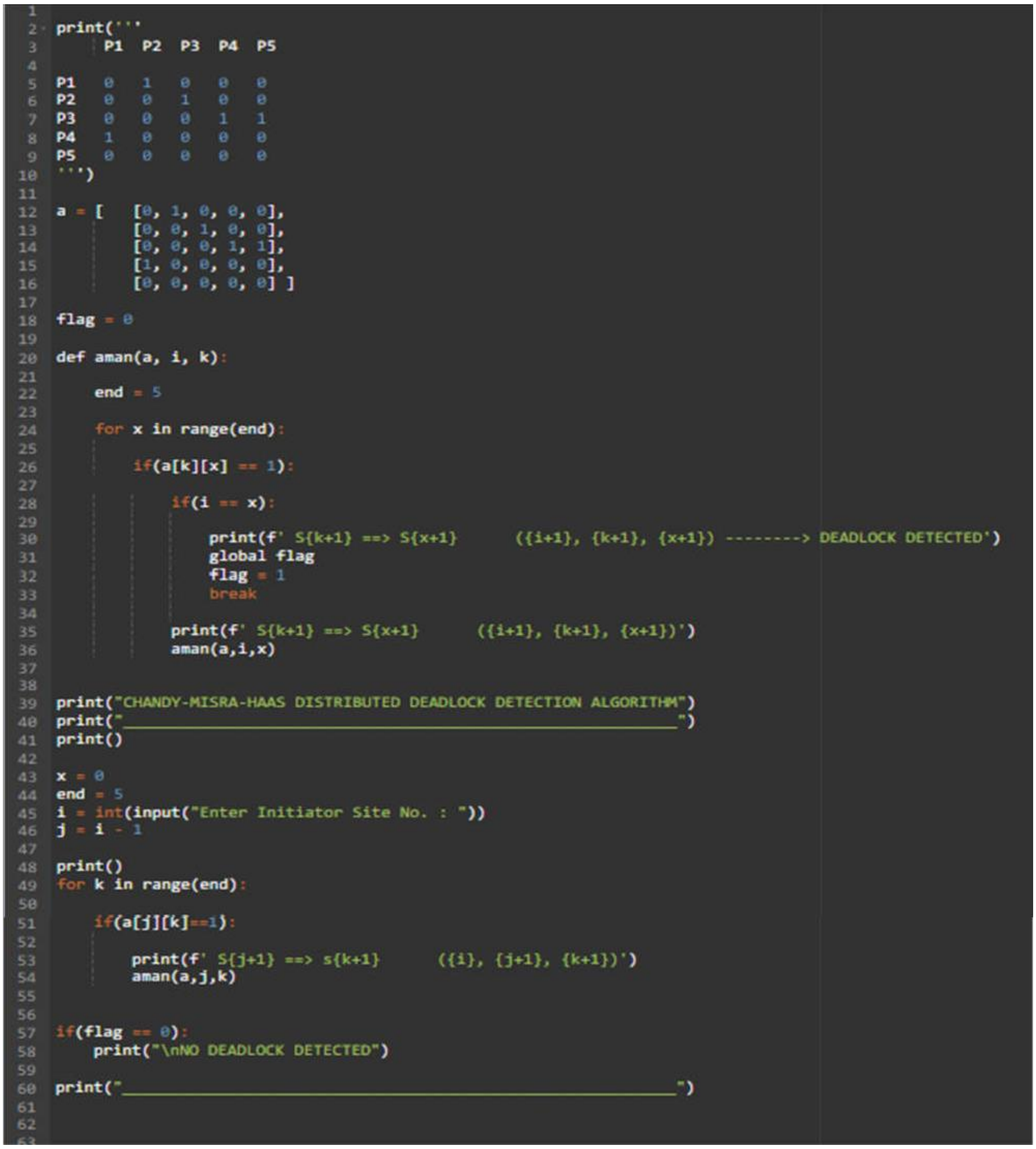
1 *Pk* is blocked, and
2 *dependent k*(i) is false, and
3 Pk has not replied to all requests *Pj*

then begin
dependent k (i) =true; if k=i
then declare that *Pi* is deadlocked else for all *Pm*and *Pn* such that

(a') *Pk* is locally dependent upon *Pm*, and
(b') *Pm* is waiting on *Pn*, and

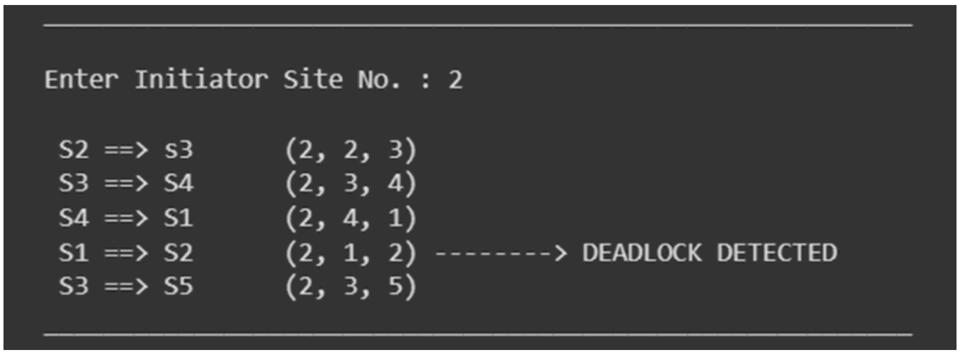
(c’) *Pm* and *Pn* are on different sites, send a probe (i, m, n) to the home site of *Pn* end.

CODE:



```

print('''
    P1  P2  P3  P4  P5

P1  0   1   0   0   0
P2  0   0   1   0   0
P3  0   0   0   1   1
P4  1   0   0   0   0
P5  0   0   0   0   0
''')

a = [   [0, 1, 0, 0, 0],
        [0, 0, 1, 0, 0],
        [0, 0, 0, 1, 1],
        [1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0] ]

flag = 0

def aman(a, i, k):

    end = 5

    for x in range(end):

        if(a[k][x] == 1):

            if(i == x):

                print(f' S{k+1} ==> S{x+1}      ({i+1}, {k+1}, {x+1}) --------> DEADLOCK DETECTED')
                global flag
                flag = 1
                break

            print(f' S{k+1} ==> S{x+1}      ({i+1}, {k+1}, {x+1})')
            aman(a,i,x)


print("CHANDY-MISRA-HAAS DISTRIBUTED DEADLOCK DETECTION ALGORITHM")
print("_________________________________________________________")
print()

x = 0
end = 5
i = int(input("Enter Initiator Site No. : "))
j = i - 1

print()
for k in range(end):

    if(a[j][k]==1):

        print(f' S{j+1} ==> s{k+1}      ({i}, {j+1}, {k+1})')
        aman(a,j,k)


if(flag == 0):
    print("\nNO DEADLOCK DETECTED")

print("_________________________________________________________")


```

OUTPUT:



```
______________________________________________________

Enter Initiator Site No. : 2

 S2 ==> s3      (2, 2, 3)
 S3 ==> S4      (2, 3, 4)
 S4 ==> S1      (2, 4, 1)
 S1 ==> S2      (2, 1, 2) --------> DEADLOCK DETECTED
 S3 ==> S5      (2, 3, 5)

______________________________________________________
```

Observations and Findings:

- Distributed deadlock detection is a technique used to detectdeadlocks in distributed systems.
- Deadlocks can occur in distributed systems when multiple processes complete for shared resources, and each process is waiting for a resource that is being held by another process.
- Distributed deadlock detection algorithms use a distributed graph representation to model the resource allocation graph in the system.
- These algorithms use message passing to exchange information aboutresource allocations and requests between processes.
- When a deadlock is detected, the algorithms can take corrective actions, such as aborting one of the processes or releasing some resources, to break the deadlock.

Conclusion:

- Distributed deadlock detection is an important technique for ensuring the correctness and reliability of distributed systems.
- The algorithms used for distributed deadlock detection can be complex require a significant number of computational resources.
- The message passing required for distributed deadlock detection can add to the overhead of the system, affecting its performance.
- Distributed deadlock detection algorithms can help identify potential deadlocks and provide mechanism for taking corrective actions to prevent system failures.
- Distributed deadlock detection can be used in conjunction with other techniques, such as resource allocation and scheduling, to provide a robust and reliabledistributed system

SIGN AND REMARK

DATE

R1 **(4 Marks)**	**R2** **(4 Marks)**	**R3** **(4 Marks)**	**R4** **(3 Mark)**	**Total** **(15 Marks)**	**Signature**

EXPERIMENT No. 8

Date of Performance:

Date of Submission:

Aim: To implement load balancing.

Theory:

Load balancing is the way of distributing load units (jobs or tasks) across a set of processors which are connected to a network which may be distributed across the globe.
• The excess load or remaining unexecuted load from a processor is migrated to other processors which have load below the threshold load.
• Threshold load is such an amount of load to a processor that any load may come further to that processor.
• By load balancing strategy it is possible to make every processor equally busy and to finish the works approximately at the same time.

Load-balancing approach:

Issues in designing Load-balancing algorithms:

1. **Load estimation policy**
    - determines how to estimate the workload of a node
2. **Process transfer policy**
    - determines whether to execute a process locally or remote
3. **State information exchange policy**
    - determines how to exchange load information among nodes
4. **Location policy**
    - determines to which node the transferable process should be sent
5. **Priority assignment policy**
    - determines the priority of execution of local and remote processes
6. **Migration limiting policy**
    - determines the total number of times a process can migrate

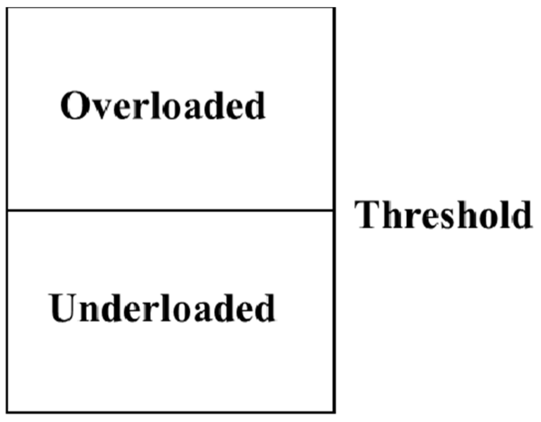
1Load estimation policy

- To balance the workload on all the nodes of the system, it is necessary to decide how to measure the workload of a particular node
- Some measurable parameters (with time and node dependent factor) can be the following:
    - Total number of processes on the node
    - Resource demands of these processes
    - Instruction mixes of these processes

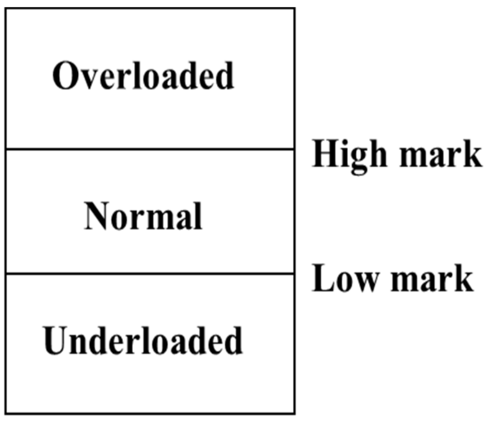
Architecture and speed of the node's processor
Several load-balancing algorithms use the total number of processes to achieve big efficiency.

2) Process transfer policy

- Most of the algorithms use the threshold policy to decide on whether the node is lightly-loaded or heavily-loaded
- Threshold value is a limiting value of the workload of node which can be determined by
  - Static policy: predefined threshold value for each node depending on processing capability
  - Dynamic policy: threshold value is calculated from average workload and a predefined constant
- Below threshold value node accepts processes to execute, above threshold value node tries to transfer processes to a lightly-loaded node
- Single-threshold policy may lead to unstable algorithm because underloaded node could turn to be overloaded right after a process migration



Single-threshold policy



Double-threshold policy

3) State information exchange policy:

- Dynamic policies require frequent exchange of state information, but these extra messages arise two opposite impacts:
- Increasing the number of messages gives more accurate scheduling decision
- Increasing the number of messages raises the queuing time of messages
- State information policies can be the following:
  - Periodic broadcast
  - Broadcast when state changes
  - On-demand exchange
  - Exchange by polling

4) Location policy:

a)Threshold method

Policy selects a random node, checks whether the node is able to receive the process, then transfers the process. If node rejects, another node is selected randomly. This continues until probe limit is reached.

b)Shortest method:

L distinct nodes are chosen at random; each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits to accept the process.

Simple improvement is to discontinue probing whenever a node with zero load is encountered.

L distinct nodes are chosen at random; each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits to accept the process.

Simple improvement is to discontinue probing whenever a node with zero load is encountered

5. Priority assignment policy:

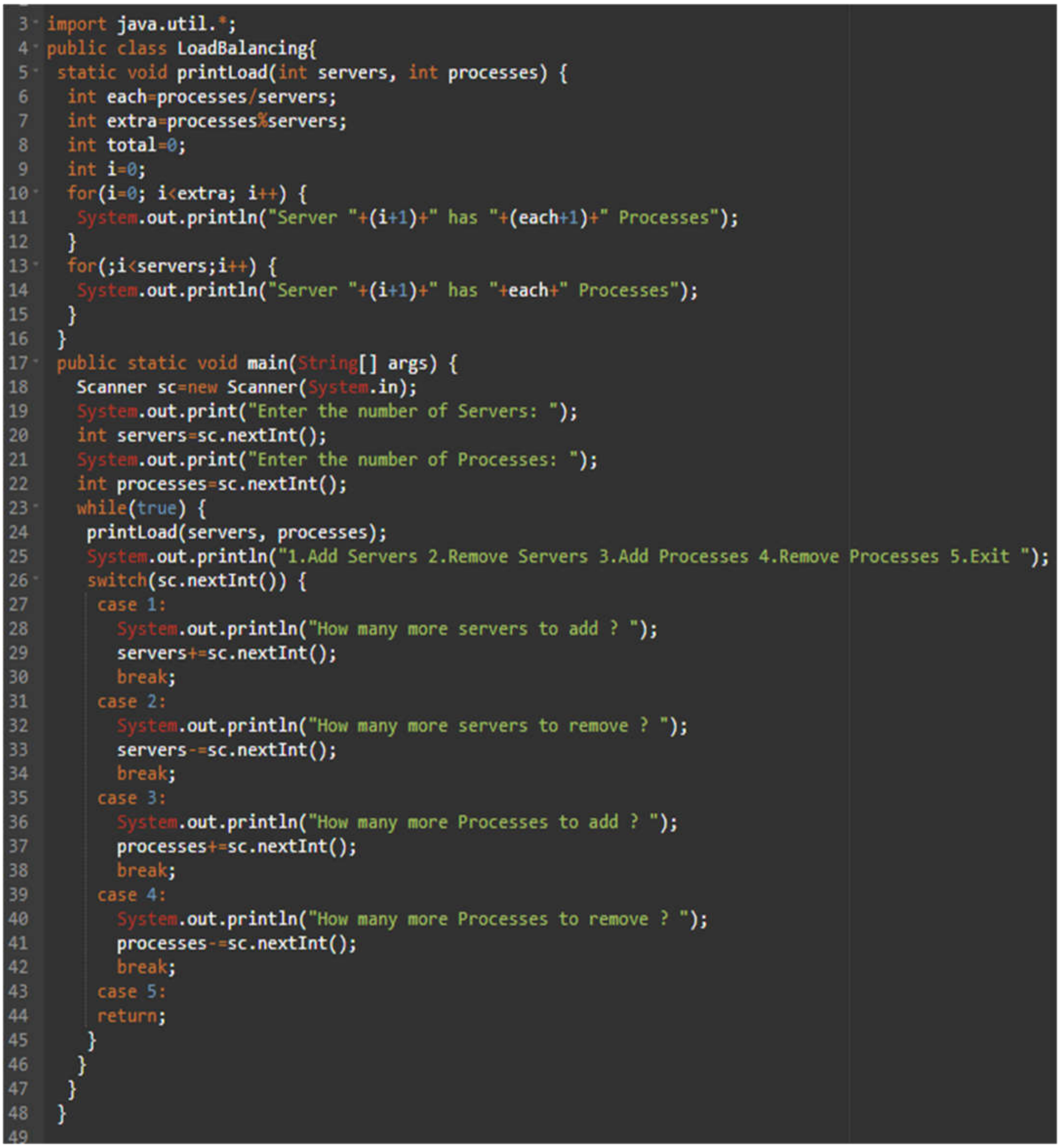
- Selfish
  - Local processes are given higher priority than remote processes. Worst response time performance of the three policies.
- Altruistic
  - Remote processes are given higher priority than local processes. Best response time performance of the three policies.
- Intermediate
  - When the number of local processes is greater or equal to the number of remote processes, local processes are given higher priority than remote processes. Otherwise, remote processes are given higher priority than local processes.

6. Migration limiting policy:

This policy determines the total number of times a process can migrate.

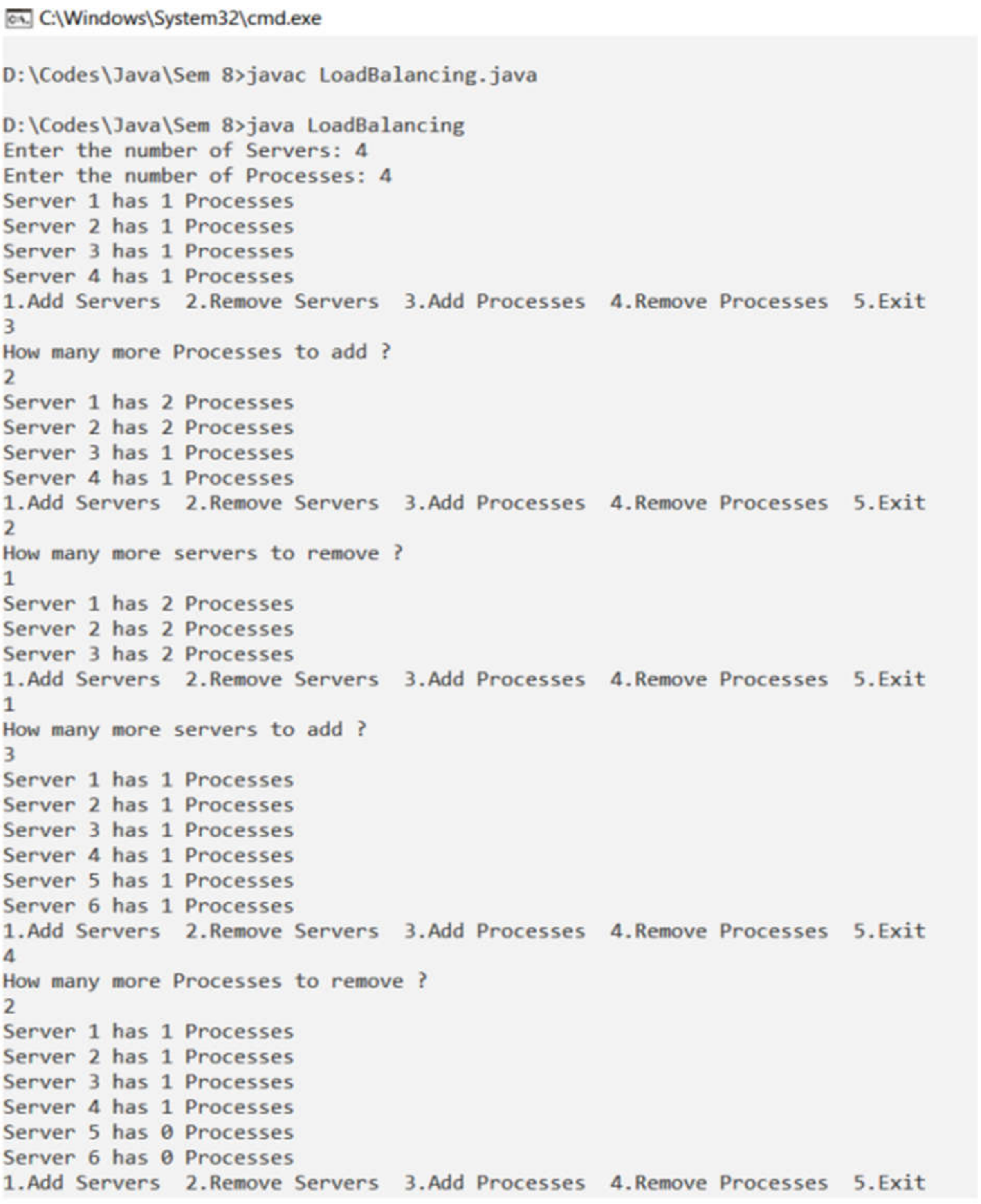
- Uncontrolled
  - A remote process arriving at a node is treated just as a process originating at a node, so a process may be migrated any number of times
- Controlled
  - Avoids the instability of the uncontrolled policy
  - Use a migration count parameter to fix a limit on the number of times a process can migrate
  - Irrevocable migration policy: migration count is fixed to 1
  - For long execution processes migration count must be greater than 1 to adapt for dynamically changing states

CODE:



```java
import java.util.*;
public class LoadBalancing{
 static void printLoad(int servers, int processes) {
  int each=processes/servers;
  int extra=processes%servers;
  int total=0;
  int i=0;
  for(i=0; i<extra; i++) {
   System.out.println("Server "+(i+1)+" has "+(each+1)+" Processes");
  }
  for(;i<servers;i++) {
   System.out.println("Server "+(i+1)+" has "+each+" Processes");
  }
 }
 public static void main(String[] args) {
   Scanner sc=new Scanner(System.in);
   System.out.print("Enter the number of Servers: ");
   int servers=sc.nextInt();
   System.out.print("Enter the number of Processes: ");
   int processes=sc.nextInt();
   while(true) {
    printLoad(servers, processes);
    System.out.println("1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit ");
    switch(sc.nextInt()) {
     case 1:
       System.out.println("How many more servers to add ? ");
       servers+=sc.nextInt();
       break;
     case 2:
       System.out.println("How many more servers to remove ? ");
       servers-=sc.nextInt();
       break;
     case 3:
       System.out.println("How many more Processes to add ? ");
       processes+=sc.nextInt();
       break;
     case 4:
       System.out.println("How many more Processes to remove ? ");
       processes-=sc.nextInt();
       break;
     case 5:
     return;
    }
   }
  }
 }
```

Output:


C:\Windows\System32\cmd.exe

```
D:\Codes\Java\Sem 8>javac LoadBalancing.java

D:\Codes\Java\Sem 8>java LoadBalancing
Enter the number of Servers: 4
Enter the number of Processes: 4
Server 1 has 1 Processes
Server 2 has 1 Processes
Server 3 has 1 Processes
Server 4 has 1 Processes
1.Add Servers  2.Remove Servers  3.Add Processes  4.Remove Processes  5.Exit
3
How many more Processes to add ?
2
Server 1 has 2 Processes
Server 2 has 2 Processes
Server 3 has 1 Processes
Server 4 has 1 Processes
1.Add Servers  2.Remove Servers  3.Add Processes  4.Remove Processes  5.Exit
2
How many more servers to remove ?
1
Server 1 has 2 Processes
Server 2 has 2 Processes
Server 3 has 2 Processes
1.Add Servers  2.Remove Servers  3.Add Processes  4.Remove Processes  5.Exit
1
How many more servers to add ?
3
Server 1 has 1 Processes
Server 2 has 1 Processes
Server 3 has 1 Processes
Server 4 has 1 Processes
Server 5 has 1 Processes
Server 6 has 1 Processes
1.Add Servers  2.Remove Servers  3.Add Processes  4.Remove Processes  5.Exit
4
How many more Processes to remove ?
2
Server 1 has 1 Processes
Server 2 has 1 Processes
Server 3 has 1 Processes
Server 4 has 1 Processes
Server 5 has 0 Processes
Server 6 has 0 Processes
1.Add Servers  2.Remove Servers  3.Add Processes  4.Remove Processes  5.Exit
```

Observations and Findings:

- Load balancing distributes workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units or disk drives.
- Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy.
- Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process. Load balancing is dividing the amount of work that a computer has to do between two or more computers so that more work gets done in the same amount of time and, in general, all users get served faster.

Conclusion:

Load balancing aims to:

- optimize resource use,
- maximize throughput,
- minimize response time, and
- avoid overload of any single resource.

Load balancing can be implemented with hardware, software, or a combination of both.

SIGN AND REMARK

DATE

R1 (4 Marks)	**R2** (4 Marks)	**R3** (4 Marks)	**R4** (3 Mark)	**Total** (15 Marks)	**Signature**

Experiment No. 9

Date of Performance:

Date of Submission:

Aim: To study Andrew File System (AFS)

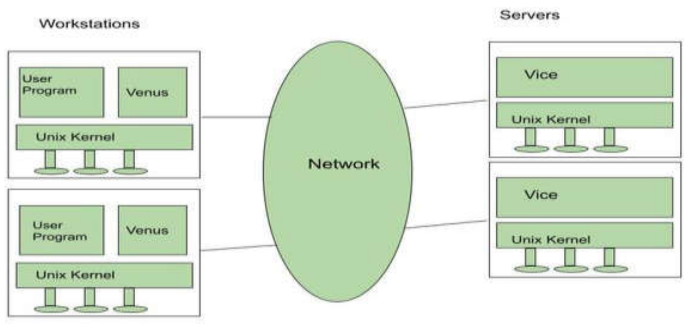
Theory:

Morris et al. 1986 created a distributed computing environment for use as a campus computer and information system at Carnegie Mellon University (CMU). An AFS is used by businesses to make it easier for AFS client workstations in different locations to access stored server files. It presents a homogeneous, location-independent file namespace to all client workstations via a group of trustworthy servers. After login onto workstations that communicate inside the Distributed Computing Infrastructure, users exchange data and programs (DCI). The goal is to facilitate large-scale information exchange by reducing client-server communication. This is accomplished by moving whole files between server and client computers and caching them until the servers get a more recent version. An AFS uses a local cache to improve speed and minimize effort in dispersed networks. A server, for example, replies to a workstation request by storing data in the workstation's local cache.

Andrew File System Architecture:

- **Vice:** The Andrew File System provides a homogeneous, location-transparent file namespace to all client workstations by utilizing a group of trustworthy servers known as Vice. The Berkeley Software Distribution of the Unix operating system is used on both clients and servers. Each workstation's operating system intercepts file system calls and redirects them to a user-level process on that workstation.
- **Venus:** This mechanism, known as Venus, caches files from Vice and returns updated versions of those files to the servers from which they originated. Only when a file is opened or closed does Venus communicate with Vice; individual bytes of a file are read and written directly on the cached copy, skipping Venus

This file system architecture was largely inspired by the need for scalability. To increase the number of clients a server can service, Venus performs as much work as possible rather than Vice. Vice only keeps the functionalities that are necessary for the file system's integrity, availability, and security. The servers are set up as a loose confederacy with little connectivity between them.



The following are the server and client components used in AFS networks:

- Any computer that creates requests for AFS server files hosted on a network qualifies as a client.
- The file is saved in the client machine's local cache and shown to the user once a server responds and transmits a requested file.
- When a user visits the AFS, the client sends all modifications to the server via a call back mechanism. The client machine's local cache stores frequently used files for rapid access.

AFS implementation:

- Client processes communicate with a UNIX kernel via standard system calls.
- The kernel is tweaked significantly to identify references to Vice files in relevant activities and route requests to the workstation's Venus client process.
- If a volume is missing from this cache, Venus contacts any server it already has a connection with, asks for the location information, and enters it into the mapping cache. Venus makes a new connection to the server unless it already has one. The file or directory is then retrieved using this connection.
- Authentication and security need the establishing of a connection. A copy of the target file is made on the local disc when it is located and cached.
- Venus then returns to the kernel, which opens the cached copy and gives the client process the handle to it. Both AFS servers and clients use the UNIX file system as a low-level storage system. On the workstation's disc, the client cache is a local directory. This directory contains files with placeholder names for cache entries.
- Both Venus and server processes use the latter's modes to access UNIX files directly, avoiding the costly path-name-to-anode conversion method.
- When a file is deleted from the cache, Venus informs the relevant server that the call back for that file has been removed.

Locating your AFS home directory

The path to the directory is:

```
/afs/cs.cmu.edu/user/example
```

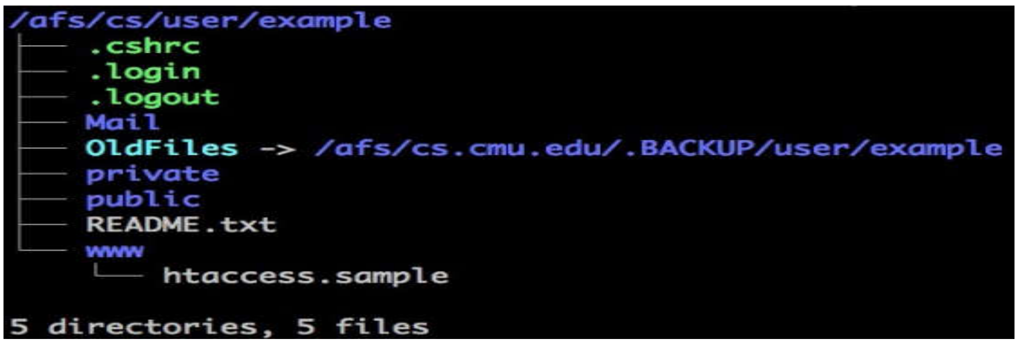
This path can be abbreviated to /afs/cs/user/example on must CMU hosts. Replace 'example' with your SCS username.

Permissions in AFS

AFS file and directory permissions do not work like regular Linux/Unix permissions.
AFS uses Access Control lists (ACLs) to protect directories. New directories that you create in AFS will inherit permissions from their parent directory. When created, your AFS home directory has permissions set so that anyone can list the names of files in it, but only you can read the contents of those files.

Default AFS Home Directory Setup

When your AFS home directory is set up, it is automatically populated with a few files and directories:



```
/afs/cs/user/example
├── .cshrc
├── .login
├── .logout
├── Mail
├── OldFiles -> /afs/cs.cmu.edu/.BACKUP/user/example
├── private
├── public
├── README.txt
└── www
    └── htaccess.sample

5 directories, 5 files
```

AFS uses Access Control Lists (ACLs) to determine permissions for accessing data. An ACL is a set of Kerberos instances, IP addresses, and/or AFS Groups along with an associated AFS permission.

For example, the ACL for the directory /afs/cs.cmu.edu/user/example has entries for:

```
wwwsrv:http-ftp rl
system:anyuser rl
example rlidwka
```

The above ACL gives just "read" and "lookup" rights to the special groups wwwsrv:http-ftp and system:anyuser, and all AFS ACL permissions to the user "example".

ACLs allow very flexible control over who may access data in AFS. Some features of ACLs and AFS access permissions are:

- ACLs apply only to directories in AFS, not to files.
- AFS ignores standard Unix permissions (the ones you set with the chmod command), with the exception of the file owner mode bits (see the section below on protecting individual files for details).
- The owner of a directory can always change the ACL on that directory, no matter what the ACL is (so you can fix things if you accidentally remove yourself from the ACL of a directory you own).
- When you create a directory, it automatically inherits the ACL of its parent directory.
- In order to access a subdirectory, one must have "l" (lookup) permissions on all parent directories.

How to list an ACL

- The command fs listacl <directory-name> will list the ACL of a directory. You can abbreviate "listacl" to "la" for the same result.

```
fs la /afs/cs/user/example
```

- will produce the output:

```
Access list for /afs/cs/user/example is
    Normal rights:
    wwwsrv:http-ftp rl
    system:anyuser rl
    example rlidwka
```

- You can use the command fs help listacl to list the complete set of options.

How to add a user or group to an ACL

- The command fs setacl -dir <directory> -acl <acl entries> will add the given ACLs to the given directory.

```
fs setacl -dir /afs/cs/user/example -acl jsmith rl
```

Managing AFS Authentication

How to list your AFS tokens

- The tokens command will list your AFS tokens and produce output like the following:

```
Tokens held by the Cache Manager:

User's (AFS ID 2102) tokens for afs@cs.cmu.edu [Expires Jun 13 22:04]
    --End of list--
```

To see the name of the user that corresponds to the given AFS id, use the command:

```
pts examine <AFS ID>
```

For example:

```
pts examine 2102
```

Authentication

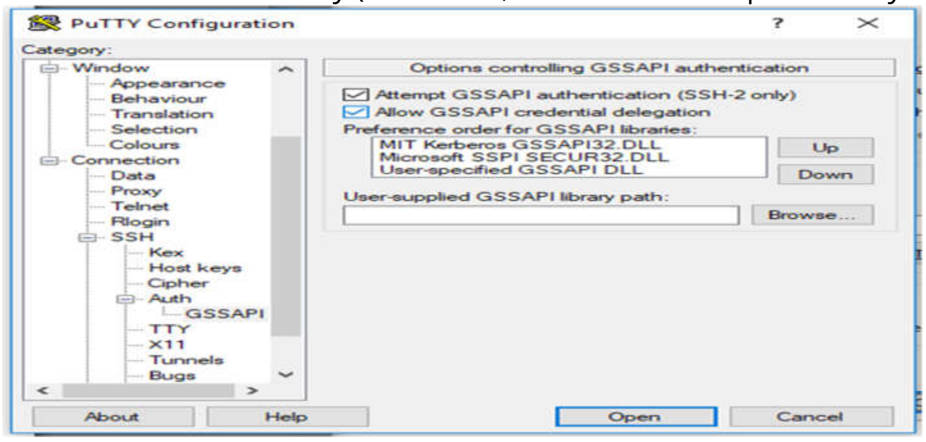
We will need your SCS Kerberos credentials to authenticate to an SCS Linux host or linux.gp.cs.cmu.edu. If you are using unix.andrew.cmu.edu, you will need your Andrew credentials and you can then perform cross-realm authentication should you need to reach a volume on our cs.cmu.edu cell.

Credential Delegation

This change will effectively get a token on login without typing a password on the target Linux host. Alternatively, users can type their Kerberos password when prompted, which will also produce a token. There may be scenarios where delegation is preferred to typing the Kerberos password, such as configuring non-interactive sessions on applicable software.

PuTTY: PuTTY has the ability to perform credential delegation. **You must be using the latest version of PuTTY (for security and functionality reasons). These options may not apply in previous versions of the software.** To delegate credentials in PuTTY:

1. It is best to save the session in Putty and apply changes specifically to it.
2. Click on the PuTTY Menu Dropdown and select "Change Settings".
3. Click on Connection, expand the SSH sub-menu, then expand the Auth sub-menu.
4. Select GSSAPI and check the box for 'Allow GSSAPI Credential Delegation'.
5. Ensure you save the Credential Delegation setting specifically to the saved session in Putty (otherwise, it will reset the option every time).



Options controlling GSSAPI authentication

☑ Attempt GSSAPI authentication (SSH-2 only)

☑ Allow GSSAPI credential delegation

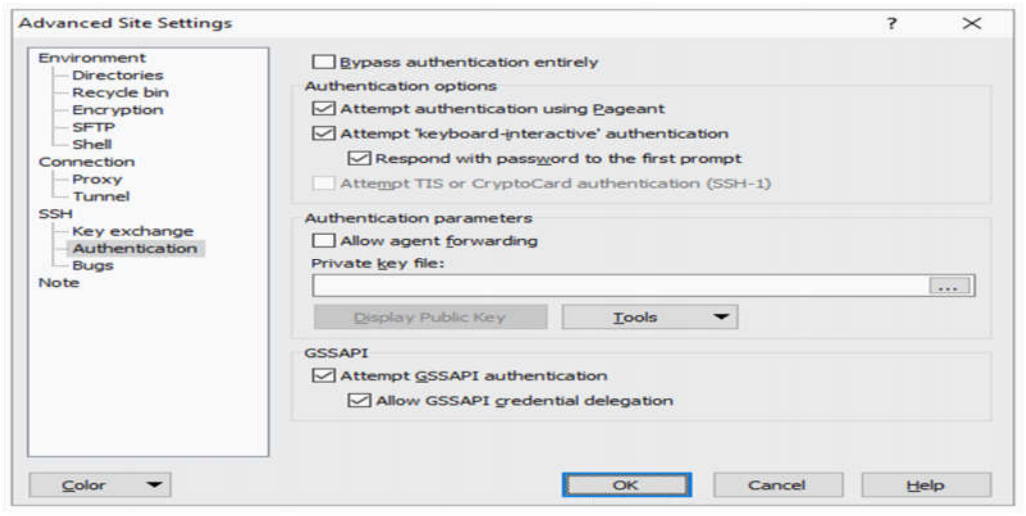
Preference order for GSSAPI libraries:

MIT Kerberos GSSAPI32.DLL
Microsoft SSPI SECUR32.DLL
User-specified GSSAPI DLL

User-supplied GSSAPI library path:

WinSCP: You can enable credential delegation in WinSCP by enabling it as a change to a saved session using the Edit option or at first new login (no saved session). **You must be using the latest version of WinSCP (for security and functionality reasons). These options may not apply in previous versions of the software.**

1. If you have a saved session, select the session and click Edit. If you do not have a saved session, create the session for the desired SCS-managed Linux host or desired target host and proceed.
2. Click Advanced.
3. Under SSH (in the now visible advanced options), select Authentication.
4. Check both boxes under GSSAPI section.
5. Click OK to finish.



☐ Bypass authentication entirely

Authentication options
- ☑ Attempt authentication using Pageant
- ☑ Attempt 'keyboard-interactive' authentication
  - ☑ Respond with password to the first prompt
- ☐ Attempt TIS or CryptoCard authentication (SSH-1)

Authentication parameters
- ☐ Allow agent forwarding
- Private key file:

GSSAPI
- ☑ Attempt GSSAPI authentication
  - ☑ Allow GSSAPI credential delegation

AFS Backups and Restores

SCS Computing Facilities perform nightly backups of AFS volumes in the **cs.cmu.edu** AFS cell (volumes within /afs/cs.cmu.edu).

Checking the latest backup of your volumes

We can access the most recent backup of your AFS volume. Many AFS volumes contain a symbolic link (symlink) called OldFiles that points to the backup volume for that AFS volume. If the symlink does not exist, you can access the backup volume directly by looking in the corresponding directory under /afs/cs.cmu.edu/.BACKUP.

For example, the backup volume for the user volume /afs/cs.cmu.edu/user/example would be located at:

```
/afs/cs.cmu.edu/.BACKUP/user/example
```

Similarly, the backup volume for the project volume /afs/cs.cmu.edu/project/superlab would be located at:

```
/afs/cs.cmu.edu/.BACKUP/project/superlab
```

Please note that this volume will contain the contents of your volume as it looked when the most recent backup was run.

Data Restores

If the data you are looking for is not in the backup directory, please submit an AFS Data Restore Request. We recommend sending restore requests as soon as possible after the data has been deleted. **Sending restore requests promptly improves the chances that we can recover the data.**

Observations and Findings:

- Shared files that aren't updated very often and local user files that aren't updated too often will last a long time.
- It sets up a lot of storage space for caching.
- It offers a big enough working set for all of a user's files, ensuring that the file is still in the cache when the user accesses it again.

Conclusion: Thus, we have seen the proper working and implementation of AFS.

SIGN AND REMARK

DATE

R1 (4 Marks)	R2 (4 Marks)	R3 (4 Marks)	R4 (3 Mark)	Total (15 Marks)	Signature

EXPERIMENT No. 10

Date of Performance :

Date of Submission :

Aim: To demonstrate CORBA Application using Java.

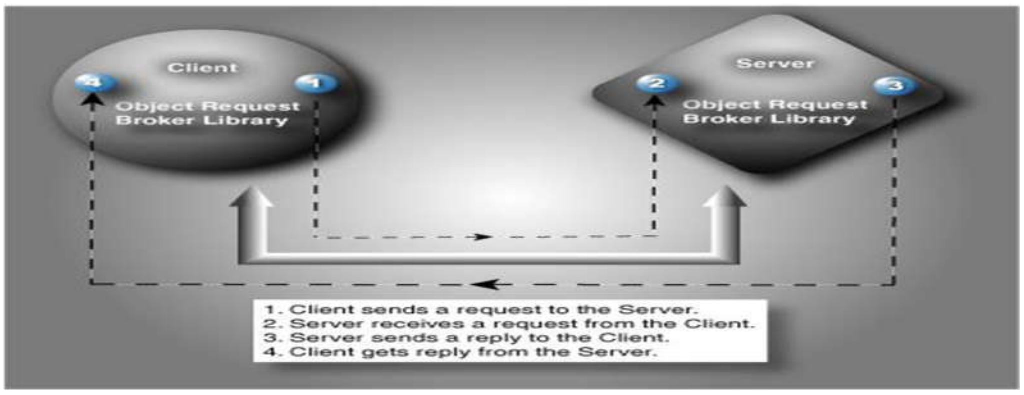
THEORY:-

The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems. 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.

The major components that make up the CORBA architecture include the:

- Interface Definition Language (IDL), which is how CORBA interfaces are defined,
- Object Request Broker (ORB), which is responsible for all interactions between remote objects and the applications that use them,
- The Portable Object Adaptor (POA), which is responsible for object activation/deactivation, mapping object ids to actual object implementations.
- Naming Service, a standard service in CORBA that lets remote clients find remote objects on the networks, and
- Inter-ORB Protocol (IIOP).

The illustration below identifies how a client sends a request to a server through the ORB:


Client
Object Request Broker Library
4
1
Server
Object Request Broker Library
2
3
1. Client sends a request to the Server.
2. Server receives a request from the Client.
3. Server sends a reply to the Client.
4. Client gets reply from the Server.

Interface Definition Language (IDL)

A cornerstone of the CORBA standards is the Interface Definition Language. IDL is the OMG standard for defining language-neutral APIs and provides the platform-independent delineation of the interfaces of distributed objects. The ability of the CORBA environments to provide consistency between clients and servers in heterogeneous environments begins with a standardized definition of the data and operations constituting the client/server interface. This standardization mechanism is the IDL, and is used by CORBA to describe the interfaces of objects.

IDL defines the modules, interfaces and operations for the applications and is not considered a programming language. The various programming languages, such as Ada, C++, or Java, supply the implementation of the interface via standardized IDL mappings.

Object Request Broker (ORB)

- The core of the CORBA architecture is the ORB. Each machine involved in a CORBA application must have an ORB running in order for processes on that machine to interact with CORBA objects running in remote processes.
- Object clients and servers make requests through their ORBs and the remote ORB locates the appropriate object and passes back the object reference to the requestor.
- The ORB provides the communication infrastructure needed to identify and locate objects, handles connection management, etc. The ORBs communicate with each other via the IIOP.

Naming Service

- The CORBA naming service provides a naming structure for remote objects.
- The CORBA standard includes specifications for inter-ORB communication protocols that transmit object requests between various ORBs running on the network.
- The protocols are independent of the particular ORB implementations running at either end. An ORB implemented in Java can talk to an ORB implemented in C, as long as they are both compliant with the CORBA standard.
- The inter-ORB protocol delivers messages between two ORBs. These messages might be method requests, return values, error messages etc.
- The inter-ORB protocol (IIOP) also deals with differences between two ORB implementations, like machine-level byte ordering etc. As a CORBA developer, you don't have to be concerned with the low-level communication protocol between ORBs. If you want two ORBs to talk, just make sure they both speak the same inter-ORB protocol (IIOP).
- The Inter-ORB Protocol (IIOP) is an inter-ORB protocol based on TCP/IP and so is extensively used on the Internet.

CORBA Server Object

POA ORB

- The POA connects the server object implementation to the ORB. It extends the functionality of the ORB and some its services include: activation and deactivation of the object implementations, generation and management of object references, mapping of object references to their implementations, and dispatching of client requests to server objects through a skeleton.

CODE:

Hello.idl

```
moduleHelloApp {

interface Hello {

stringsayHello();

oneway void shutdown();

};

};
```

HelloServer.java

```
importHelloApp.*; importorg.omg.CosNaming.*;

importorg.omg.CosNaming.NamingContextPackage.*; importorg.omg.CORBA.*;
importorg.omg.PortableServer.*; importorg.omg.PortableServer.POA; importjava.util.Properties

classHelloImpl extends HelloPOA {

private ORB orb;

public void setORB(ORB orb_val)

{ orb =orb_val;

} public String sayHello()

{ return "\nHello World !!\n";

} public void shutdown()

{ orb.shutdown(true);

}

}

public class HelloServer
```

```
{ public static void main(String args[])
{ try
{
ORB orb = ORB.init(args,null);
POA rootpoa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
rootpoa.the_POAManager().activate();
HelloImplhelloImpl= new HelloImpl(); helloImpl.setORB(orb);
org.omg.CORBA.Object ref =rootpoa.servant_to_reference(helloImpl);
 Hello href =HelloHelper.narrow(ref);
org.omg.CORBA.ObjectobjRef=orb.resolve_initial_references("NameService");

NamingContextExtncRef=NamingContextExtHelper.narrow(objRef);
String name="Hello"; NameComponentpath[]=ncRef.to_name(name); ncRef.rebind(path,href);
System.out.println("helloServer ready and waiting..."); orb.run();
} catch(Exception e)
{

System.err.println("ERROR:"+e); e.printStackTrace(System.out);
    }
    System.out.println("helloServer Exiting...");
    }
    }
```

HelloClient.java

```
    importHelloApp.*;
    importorg.omg.CosNaming.*;
    importorg.omg.CosNaming.NamingContextPa
    ckage.*; importorg.omg.CORBA.*;
    public class HelloClient
    {
```

static Hello helloImpl;

public static void main(String args[])

{ try

{

ORB orb = ORB.init(args,null);
org.omg.CORBA.ObjectobjRef=orb.resolve_initial_references("NameService");
NamingContextExtncRef =NamingContextExtHelper.narrow(objRef);

String name="Hello"; helloImpl=HelloHelper.narrow(ncRef.resolve_str(name));
System.out.println("Obtained a handle on server object :"+ helloImpl);

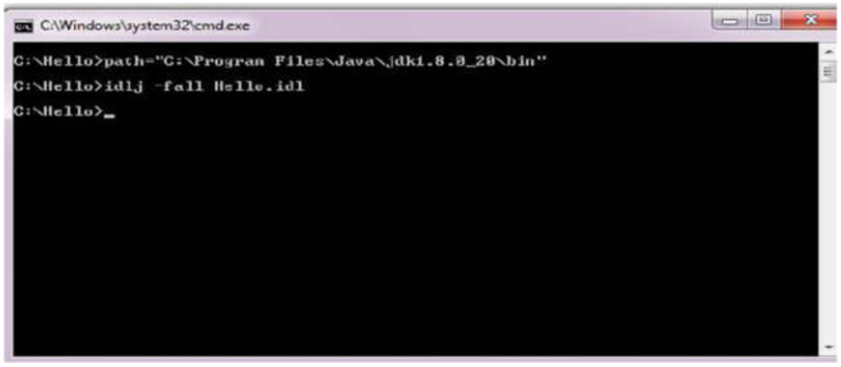
System.out.println(helloImpl.sayHello()); helloImpl.shutdown();

} catch(Exception e)

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

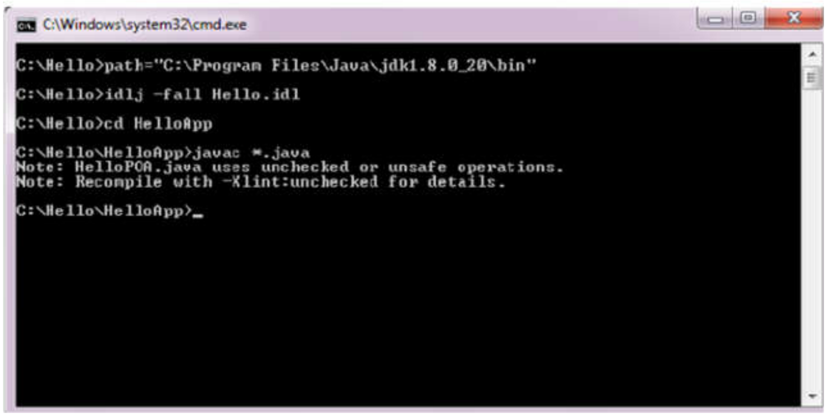
}
}
}

Output:



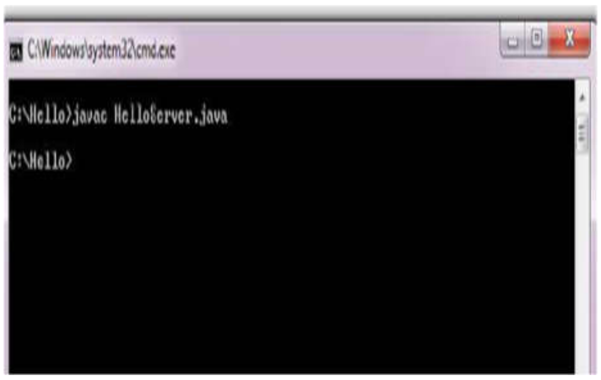
```
C:\Hello>path="C:\Program Files\Java\jdk1.8.0_20\bin"
C:\Hello>idlj -fall Hello.idl
C:\Hello>_
```

Hello file is created.

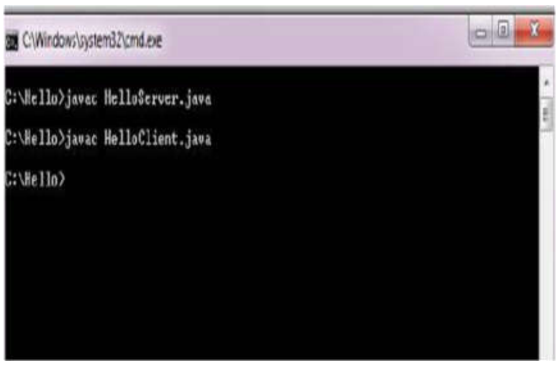


```
C:\Hello>path="C:\Program Files\Java\jdk1.8.0_20\bin"
C:\Hello>idlj -fall Hello.idl
C:\Hello>cd HelloApp
C:\Hello\HelloApp>javac *.java
Note: HelloPOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
C:\Hello\HelloApp>_
```

All the files are compiled inside HelloApp directory using javac compiler.

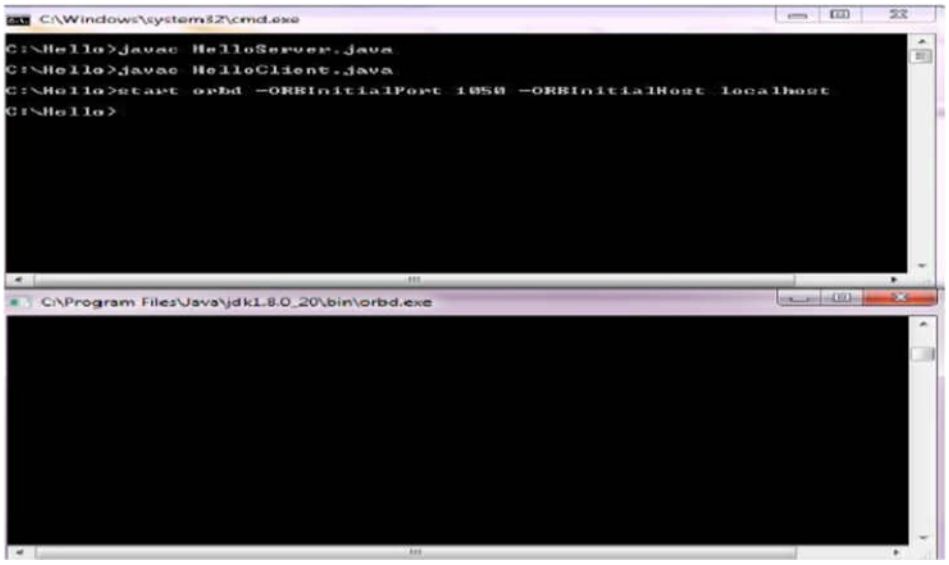

C:\Windows\system32\cmd.exe

```
C:\Hello>javac HelloServer.java

C:\Hello>
```


C:\Windows\system32\cmd.exe

```
C:\Hello>javac HelloServer.java

C:\Hello>javac HelloClient.java

C:\Hello>
```

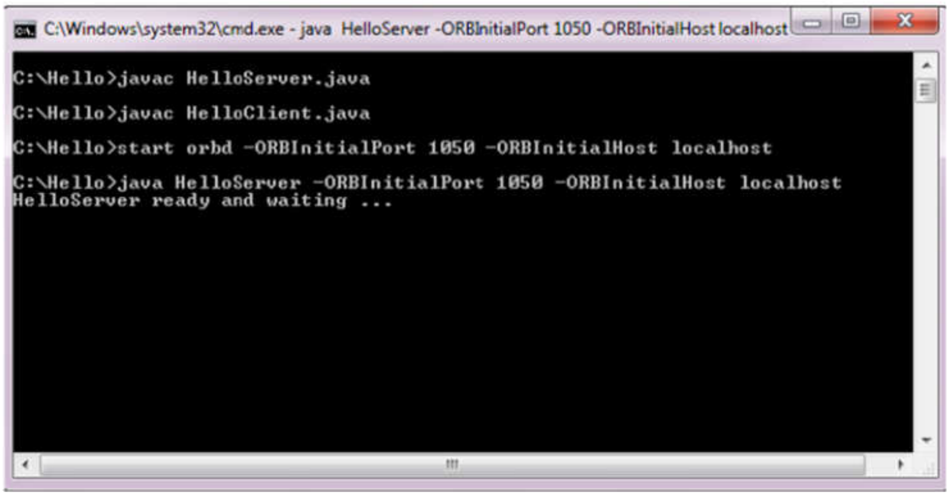
Server **Client**


C:\Windows\system32\cmd.exe

```
C:\Hello>javac HelloServer.java
C:\Hello>javac HelloClient.java
C:\Hello>start orbd -ORBInitialPort 1050 -ORBInitialHost localhost
C:\Hello>
```

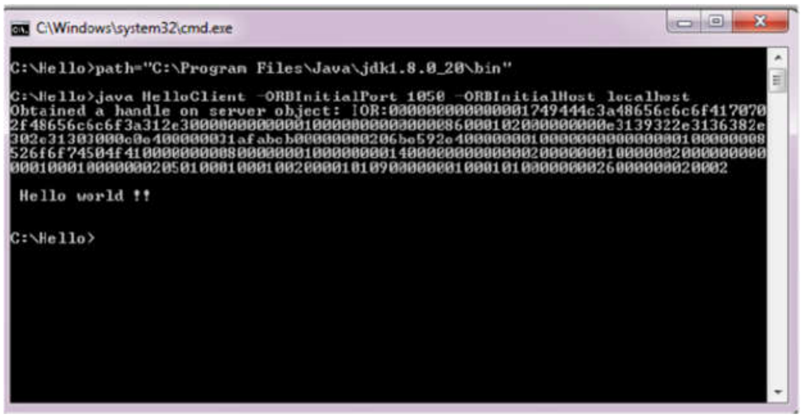
C:\Program Files\Java\jdk1.8.0_20\bin\orbd.exe

CORBA Registry


C:\Windows\system32\cmd.exe - java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost

```
C:\Hello>javac HelloServer.java

C:\Hello>javac HelloClient.java

C:\Hello>start orbd -ORBInitialPort 1050 -ORBInitialHost localhost

C:\Hello>java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost
HelloServer ready and waiting ...
```

Executed Server Program



```
C:\Windows\system32\cmd.exe

C:\Hello>path="C:\Program Files\Java\jdk1.8.0_20\bin"

C:\Hello>java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost
Obtained a handle on server object: IOR:000000000000001749444c3a48656c6c6f417070
2f48656c6c6f3a312e30000000000000010000000000000086000102000000000e3139322e3136382e
302e31303000c0e400000031afabcb00000000206be592e40000000100000000000000010000000008
526f6f74504f410000000008000000010000000014000000000000002000000010000000200000000
00010001000000020501000100010020000101090000000100010100000000260000000020002

 Hello world !!

C:\Hello>
```

Executed Client Program

Observations and Findings:

- **Common Object Request Broker Architecture (CORBA)** could be a specification of a regular design for middleware.
- It is a client-server software development model.
- CORBA server applications create CORBA objects and put object references in a naming service so that clients can call them.
- At deployment time, the node contacts a naming service to get an object reference.
- When a message arrives, the node uses the object reference to call an operation on an object in the CORBA server.

Conclusion:

CORBA's benefits include:

- language- and OS-independence,
- freedom from technology-linked implementations,
- strong data-typing,
- high level of tunability,
- and freedom from the details of distributed data transfers.

Thus, we have seen successful implementation of CORBA and benefits of the same.

SIGN AND REMARK

DATE

R1 (4 Marks)	R2 (4 Marks)	R3 (4 Marks)	R4 (3 Mark)	Total (15 Marks)	Signature