



BAHIR DAR INSTITUTE OF TECHNOLOGY
FACULTY OF ELECTRICAL AND COMPUTER
ENGINEERING
COMPUTER ENGINEERING PROGRAM

INTRODUCTION TO DISTRIBUTED SYSTEMS
LABORATORY MANUAL

Prepared

By

Adugna Necho

17-Sep-19

BAHIR DAR INSTITUTE OF TECHNOLOGY
FACULTY OF ELECTRICAL AND COMPUTER
ENGINEERING

Approval Certificate of the Reviewer

I have certified and approved the “*Introduction to Distributed systems*” *laboratory* manual which is prepared by **Adugna Necho**. As I have reviewed and commented the manual, it carries out the objectives, completes practical part of the course and fulfills the standard of the curriculum of the course.

Name

Signature

Date

Contents

Introduction.....	1
Part I: Client Server communication.....	1
Lab 1: Connection oriented client- server communication	2
Task 1- A: A client server based program using TCP to find if the number entered is prime.	2
Task 2: A client server TCP based chatting application	4
Lab 2: Connectionless client-server communication	8
Task 1: A client server based program using UDP to find if the number entered is even or odd... 8	
Task 2: A client server based program using UDP to find the factorial of the entered number. ..	10
Task 3: A multicast Socket programming.....	13
Part II: communication models	16
Lab 3: Implementation of Remote Procedure Call(RPC).	16
Task 1: Write a program to implement RPC (Remote Procedure Call).....	17
Task 2: A program to implement simple calculator operations like addition, subtraction, multiplication and division.....	19
Task 3: A program that finds the square, square root, cube and cube root of the entered number	23
Lab4: Write a program to implement remote method invocation using Java RMI.....	26
Task 1: A RMI based application program to display current date and time.....	30
Task 2: A RMI based application program that converts digits to words, e.g. 123 will be converted to one two three.	32
Lab 5: Write a program to implement Message Oriented Communication	36
Task 1: Write a server and client program that implements the following MPI primitives.....	38
Part III: process synchronization and coordination.....	43
Lab 5: Write a program to execute any one mutual exclusion algorithm.	43
Task 1: Implement a simple ring based Mutual exclusion algorithm using a single server and two clients.	44
Lab 6: Implement any one election algorithm	49
Task 1: Write a program to implement any one election algorithm	51
1. Bully Algorithm	60
2. Ring Algorithm.....	63
Lab 7: Implementation of clock synchronization algorithms.	66
Task 1: Write a program for implementation of clock synchronization algorithms.	66
Lab 8: Implementation of two phases commit protocol.	70

Task 1: Write a program to implement two phase commit protocol. 70

Introduction

This manual is intended for the computer engineering students in the course of Distributed Systems. It contains Lab Sessions related to various aspects of distributed systems to enhance understanding of students. It introduces implementation of the basic concepts of distributed systems like client-server communication using RMI, RPC, MOC, SOC, CORBA, and soon.

Part I: Client Server communication

Objective:

This section is an introduction and review of the basics of client and server communication. As we discussed in lecture classes clients and servers can communicate in one of two mechanisms. These are namely connection-less and connection-oriented communication. We are going to see sockets as a means of achieving a communication between the client and server in a connection-oriented manner whereas we will see datagram as a means of achieving a communication between the client and server in a connection-less manner.

We are going to use Java programming language for the rest of the course so it is advisable that you review the basics of Java programming language. The followings are the primary objectives of this lab session:

- ✓ Understanding the basics of client and server communication.
- ✓ Discussion of the basics of clients and servers.
- ✓ Differentiate between sockets and datagrams.
- ✓ Implementing simple client and server communication using connection-oriented communication.
- ✓ Implementing simple client and server communication using connection-less communication.

Aim: Write a program for implementing Client Server communication model using both connections oriented and connection-less communication.

Definition: A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

Definition: A *datagram* is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

Procedures:

1. Implementing the server

- a. Enable the server to listen connections on a specific port.
- b. Enable the server to accept sockets coming on that port.
- c. Create streams for writing and reading data to and from the socket.
- d. Communicate with the client through the socket using the I/O streams.
- e. Close the socket and the I/O streams when the communication is over.

2. Implementing the client

- a. Create a socket from the client to the server using the address and listening port of the server.
- b. Create streams for writing and reading data to and from the socket.
- c. Communicate with the server through the socket using the I/O streams.
- d. Close the socket and the I/O streams when the communication is over.

Lab 1: Connection oriented client- server communication

Task 1- A: A client server based program using TCP to find if the number entered is prime.

Code: client.java

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

class Client
{
    public static void main(String args[])
    {
        try{
            Socket cs = new Socket("LocalHost",8001);
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
            System.out.println("Enter a number : ");
            int a = Integer.parseInt(br.readLine());
            DataOutputStream out = new DataOutputStream(cs.getOutputStream());
            out.writeInt(a);
            DataInputStream in = new DataInputStream(cs.getInputStream());
            System.out.println(in.readUTF());
        }
    }
}
```

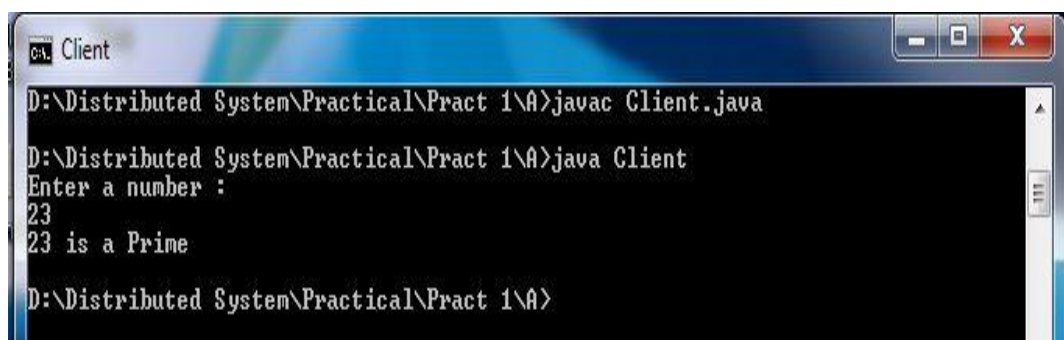
```
        cs.close();
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
}
}
```

Code: server.java

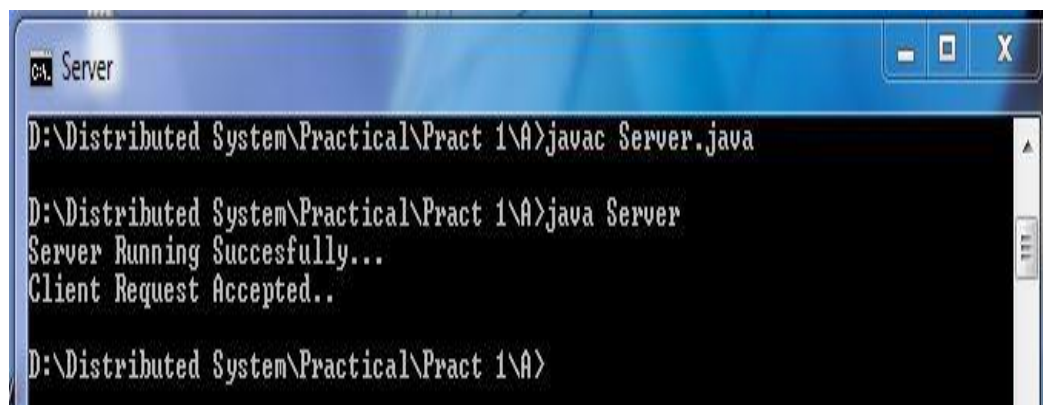
```
import java.net.*;
import java.io.*;

class Server
{
    public static void main(String args[]) {
        try{
            ServerSocket ss = new ServerSocket(8001);
            System.out.println("Server Running Succesfully... ");
            Socket s = ss.accept();
            System.out.println("Client Request Accepted..");
            DataInputStream in = new DataInputStream(s.getInputStream());
            int x = in.readInt();
            DataOutputStream otc = new DataOutputStream(s.getOutputStream());
            for(int i=2; i<=x; i++) {
                if(x%i==0) {
                    if(i==x)
                        otc.writeUTF(x + " is a Prime");
                    else
                        otc.writeUTF(x + " is not a Prime");
                    break;
                }
            }
        }
    }
}
```

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

OUTPUT: Client

```
Client  
D:\Distributed System\Practical\Pract 1\A>javac Client.java  
D:\Distributed System\Practical\Pract 1\A>java Client  
Enter a number :  
23  
23 is a Prime  
D:\Distributed System\Practical\Pract 1\A>
```

OUTPUT: Server

```
Server  
D:\Distributed System\Practical\Pract 1\A>javac Server.java  
D:\Distributed System\Practical\Pract 1\A>java Server  
Server Running Succesfully...  
Client Request Accepted..  
D:\Distributed System\Practical\Pract 1\A>
```

Task 2: A client server TCP based chatting application**Code:ChatClient.java**

```
import java.net.*;  
import java.io.*;  
class ChatClient  
{
```



```
public static void main(String args[])
{
    try{
        Socket s = new Socket("Localhost",8000);

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

        DataOutputStream out = new DataOutputStream(s.getOutputStream());
        DataInputStream in = new DataInputStream(s.getInputStream());

        String msg;
        System.out.println("To stop chatting with server type STOP");
        System.out.print("Client Says: ");
        while((msg = br.readLine()) != null)
        {
            out.writeBytes(msg+"\n");
            if(msg.equals("STOP"))
                break;

            System.out.println("Server Says : "+in.readLine());
            System.out.print("Client Says : ");
        }
        br.close();
        in.close();
        out.close();
        s.close();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

```
}
```

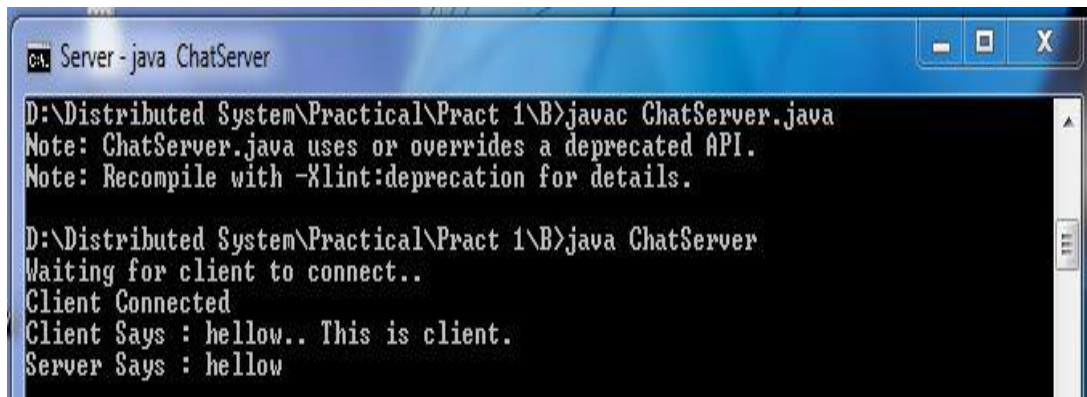
Code:ChatServer.java

```
import java.net.*;
import java.io.*;
class ChatServer
{
    public static void main(String args[]) {
        try{
            ServerSocket ss = new ServerSocket(8000);
            System.out.println("Waiting for client to connect..");
            Socket s = ss.accept();
            System.out.println("Client Connected");
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
            DataOutputStream out = new DataOutputStream(s.getOutputStream());
            DataInputStream in = new DataInputStream(s.getInputStream());
            String receive, send;
            while((receive = in.readLine()) != null)
            {
                if(receive.equals("STOP"))
                    break;
                System.out.println("Client Says : "+receive);
                System.out.print("Server Says : ");
                send = br.readLine();
                out.writeBytes(send+"\n");
            }
            br.close();
            in.close();
        }
    }
}
```

```
        out.close();  
        s.close();  
    }  
    catch(Exception e)  
    {  
        e.printStackTrace();  
    }  
}  
}
```

OUTPUT: Client

```
Client - java ChatClient  
D:\Distributed System\Practical\Pract 1\B>javac ChatClient.java  
Note: ChatClient.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
D:\Distributed System\Practical\Pract 1\B>java ChatClient  
To stop chatting with server type STOP  
Client Says: hellow.. This is client.  
Server Says : hellow  
Client Says :
```

OUTPUT: Server

```
Server - java ChatServer  
D:\Distributed System\Practical\Pract 1\B>javac ChatServer.java  
Note: ChatServer.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
D:\Distributed System\Practical\Pract 1\B>java ChatServer  
Waiting for client to connect..  
Client Connected  
Client Says : hellow.. This is client.  
Server Says : hellow
```

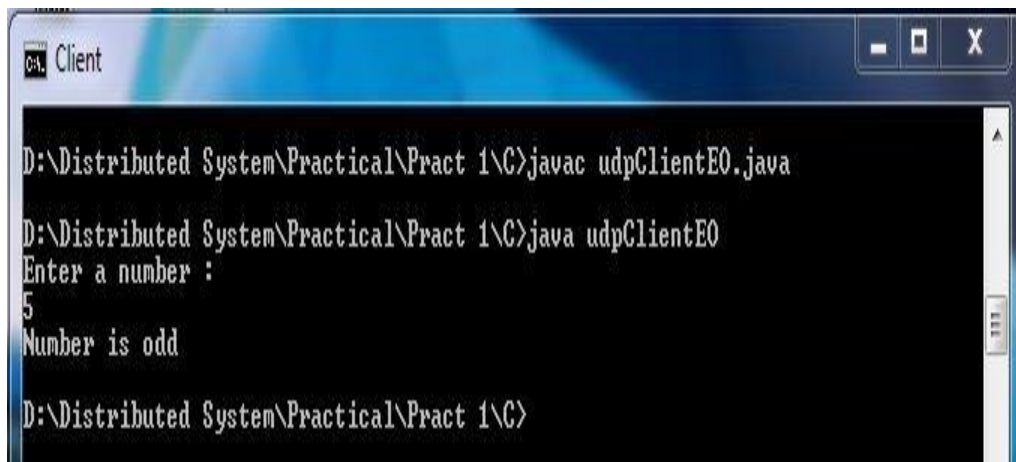
Lab 2: Connectionless client-server communication**Task 1: A client server based program using UDP to find if the number entered is even or odd.****Code: udpClientEO.java**

```
import java.io.*;
import java.net.*;
public class udpClientEO
{
    public static void main(String args[])
    {
        try{
            DatagramSocket ds = new DatagramSocket(1000);
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
            System.out.println("Enter a number : ");
            String num = br.readLine();
            byte b[] = new byte[1024]; b=num.getBytes();
            DatagramPacket dp = new
            DatagramPacket(b,b.length,InetAddress.getLocalHost(),2000);
            ds.send(dp);
            byte b1[] = new byte[1024];
            DatagramPacket dp1 = new DatagramPacket(b1,b1.length);
            ds.receive(dp1);
            String str = new String(dp1.getData(),0,dp1.getLength());
            System.out.println(str);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

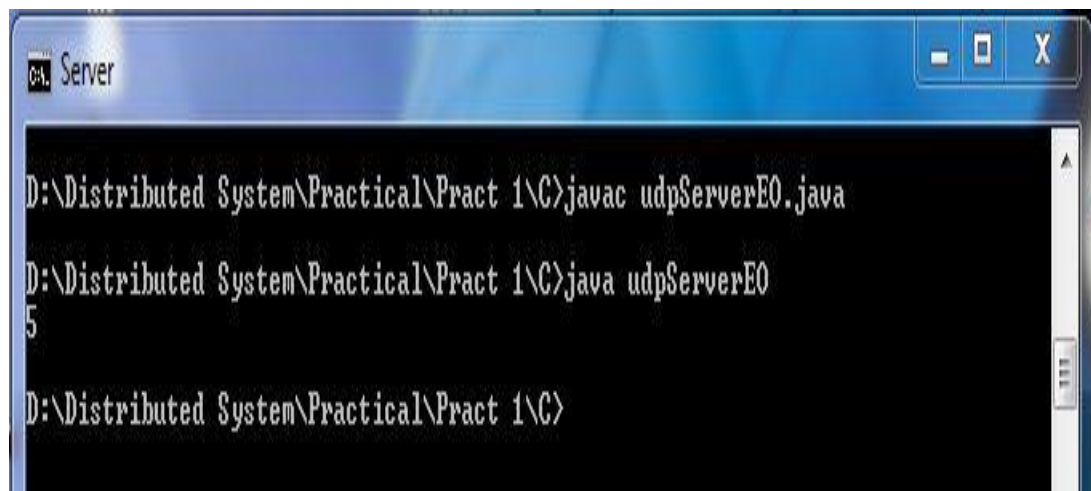
Code: udpServerEO.java

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

public class udpServerEO
{
    public static void main(String args[])
    {
        try{
            DatagramSocket ds = new DatagramSocket(2000);
            byte b[] = new byte[1024];
            DatagramPacket dp = new DatagramPacket(b,b.length);
            ds.receive(dp);
            String str = new String(dp.getData(),0,dp.getLength());
            System.out.println(str);
            int a= Integer.parseInt(str);
            String s= new String();
            if (a%2 == 0)
                s = "Number is even";
            else
                s = "Number is odd";
            byte b1[] = new byte[1024]; b1 = s.getBytes();
            DatagramPacket dp1 = new
            DatagramPacket(b1,b1.length,InetAddress.getLocalHost(),1000);
            ds.send(dp1);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Output: Client

```
D:\Distributed System\Practical\Pract 1\C>javac udpClientE0.java
D:\Distributed System\Practical\Pract 1\C>java udpClientE0
Enter a number :
5
Number is odd
D:\Distributed System\Practical\Pract 1\C>
```

Output: Server

```
D:\Distributed System\Practical\Pract 1\C>javac udpServerE0.java
D:\Distributed System\Practical\Pract 1\C>java udpServerE0
5
D:\Distributed System\Practical\Pract 1\C>
```

Task 2: A client server based program using UDP to find the factorial of the entered number.

Code: udpClientFact.java

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

public class udpClientFact
{
    public static void main(String args[]) {
        try{
            DatagramSocket ds = new DatagramSocket(1000);
```

```
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        System.out.println("Enter a number : ");
        String num = br.readLine();
        byte b[] = new byte[1024];b=num.getBytes();
        DatagramPacket dp = new
        DatagramPacket(b,b.length,InetAddress.getLocalHost(),2000);
        ds.send(dp); byte b1[] = new byte[1024];
        DatagramPacket dp1 = new DatagramPacket(b1,b1.length);
        ds.receive(dp1);
        String str = new String(dp1.getData(),0,dp1.getLength());
        System.out.println(str);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
}
```

Code: udpServerFact.java

```
import java.io.*;
import java.net.*;
public class udpServerFact
{
    public static void main(String args[]){
        try {
            DatagramSocket ds = new DatagramSocket(2000);
            byte b[] = new byte[1024];
            DatagramPacket dp = new DatagramPacket(b,b.length);
```

```
        ds.receive(dp);

        String str = new String(dp.getData(),0,dp.getLength());

        System.out.println(str);

        int a= Integer.parseInt(str);    int f = 1, i;

        String s= new String();

        for(i=1;i<=a;i++)

            f=f*i;

        s=Integer.toString(f);

        String str1 = "The Factorial of " + str + " is : " + f; byte b1[] = new
        byte[1024]; b1 = str1.getBytes();

        DatagramPacket dp1 = new
        DatagramPacket(b1,b1.length,InetAddress.getLocalHost(),1000);

        ds.send(dp1);

    }

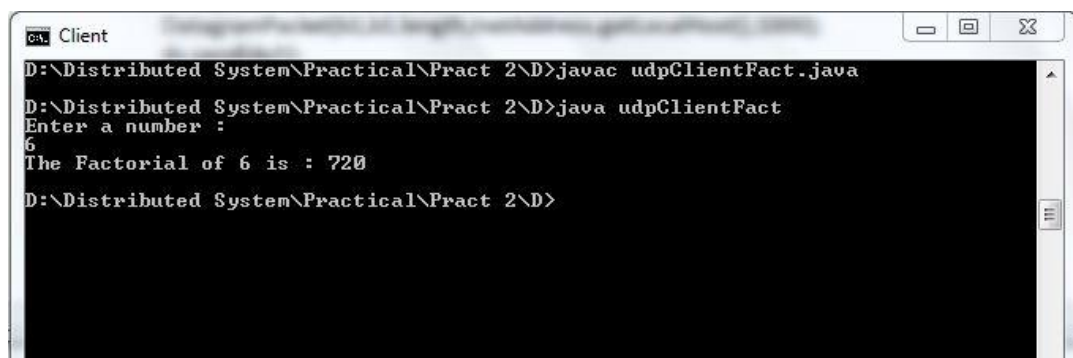
    catch(Exception e) {

        e.printStackTrace();

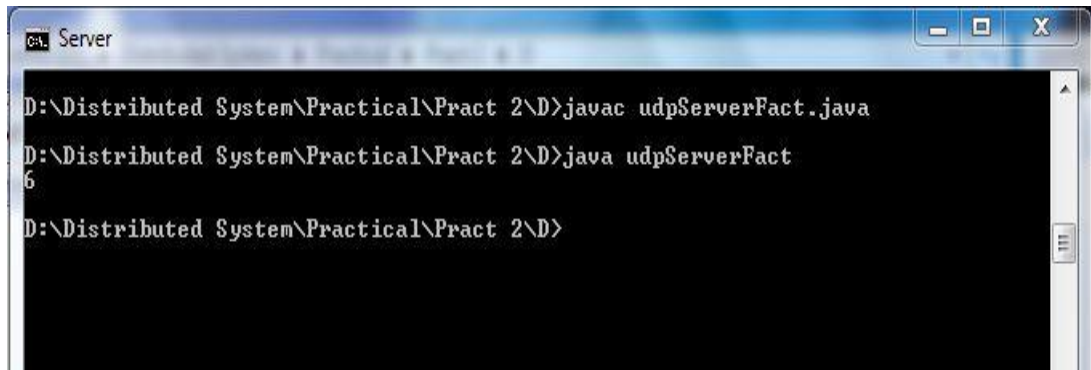
    }

}

}
```

Output: Client

```
Client
D:\Distributed System\Practical\Pract 2\D>javac udpClientFact.java
D:\Distributed System\Practical\Pract 2\D>java udpClientFact
Enter a number :
6
The Factorial of 6 is : 720
D:\Distributed System\Practical\Pract 2\D>
```


Output: Server

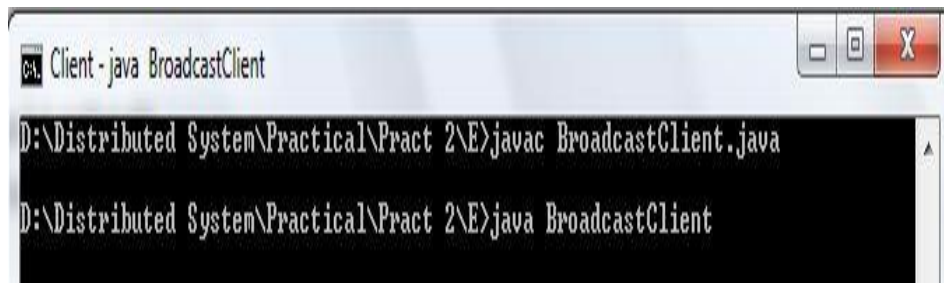
```
Server
D:\Distributed System\Practical\Pract 2\D>javac udpServerFact.java
D:\Distributed System\Practical\Pract 2\D>java udpServerFact
6
D:\Distributed System\Practical\Pract 2\D>
```

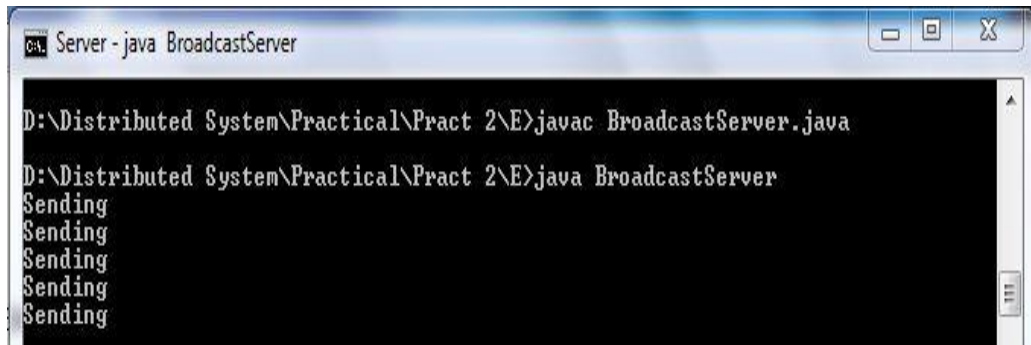
Task 3: A multicast Socket programming**Code: BroadcastClient.java**

```
import java.net.*;
import java.io.*;
public class BroadcastClient
{
    public static final int PORT = 1234;
    public static void main(String args[])throws Exception
    {
        MulticastSocket socket;
        DatagramPacket packet;
        InetAddress address;
        address = InetAddress.getByName("239.1.2.3"); socket = new
        MulticastSocket(PORT);
        //join a Multicast group and wait for a message
        socket.joinGroup(address);
        byte[] data = new byte[100];
        packet = new DatagramPacket(data,data.length);
        for(;;)
        {
            // receive the packets
            socket.receive(packet);
            String str = new String(packet.getData());
            System.out.println("Message received from "+
            packet.getAddress() + " Message is : "+str);
        }
    }
}
```

Code: BroadcastServer.java

```
import java.net.*;
import java.io.*;
import java.util.*;
public class BroadcastServer
{
    public static final int PORT = 1234;
    public static void main(String args[])throws Exception
    {
        MulticastSocket socket;
        DatagramPacket packet;
        InetAddress address;
        address = InetAddress.getByName("239.1.2.3");
        socket = new MulticastSocket();
        // join a Multicast group and send the group messages
        socket.joinGroup(address);
        byte[] data = null;
        for(;;)
        {
            Thread.sleep(10000);
            System.out.println("Sending ");
            String str = ("This is Neha Calling ");
            data = str.getBytes();
            packet = new DatagramPacket(data,
            str.length(),address,PORT); //Sends the packet
            socket.send(packet);
        }
    }
}
```

Output: Client

Output: Server

```
Server - java BroadcastServer
D:\Distributed System\Practical\Pract 2\E>javac BroadcastServer.java
D:\Distributed System\Practical\Pract 2\E>java BroadcastServer
Sending
Sending
Sending
Sending
Sending
```

Exercise: Use connection oriented and connectionless communication implementation

1. Implement a server that can perform the four arithmetic operations of addition, subtraction, division and multiplication. Enable the clients to connect with the server, supply two numbers and get the result of the four operations.
2. Implement a time server that can return the current time, whenever a client requests for it.

Part II: communication models

Lab 3: Implementation of Remote Procedure Call(RPC).

AIM: To write a java program to implement RPC (remote procedure call).

THEORY:

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports. RPC makes the client/server model of computing more powerful and easier to program. When combined with the ONC RPCGEN protocol compiler clients transparently make remote calls through a local procedure interface.

An RPC is analogous to a function call. Like a function call, when an RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure. Figure shows the flow of activity that takes place during an RPC call between two networked systems. The client makes a procedure call that sends a request to the server and waits. The thread is blocked from processing until either a reply is received, or it times out. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply to the client. After the RPC call is completed, the client program continues. RPC specifically supports network applications.

Algorithm:

- Step 1: start the program.
- Step 2: include the necessary packages in java.
- Step 3: create an interface as ucet and extends the predefined interface remote into it.
- Step 4: declare the remote methods inside the interface used.
- Step 5: declare the class rpc and extends the predefined class.
- Step 6: unicast remote object an implement the interface ucet into it.
- Step 7: declare the class server &create an object ob for the class rpc.
- Step 8: declare the class client
- Step 9: call the remote methods using the object ob which is the object of the interface ucet.
- Step 10: the remote method contains the methods such as functions(a,b),power(a,b)and log(a).
- Step 11:Stop the program.

Task 1: Write a program to implement RPC (Remote Procedure Call)**Program:****Client program:**

```
import java.rmi.*;
import java.io.*;
import java.rmi.server.*;
public class clientrpc
{
    public static void main(String arg[])
    {
        try
        {
            String serverurl="rmi://localhost/serverrpc";
            ucet ob=(ucet) Naming.lookup(serverurl);
            int r=ob.function(10,5);
            System.out.println("the answer of(a+b)^2 is:"+r);
            int t =ob.power(10,5);
            System.out.println("the answer of(a)^(b) is:"+t);
            double d=ob.log(10);
            System.out.println("the log value of the given number "+10+" is :"+d);
        }
        catch(Exception e)
        {
            System.out.println("error.."+e.getMessage());
        }
    }
}
```

Server program:

```
import java.rmi.*;
import java.rmi.server.*;
public class serverrpc
{
    public static void main(String arg[])
    {
        try
        {
            rpc ob=new rpc();
            Naming.rebind("serverrpc",ob);
        }
        catch(Exception e)
```

```
{  
}  
}  
}
```

RPC:

```
import java.rmi.*;  
import java.lang.Math.*;  
import java.rmi.server.*;  
public class rpc extends UnicastRemoteObject implements ucet  
{  
    public rpc()throws Exception  
    {  
    }  
    public int function(int a,int b)throws RemoteException  
    {  
        int m;  
        m=(a*a)+(b*b)+(2*a*b);  
        return m;  
    }  
    public int power(int a,int b)throws RemoteException  
    {  
        int m=(int)Math.pow(a,b);  
        return m;  
    }  
    public double log(int a)throws RemoteException  
    {  
        return(Math.log(a));  
    }  
}
```

Ucet:

```
import java.rmi.*;  
public interface ucet extends Remote  
{  
    public int function(int a,int b)throws RemoteException;  
    public int power(int a,int b)throws RemoteException;  
    public double log(int a)throws RemoteException;  
}
```

Output:

```
Javac ucet.java  
Start rmiregistry
```

```
Javac rpc.java
rmi rpc
javac clientrpc.java
javac serverrpc.java
javac rpc.java
javac ucet.java
```

```
javac serverrpc.java
java serverrpc
```

```
javac clientrpc.java
java clientrpc
the ans of (a+b)^2 is:225
the ans of (a)^(b) is :100000
the log value of the given number 10 is 2.30258
```

Task 2: A program to implement simple calculator operations like addition, subtraction, multiplication and division

RPCClient.java

```
import java.io.*;
import java.net.*;
class RPCClient
{
    RPCClient()
    {
        try
        {
            InetAddress ia = InetAddress.getLocalHost();
            DatagramSocket ds = new DatagramSocket();
            DatagramSocket ds1 = new DatagramSocket(1300);
            System.out.println("\nRPC Client\n");
            System.out.println("Enter method name and parameter like add 3 4\n");
            while (true)
            {
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));
                String str = br.readLine();
                byte b[] = str.getBytes();
                DatagramPacket dp = new DatagramPacket(b,b.length,ia,1200);
                ds.send(dp);
                dp = new DatagramPacket(b,b.length);
                ds1.receive(dp);
                String s = new String(dp.getData(),0,dp.getLength());
                System.out.println("\nResult = " + s + "\n");
            }
        }
    }
}
```

```
    }  
    catch (Exception e)  
    {  
        e.printStackTrace();  
    }  
}  
public static void main(String[] args)  
{  
    new RPCClient();  
}  
}
```

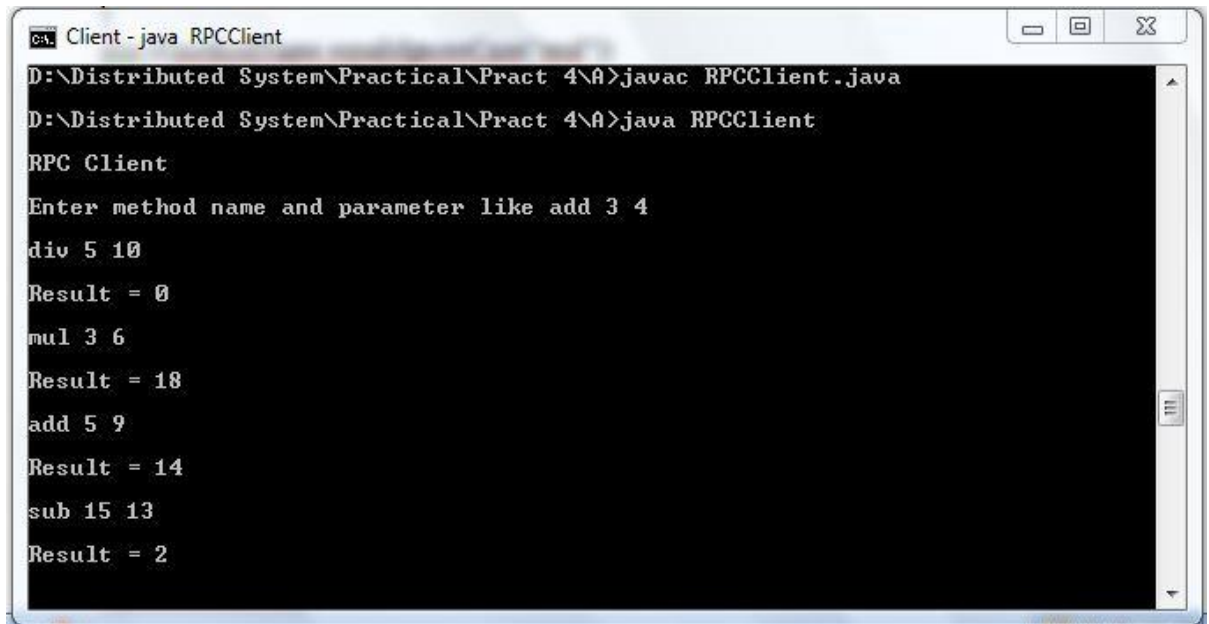
RPCServer.java

```
import java.util.*;  
import java.net.*;  
class RPCServer  
{  
    DatagramSocket ds;  
    DatagramPacket dp;  
    String str,methodName,result;  
    int val1,val2;  
    RPCServer()  
    {  
        try  
        {  
            ds=new DatagramSocket(1200);  
            byte b[]=new byte[4096];  
            while(true)  
            {  
                dp=new DatagramPacket(b,b.length);  
                ds.receive(dp);  
                str=new String(dp.getData(),0,dp.getLength());  
                if(str.equalsIgnoreCase("q"))  
                {  
                    System.exit(1);  
                }  
            }  
        }  
        catch (Exception e)  
        {  
            e.printStackTrace();  
        }  
    }  
}
```

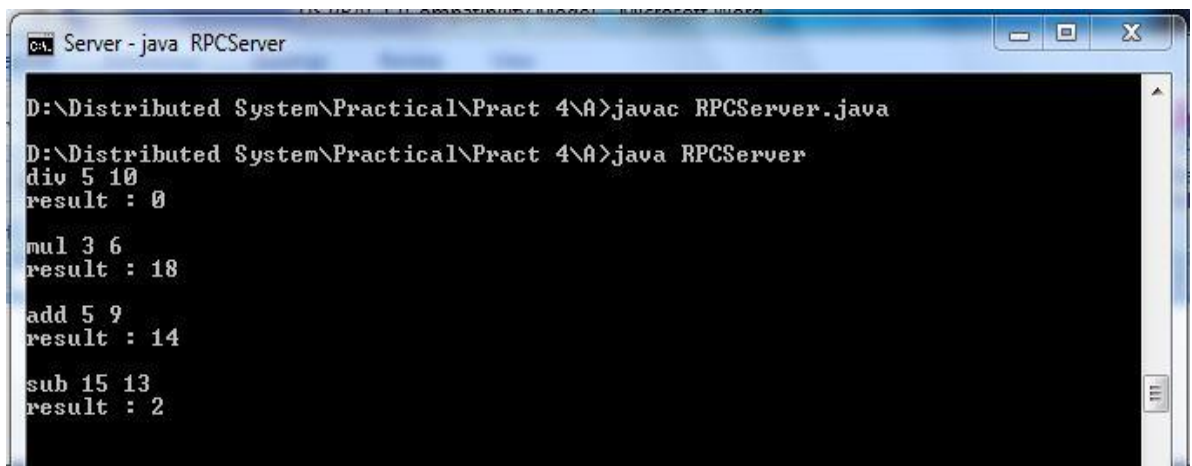


```
        else
        {
StringTokenizer st = new StringTokenizer(str," ");
        int i=0;
        while(st.hasMoreTokens())
        {
String token=st.nextToken();
        methodName=token;
        val1 = Integer.parseInt(st.nextToken());
        val2 = Integer.parseInt(st.nextToken());
        }
        }
        System.out.println(str);
        InetAddress ia = InetAddress.getLocalHost();
        if(methodName.equalsIgnoreCase("add"))
        {
        result= "" + add(val1,val2);
        }
        dName.equalsIgnoreCase("sub"))
        {
        result= "" + sub(val1,val2);
        }
        else if(methodName.equalsIgnoreCase("mul"))
        {
        result= "" + mul(val1,val2);
        }
        else if(methodName.equalsIgnoreCase("div"))
        {
        result= "" + div(val1,val2);
        }
        byte b1[]=result.getBytes();
        DatagramSocket ds1 = new DatagramSocket();
```

```
DatagramPacket dp1 = new
DatagramPacket(b1,b1.length,InetAddress.getLocalHost(), 1300);
System.out.println("result : "+result+"\n");
ds1.send(dp1);
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
public int add(int val1, int val2)
{
    return val1+val2;
}
public int sub(int val3, int val4)
{
    return val3-val4;
}
public int mul(int val3, int val4)
{
    return val3*val4;
}
public int div(int val3, int val4)
{
    return val3/val4;
}
public static void main(String[] args)
{
    new RPCServer();
}
}
```

Output: Client

```
C:\ Client - java RPCClient
D:\Distributed System\Practical\Pract 4\A>javac RPCClient.java
D:\Distributed System\Practical\Pract 4\A>java RPCClient
RPC Client
Enter method name and parameter like add 3 4
div 5 10
Result = 0
mul 3 6
Result = 18
add 5 9
Result = 14
sub 15 13
Result = 2
```

Output: Server

```
C:\ Server - java RPCServer
D:\Distributed System\Practical\Pract 4\A>javac RPCServer.java
D:\Distributed System\Practical\Pract 4\A>java RPCServer
div 5 10
result : 0
mul 3 6
result : 18
add 5 9
result : 14
sub 15 13
result : 2
```

Task 3: A program that finds the square, square root, cube and cube root of the entered number

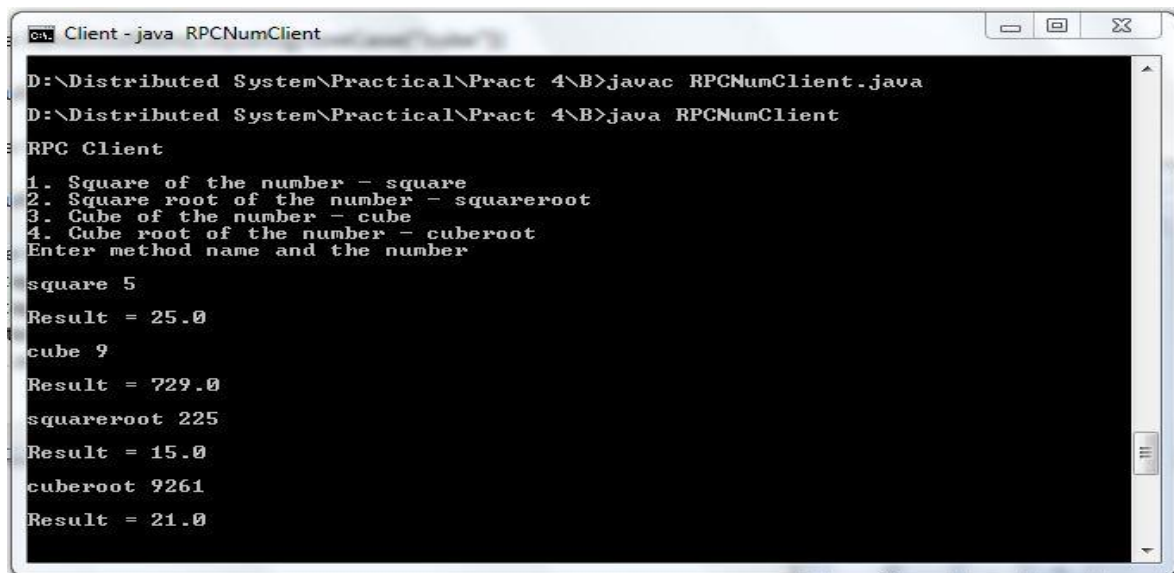
RPCNumClient.java

```
import java.io.*;
import java.net.*;
class RPCNumClient
{
    RPCNumClient()
    {
        try
```

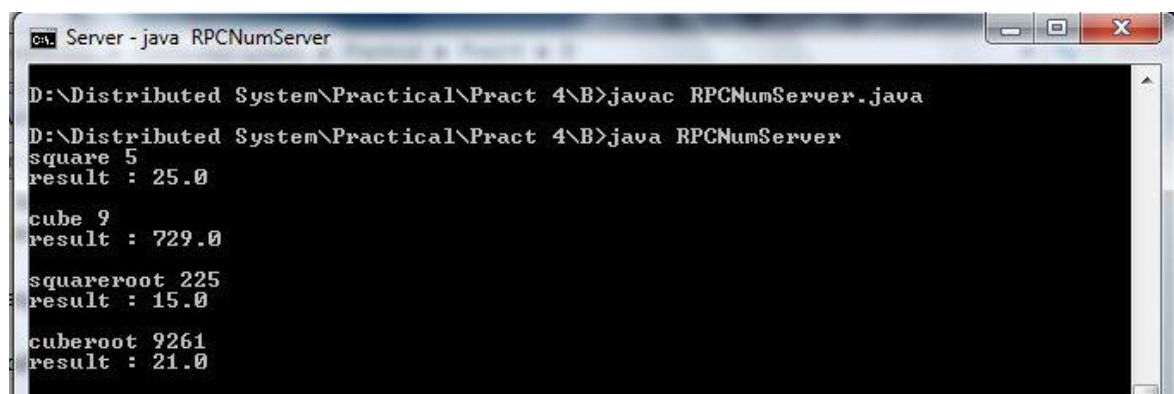
```
{  
    InetAddress ia = InetAddress.getLocalHost();  
    DatagramSocket ds = new DatagramSocket();  
    DatagramSocket ds1 = new DatagramSocket(1300);  
    System.out.println("\nRPC Client\n");  
    System.out.println("1. Square of the number - square\n2. Square root of the  
    number - squareroot\n3. Cube of the number - cube\n4. Cube root of the  
    number - cuberoot");  
    System.out.println("Enter method name and the number\n");  
    while (true)  
    {  
        BufferedReader br = new BufferedReader(new  
        InputStreamReader(System.in));  
        String str = br.readLine();  
        byte b[] = str.getBytes();  
        DatagramPacket dp = new  
        DatagramPacket(b,b.length,ia,1200);  
        ds.send(dp);  
        dp = new DatagramPacket(b,b.length);  
        ds1.receive(dp);  
        String s = new  
        String(dp.getData(),0,dp.getLength());  
        System.out.println("\nResult = " + s + "\n");  
    }  
}  
catch (Exception e)  
{  
    e.printStackTrace();  
}  
}  
public static void main(String[] args)  
{  
    new RPCNumClient();
```

}

}

Output: Client

```
Client - java RPCNumClient
D:\Distributed System\Practical\Pract 4\B>javac RPCNumClient.java
D:\Distributed System\Practical\Pract 4\B>java RPCNumClient
RPC Client
1. Square of the number - square
2. Square root of the number - squareroot
3. Cube of the number - cube
4. Cube root of the number - cuberoot
Enter method name and the number
square 5
Result = 25.0
cube 9
Result = 729.0
squareroot 225
Result = 15.0
cuberoot 9261
Result = 21.0
```

Output: Server

```
Server - java RPCNumServer
D:\Distributed System\Practical\Pract 4\B>javac RPCNumServer.java
D:\Distributed System\Practical\Pract 4\B>java RPCNumServer
square 5
result : 25.0
cube 9
result : 729.0
squareroot 225
result : 15.0
cuberoot 9261
result : 21.0
```

Exercise:

1. Write an RPC program that performs addition, subtraction, multiplication and division on the server side. Enable the client to accept the two input values and display a menu for the user to select the operations that needs to be performed?

Lab4: Write a program to implement remote method invocation using Java RMI.**Objective:**

This Lab is a demonstration on how to implement remote method invocation using Java RMI. We will be defining remote interfaces, provide their implementation in a server computer, register the remote object using java registry and finally make remote method invocations from the client computer or programs. The following are the primary objectives of this lab session:

- ✓ Understanding the basics concepts of remote method invocations.
- ✓ Discussion of how interfaces are defined for Java RMI programs.
- ✓ Be able to implement remote interface definitions.
- ✓ Be able to register remote objects using Java registry.
- ✓ Be able to make remote method invocations from the client

Aim: Write a program for implementing remote invocation using Java RMI.

Requirements:

- ✓ A computer running Java and have Java Development kit (JDK).
- ✓ Any Java IDE

Creating Distributed Applications by Using Java RMI

Using RMI to develop a distributed application involves these general steps:

- a. Designing and implementing the components of your distributed application.
- b. Compiling sources.
- c. Making classes network accessible.
- d. Starting the application.

Designing and Implementing the Application Components

First, determine your application architecture, including which components are local objects and which components are remotely accessible. This step includes:

- ✓ Defining the remote interfaces. A remote interface specifies the methods that can be invoked remotely by a client. The design of such interfaces includes the determination of the types of objects that will be used as the parameters and return values for these methods.
- ✓ Implementing the remote objects. Remote objects must implement one or more remote interfaces. The remote object class may include implementations of other

interfaces and methods that are available only locally. If any local classes are to be used for parameters or return values of any of these methods, they must be implemented as well.

- ✓ Implementing the clients. Clients that use remote objects can be implemented at any time after the remote interfaces are defined, including after the remote objects have been deployed.

The key components of the RMI architecture are:

- ✓ Server object interface: A sub-interface of `java.rmi.Remote` that defines the methods for the server object.
- ✓ Server implementation: A class that implements the remote object interface.
- ✓ Server object: An instance of the server implementation.
- ✓ RMI registry: A utility that registers remote objects and provides naming services for locating objects.
- ✓ Client program: A program that invokes the methods in the remote server object.
- ✓ Client stub: An object that resides on the client host and serves as a surrogate for the remote server object.
- ✓ Server skeleton: An object that resides on the server host, and communicates with the stub and the actual server object.

Steps in writing RMI Applications

1. **Defining the remote interfaces.** This has the general form of a normal java interface except some additional characteristics. An object becomes remote by implementing a remote Interface, which has the following characteristics:
 - ✓ A remote interface extends the interface `java.rmi.Remote`.
 - ✓ Each method of the interface declares `java.rmi.RemoteException` in its throws clause, in addition to any application-specific exceptions.

Example:

```
import java.rmi.*;

public interface AddInterface extends Remote
{
    public int sum(int n1,int n2) throws RemoteException;
}
```

2. **Implementing the remote interface.** We will write the implementation of the remote interface definition. The implementation class should provide the implementation of

the methods defined in the remote interface definition with constructors and main function. The things that need to be done in the implementation of the remote object are.

- ✓ Declare the remote interfaces being implemented
- ✓ Define the constructor for the remote object
- ✓ Provide an implementation for each remote method in the remote interfaces

Example

```
import java.rmi.*;
import java.rmi.server.*;

public class Add extends UnicastRemoteObject implements AddInterface {
    public Add() throws RemoteException{
        super();
    }

    public int sum(int n1,int n2) throws RemoteException {
        return n1+n2;
    }
}
```

UnicastRemoteObject is a convenience class, defined in the RMI public API, that can be used as a superclass for remote object implementations. The superclass UnicastRemoteObject supplies implementations for a number of java.lang.Object methods (equals, hashCode, toString) so that they are defined appropriately for remote objects.

3. **Create a main method** for the server or a separate class that will create an object of the remote object and register it to the RMI registry. You need to include the java.rmi.Naming class to be able to register objects to the RMI registry.

```
public static void main(String args[]){
    try {
        Naming.rebind("rmi://localhost:1099/Add",new Add());
        System.out.println("Server is connected and waiting for the client");
    }
    catch(Exception e) {
        System.out.println("Server could not connect: "+e);
    }
}
```


4. **Implementing the clients.** Clients that use remote objects can be implemented at any time after the remote interfaces are defined, including after the remote objects have been deployed. They simply make requests to access remote objects. The clients should locate the remote object and perform binding before they can invoke methods so they will need to contact the RMI registry to locate the remote object.

```
import java.rmi.Naming;

public class AddClient {

    public static void main(String args[]) {

        try {

            AddInterface

            ai=(AddInterface)Naming.lookup("rmi://localhost:1099/Add");

            System.out.println("The sum of 2 numbers is: "+ai.sum(10,2));

        }

        catch(Exception e){

            System.out.println("Client Exception: "+e);

        }

    }

}
```

5. **Compile the java files:** compile all java files using java compiler (javac)
6. **Run the Java RMI registry service:** In a new command window run the rmi registry. Make sure that the RMI registry is run from within the directory in which the remote object (RMI server) is implemented otherwise you will get the error ClassNotFoundException.

:> start rmiregistry

7. **Run the RMI server:** Use the java runtime to execute the RMI server

:> java AddServer

8. **Run the RMI client:** User the java runtime to execute the RMI Client

:> java AddClient

RMI works as follows:

1. A server object is registered with the RMI registry.
2. A client looks through the RMI registry for the remote object.
3. Once the remote object is located, its stub is returned in the client.

4. The remote object can be used in the same way as a local object. Communication between the client and the server is handled through the stub and the skeleton.

Task 1: A RMI based application program to display current date and time.

Procedures:

Step 1: Write the Remote Interface

InterDate.java

```
import java.rmi.*;
public interface InterDate extends Remote
{
    public String display() throws Exception;
}
```

Step 2: Write the Remote Interface implementation (RMI Server), create a remote object and bind the object.

ServerDate.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

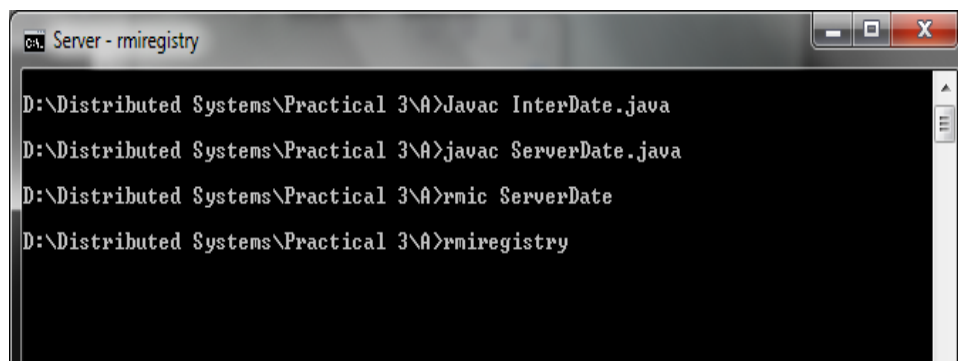
public class ServerDate extends UnicastRemoteObject implements InterDate
{
    public ServerDate() throws Exception
    {
    }
    public String display() throws Exception
    {
        String str = "";
        Date d = new Date();
        str = d.toString();
        return str;
    }
    public static void main(String args[]) throws Exception
    {
        ServerDates1 = new ServerDate();
        Naming.bind("DS", s1);
        System.out.println("Object registered.....");
    }
}
```

Step 3: Write the RMI Client.

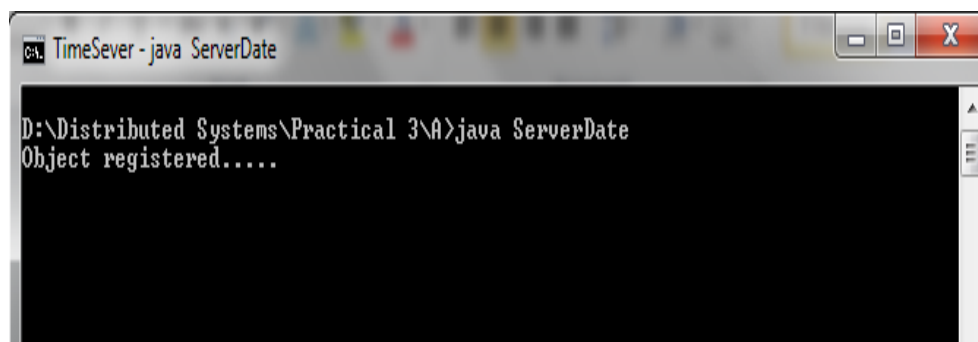
ClientDate.java

```
import java.rmi.*;
import java.io.*;
public class ClientDate
```

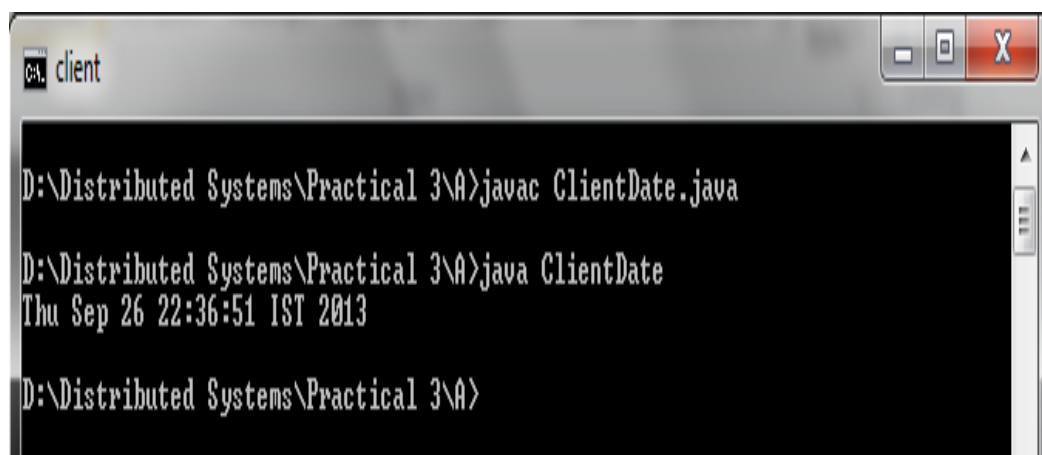
```
{  
    public static void main(String args[]) throws Exception  
    {  
        String s1;  
        InterDateh1 = (InterDate)Naming.lookup("DS");  
        s1 = h1.display();  
        System.out.println(s1);  
    }  
}
```

Output:

```
ca. Server - rmiregistry  
D:\Distributed Systems\Practical 3\A>Javac InterDate.java  
D:\Distributed Systems\Practical 3\A>javac ServerDate.java  
D:\Distributed Systems\Practical 3\A>rmic ServerDate  
D:\Distributed Systems\Practical 3\A>rmiregistry
```



```
ca. TimeSever - java ServerDate  
D:\Distributed Systems\Practical 3\A>java ServerDate  
Object registered.....
```



```
ca. client  
D:\Distributed Systems\Practical 3\A>javac ClientDate.java  
D:\Distributed Systems\Practical 3\A>java ClientDate  
Thu Sep 26 22:36:51 IST 2013  
D:\Distributed Systems\Practical 3\A>
```

Task 2: A RMI based application program that converts digits to words, e.g. 123 will be converted to one two three.

Procedures:

Step 1: Write the Remote Interface

InterConvert.java

```
import java.rmi.*;
public interface InterConvert extends Remote
{
    public String convertDigit(String no) throws Exception;
}
```

Step 2: Write the Remote Interface implementation (RMI Server), create a remote object and bind the object.

ServerConvert.java

```
import java.rmi.*;
import java.rmi.server.*;

public class ServerConvert extends UnicastRemoteObject implements InterConvert
{
    public ServerConvert() throws Exception
    {
    }
    public String convertDigit(String no) throws Exception
    {
        String str = "";
        for(int i = 0; i < no.length(); i++)
        {
            int p = no.charAt(i);
            if( p == 48)
            {
                str += "zero ";
            }
            if( p == 49)
            {
                str += "one ";
            }
            if( p == 50)
            {

```

```
        str += "two ";
    }
    if( p == 51)
    {
        str += "three ";
    }

    if( p == 52)
    {
        str += "four ";
    }
    if( p == 53)
    {
        str += "five ";
    }
    if( p == 54)
    {
        str += "six ";
    }
    if( p == 55)
    {
        str += "seven ";
    }
    if( p == 56)
    {
        str += "eight ";
    }
    if( p == 57)
    {
        str += "nine ";
    }
    }
    returnstr;
    }
    public static void main(String args[]) throws Exception
    {
        ServerConverts1 = new ServerConvert();
        Naming.bind("Wrd",s1);
        System.out.println("Object registered....");
    }
}
```

Step 3: Write the RMI Client.**ClientConvert.java**

```
import java.rmi.*;
import java.io.*;

public class ClientConvert
{
    public static void main(String args[]) throws Exception
    {
        InterConverter h1 = (InterConverter) Naming.lookup("Wrd");
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter a number : \t");
        String no = br.readLine();
        String ans = h1.convertDigit(no);
        System.out.println("The word representation of the entered digit is : " + ans);
    }
}
```

Step 4: Compile both the client and the server programs in the command line.

➔ javac CalcServer.java

➔ javac CalcClient.java

Step 5: Open a command prompt, go to the directory where the RMI server is located and start the RMI registry service.

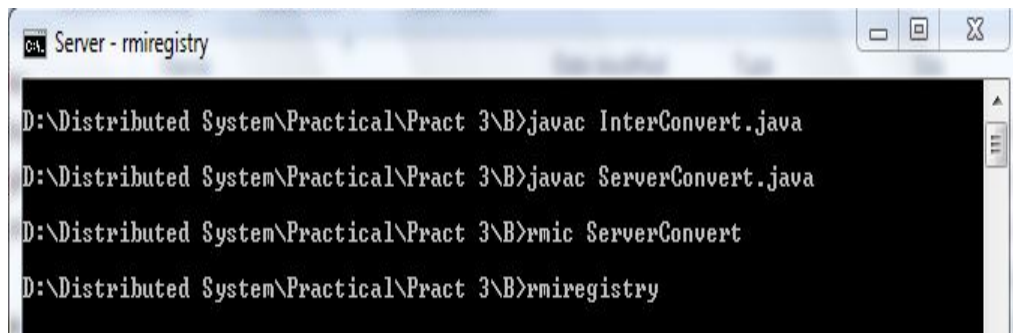
➔ start rmiregistry

Step 6: Run the RMI server in a new command prompt.

➔ java CalcServer

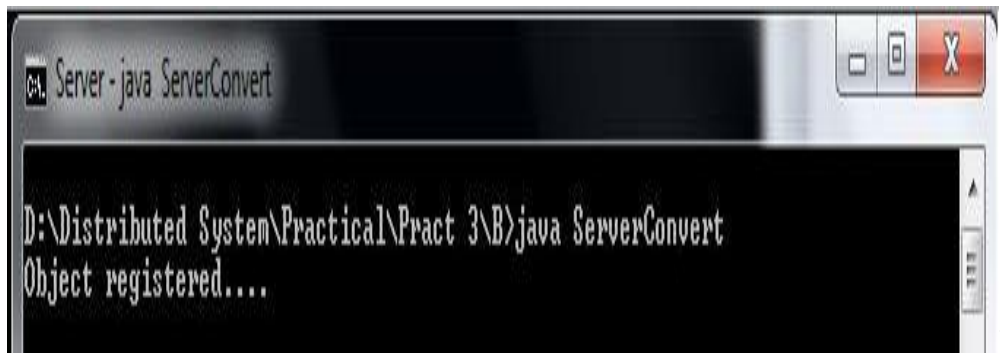
Step 7: Run the RMI Client in a new command prompt, and test the program if it works properly.

➔ java CalcClient

Output:

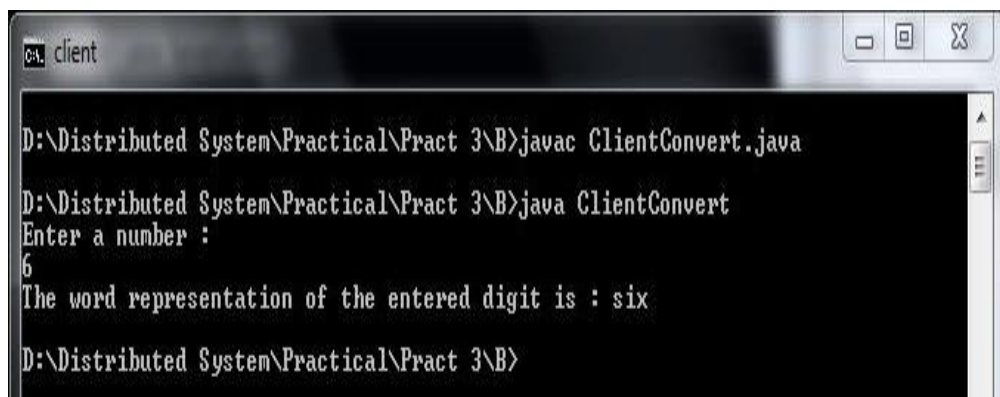
```
c:\ Server - rmiregistry

D:\Distributed System\Practical\Pract 3\B>javac InterConvert.java
D:\Distributed System\Practical\Pract 3\B>javac ServerConvert.java
D:\Distributed System\Practical\Pract 3\B>rmic ServerConvert
D:\Distributed System\Practical\Pract 3\B>rmiregistry
```



```
c:\ Server - java ServerConvert

D:\Distributed System\Practical\Pract 3\B>java ServerConvert
Object registered....
```



```
c:\ client

D:\Distributed System\Practical\Pract 3\B>javac ClientConvert.java
D:\Distributed System\Practical\Pract 3\B>java ClientConvert
Enter a number :
6
The word representation of the entered digit is : six
D:\Distributed System\Practical\Pract 3\B>
```

Exercise:

1. From the previous program, try to check if the server can handle more than one client at any given time.
2. Write a Java RMI Application which enables the client and server communicate by sending text messages. It will be used to enable the server and client implement a simple chat application. They communicate by sending and receiving string values or messages.

Lab 5: Write a program to implement Message Oriented Communication

Objective:

This Lab is a demonstration on how to implement message oriented communication by implementing message passing interface (MPI). We are going to implement some of the primitives defined in message passing interface by using java sockets. We will have two computers communicating, a server and a client and they both will implement the MPI primitives. They can then be able to communicate by using the defined primitives.

- ✓ Understanding the basics concepts of message oriented communication.
- ✓ Understanding the primitives that are used in message passing interface (MPI).
- ✓ Write our own implementation of MPI primitives.

Aim: Write an implementation of message passing interface and enable the communication of a client and server computers using MPI primitives.

Requirements:

We are going to use Java networking and sockets to implement MPI. So, we need a computer that has JDK and some decent java IDE.

Background:

The need to be hardware and platform independent eventually led to the definition of a standard for message passing, simply called the Message-Passing Interface or MPI. MPI is designed for parallel applications and as such is tailored to transient communication. It makes direct use of the underlying network. At the cores of MPI are messaging primitives to support transient communication, of which the most intuitive ones are summarized in figure below.

Primitive Meaning

MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isead	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there are none
MPI_irecv	Check if there is an incoming message, but do not block

Table 1. Some of the most common primitives used in MPI

Primitive description**MPI_bsend:**

- ✓ Transient asynchronous communication is supported by means of the MPI_bsend primitive. The sender submits a message for transmission, which is generally first copied to a local buffer in the MPI runtime system. When the message has been copied, the sender continues. The local MPI runtime system will remove the message from its local buffer and take care of transmission as soon as a receiver has called a receive primitive.

MPI_send:

- ✓ There is also a blocking send operation, called MPI_send, of which the semantics are implementation dependent. The primitive MPI_send may either block the caller until the specified message has been copied to the MPI runtime system at the sender's side, or until the receiver has initiated a receive operation.

MPI_ssend:

- ✓ Synchronous communication by which the sender blocks until its request is accepted for further processing is available through the MPI_ssend primitive.

MPI_sendrecv:

- ✓ Finally, the strongest form of synchronous communication is also supported: when a sender calls MPI_sendrecv, it sends a request to the receiver and blocks until the latter returns a reply. Basically, this primitive corresponds to a normal RPC.

MPI_recv:

- ✓ The operation MPI_recv is called to receive a message; it blocks the caller until a message arrives. There is also an asynchronous variant, called MPI_irecv, by which a receiver indicates that is prepared to accept a message. The receiver can check whether or not a message has indeed arrived, or block until one does.

The semantics of MPI communication primitives are not always straight forward, and different primitives can sometimes be interchanged without affecting the correctness of a program. The official reason why so many different forms of communication are supported is that it gives implementers of MPI systems enough possibilities for optimizing performance. MPI has been designed for high-performance parallel applications, which makes it easier to understand its diversity in different communication primitives.

Task 1: Write a server and client program that implements the following MPI primitives.

1. MPI_send()
2. MPI_recv()
3. MPI_sendrecv()

Procedure 1:

- ✓ Both client and server should provide a user interface (Menu) to enable the user to select which primitive to use.

Procedure 2:

- ✓ They will accept the user selection and call the appropriate MPI primitive implementation.

Implementing the server:

Step 1: Importing the necessary packages:

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

Step 2: Writing the server class:

```
public class Server
{
    PrintStream out=null;
    BufferedReader in=null;
    BufferedReader br=null;
    String str="";
```

Step 3: Writing the server class constructor, everything is done in the constructor including the menu and the method calls:

```
public Server()throws IOException
{
    ServerSocket ss=new ServerSocket(7000);
    System.out.println("Server Started");
    Socket s = ss.accept();
    out = new PrintStream(s.getOutputStream());
    in = new BufferedReader(new InputStreamReader(s.getInputStream()));
    br = new BufferedReader(new InputStreamReader(System.in));
    String str;
    while(s!=null)
    {
        System.out.println("Select the operation to be performed");
```

```
System.out.println("1. To send message without any response
MPI_send");
System.out.println("2. To send message with response
MPI_sendrecv");
System.out.println("3. To receive message MPI_recv");
System.out.println("4. To end the program");
str = br.readLine();
out.println(str);
if(str.equals("1"))
{
MPI_send();
}
else if(str.equals("2"))
{
MPI_sendrecv();
}
else if(str.equals("3"))
{
MPI_recv();
}
else
break;
}
}
```

Program:**EchoServer.java**

```
import java.io.*;
import java.net.*;
public class EchoServer implements Runnable
{
Socket socket=null;
static ServerSocket ss;
EchoServer(Socket newSocket)
{
this.socket=newSocket;
}
public static void main(String args[]) throws IOException
{
ss=new ServerSocket(7000);
System.out.println("Server Started");
while(true)
{
```

```
Socket s = ss.accept();
EchoServer es = new EchoServer(s);
Thread t = new Thread(es);
t.start();
}
}
public void run()
{
try {
while(ss!=null)
{
BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream() ));
PrintStream out = new PrintStream(socket.getOutputStream());
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
String str=in.readLine();
if(str.equals("1"))
{
System.out.println(in.readLine());
}
else if(str.equals("2"))
{
System.out.println(in.readLine());
System.out.println("Enter Data ");
str=br.readLine();
out.println(str);
}
else if(str.equals("3"))
{
System.out.println("Enter Data ");
str=br.readLine();
out.println(str);
}
}
System.out.println("Exiting...");
}
catch(Exception e)
{}
}
}
```

EchoClient.java

```
import java.io.*;
import java.net.*;
public class EchoClient
{
    PrintStream out=null;
    BufferedReader in=null;
    BufferedReader br=null;
    String str="";
    public EchoClient()throws IOException
    {
        Socket s=new Socket("localhost",7000);
        out = new PrintStream(s.getOutputStream());
        in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        br = new BufferedReader(new InputStreamReader(System.in));
        while(s!=null)
        {
            System.out.println("Select the operation to be performed");
            System.out.println("1. To send message without any response");
            System.out.println("2. To send message with response");
            System.out.println("3. To recieve message");
            System.out.println("4. To end the program");
            str=br.readLine();
            out.println(str);
            if(str.equals("1"))
            {
                MPI_send();
            }
            else if(str.equals("2"))
            {
                MPI_sendrec();
            }
            else if(str.equals("3"))
            {
                MPI_recv();
            }
            else
                break;
        }
    }
    public static void main(String args[])throws IOException
    {

```

```
EchoClient ec = new EchoClient();
System.out.println("Exiting...");
}
void MPI_send() throws IOException
{
    System.out.println("Enter Data ");
    str=br.readLine();
    out.println(str);
}
void MPI_sendrec() throws IOException
{
    System.out.println("Enter Data ");
    str=br.readLine();
    out.println(str);
    System.out.println(in.readLine());
}
void MPI_recv() throws IOException
{
    System.out.println(in.readLine());
}
}
```

Exercises:

1. Implement the buffered versions of the above send and receive primitives.

Part III: process synchronization and coordination

Lab 5: Write a program to execute any one mutual exclusion algorithm.

Objective:

In this Lab we will be implementing one of the Mutual Exclusion algorithms. The followings are the primary objectives of this lab session:

- ✓ Understanding the basics and need of Mutual Exclusion.
- ✓ Implement one mutual exclusion algorithm in Java.

Aim: Write a program to execute any one mutual exclusion algorithm.

Background:

Mutual Exclusion Using Token Ring

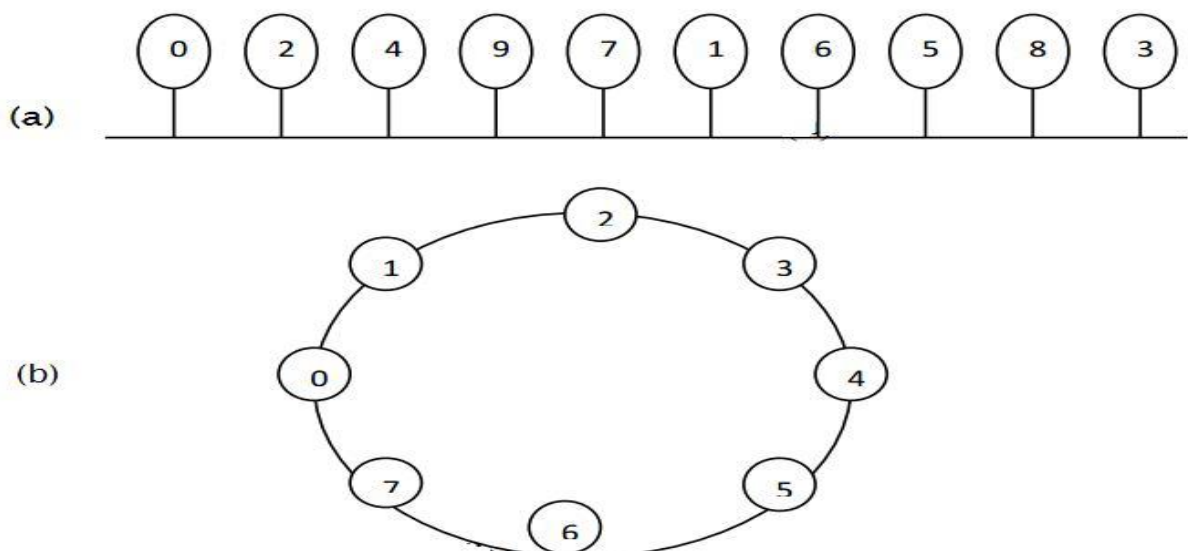
Here is a program to apply mutual exclusion using Token Ring algorithm. The program contains two or more clients and a server. Token is circulated among all client sequentially a client which has a token has right to send message to the server.

Theory:

Systems involving multiple processes are often most easily programmed using critical regions. When a process has to read or update certain shared data structures, it first enters a critical region to achieve mutual exclusion and ensure that no other process will use the shared data structures at the same time.

Token Ring Algorithm

It is a completely different approach to achieving mutual exclusion in a distributing system illustrated in the below diagram.



Here in fig.(a), we have a bus network with no inherent ordering of the processes. In software, a logical ring is constructed in which each process is assigned a position in a ring as shown in fig. (b). The ring positions may be allocated in numerical order of network addresses or some other means. When the ring is initialized, process 0 is given a token. The token circulates around the ring. It is passed from process k to process $k+1$ in point-to-point messages. When a process acquires the token from its neighbour, it checks to see if it is attempting to enter a critical region. If so, the process enters the region, does all the work it needs to, and leaves the region. After it has exited, it passes the token along the ring. It is not permitted to enter a second region using the same token. If a process is handed the token by its neighbour and is not interested in entering a critical region, it just passes it along. As a result, when no processes want to enter any critical regions, the token just circulates at high speed around the ring.

Only one process has the token at any instant, so only one process can actually be in a critical region. Once a process decides it wants to enter a critical region, it worst it will have to wait for every other process to enter and leave one critical region. If the token is ever lost, it must be regenerated. In fact, detecting that it is lost is difficult, since the amount of time between successive appearances of the token on the network is unbounded. The fact that token has not been spotted for an hour does not mean that it has been lost; somebody may still be using it. The algorithm also runs into trouble if a process crashes, but recovery is easier than in other cases. If we require a process receiving the token to acknowledge receipt, a dead process will be detected when its neighbour tries to give it the token and fails. At that point the dead process can be removed from the group, and the token holder can throw the token over the head of the dead process to the next number down the line, or the one that, if necessary. By doing so requires that everyone maintains the current ring configuration.

Task 1: Implement a simple ring based Mutual exclusion algorithm using a single server and two clients.

The clients pass around the token. A client that currently has the token can send a single message to the server.

Step 1: Implementing the server.

Server.java

```
import java.io.*;
import java.net.*;
class Server
```



```
{  
    public static void main(String args[]) throws Exception  
    {  
        ServerSocket ss=new ServerSocket(4722);  
        System.out.println("Server is Started (listen mode) ....");  
        Socket s1 = ss.accept();  
        System.out.println("1st client connected ....");  
        BufferedReader br1 = new BufferedReader(new  
        InputStreamReader(s1.getInputStream()));  
        PrintStream ps1 = new PrintStream(s1.getOutputStream());  
        ps1.println("Hi client... connection established.. Enter commit or abort  
        : ");  
        ps1.flush();  
        Socket s2 = ss.accept();  
        System.out.println("2nd client connected ....");  
        BufferedReader br2 = new BufferedReader(new  
        InputStreamReader(s2.getInputStream()));  
        PrintStream ps2 = new PrintStream(s2.getOutputStream());  
        ps2.println("Hi client... connection established.. Enter commit or abort  
        : ");  
        ps2.flush();  
        String str1 = br1.readLine();  
        String str2 = br2.readLine();  
        if(str1.equalsIgnoreCase("commit") &&  
        str2.equalsIgnoreCase("commit"))  
        {  
            System.out.println("1st client said : " + str1 );  
            System.out.println("2nd client said : " + str2 );  
            System.out.println("Result : doCommit");  
            ps1.println("Result : doCommit");  
            ps1.flush();  
            ps2.println("Result : doCommit");  
            ps2.flush();  
        }  
    }  
}
```

```
        }
    else
    {
        System.out.println("1st client said : " + str1 );
        System.out.println("2nd client said : " + str2 );
        System.out.println("Result : doAbort");
        ps1.println("Result : doAbort");
        ps1.flush();
        ps2.println("Result : doAbort");
        ps2.flush();
    }
}
}
```

Step 2: Implementing the first client.**Cient1.java**

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

class Client1
{
    public static void main(String args[]) throws Exception
    {
        BufferedReader brd = new BufferedReader(new
        InputStreamReader(System.in));
        String serverMsg, clientMsg;
        Socket s=new Socket("localhost",4722);
        BufferedReader br = new BufferedReader(new
        InputStreamReader(s.getInputStream()));
        PrintStream ps = new PrintStream(s.getOutputStream());
        serverMsg = br.readLine();
        System.out.println(serverMsg);
        clientMsg = brd.readLine();
```

```
        ps.println(clientMsg);
        ps.flush();
        serverMsg = br.readLine();
        System.out.println(serverMsg);
    }
}
```

Step 3: Implementing the second client. The two clients are very similar so you can modify the previous client than writing this one from scratch.

Client2.java

```
import java.io.*;
import java.net.*;
class Client2
{
    public static void main(String args[]) throws Exception
    {
        BufferedReader brd = new BufferedReader(new
        InputStreamReader(System.in));
        String serverMsg, clientMsg;
        Socket s=new Socket("localhost",4722);
        BufferedReader br = new BufferedReader(new
        InputStreamReader(s.getInputStream()));
        PrintStream ps = new PrintStream(s.getOutputStream());
        serverMsg = br.readLine();
        System.out.println(serverMsg);
        clientMsg = brd.readLine();
        ps.println(clientMsg);
        ps.flush();
        serverMsg = br.readLine();
        System.out.println(serverMsg);
    }
}
```

Output:

```
Server
D:\Distributed Systems\Practical 5>javac Server.java
D:\Distributed Systems\Practical 5>java Server
Server is Started (listen mode) ....
1st client connected ....
2nd client connected ....
1st client said : abort
2nd client said : commit
Result : doAbort
D:\Distributed Systems\Practical 5>java Server
Server is Started (listen mode) ....
1st client connected ....
2nd client connected ....
1st client said : commit
2nd client said : commit
Result : doCommit
D:\Distributed Systems\Practical 5>_
```

```
Client 1
D:\Distributed Systems\Practical 5>java Client1
Hi client... connection established.. Enter commit or abort :
abort
Result : doAbort
D:\Distributed Systems\Practical 5>java Client1
Hi client... connection established.. Enter commit or abort :
commit
Result : doCommit
D:\Distributed Systems\Practical 5>
```

```
Client2
D:\Distributed Systems\Practical 5>java Client2
Hi client... connection established.. Enter commit or abort :
commit
Result : doAbort
D:\Distributed Systems\Practical 5>java Client2
Hi client... connection established.. Enter commit or abort :
commit
Result : doCommit
D:\Distributed Systems\Practical 5>_
```

Exercise:

1. Modify the above program to support any number of client computers in the system.
2. Implement the centralized and distributed mutual exclusion algorithms.

Lab 6: Implement any one election algorithm**Objective:**

In this Lab we are going to implement an election algorithm in a distributed environment. There will be different computers in the environment and we want to elect a coordinator. The followings are the primary objectives of this lab session:

- ✓ Understanding the basics of election algorithms.
- ✓ Discussion of how elections are held in a distributed environment.
- ✓ Implement the election algorithms.

Aim: Write a program to implement any one election algorithm.

Theory:

An algorithm for choosing a unique process to play a particular role is called an election algorithm.

A ring-based election algorithm:

Each process p_i has a communication channel to the next process in the ring, $p_{(i+1) \bmod N}$ and all messages are sent clockwise around the ring. We assume that no failures occur, and that the system is asynchronous. The goal of this algorithm is to elect a single process called the coordinator, which is the process with the largest identifier.

Initially, every process is marked as a non-participant in an election. Any process can begin an election. It proceeds by marking itself as a participant, placing its identifier in an election message and sending it to its clockwise neighbour. When a process receives an election message, it compares the identifier in the message with its own. If the arrived identifier is greater, then it forwards the message to its neighbour. If the arrived identifier is smaller and the receiver is not a participant, then it substitutes its own identifier in the message and forwards it; but it does not forward the message if it is already a participant. On forwarding an election message in any case, the process marks itself as a participant.

If, however, the received identifier is that of the receiver itself, then this process's identifier must be the greatest, and it becomes the coordinator. The coordinator marks itself as a non-participant once more and sends an elected message to its neighbour, announcing its election and enclosing its identity. When a process p_i receives an elected message, it marks itself as a nonparticipant, sets its variable i to the identifier in the message and, unless it is the new coordinator, forwards the message to its neighbour.

It is easy to see that condition E1 is met. All identifiers are compared, since a process must receive its own identifier back before sending an elected message. For any two processes, the one with the larger identifier will not pass on the other's identifier. It is therefore impossible that both should receive their own identifier back. Condition E2 follows immediately from the guaranteed traversals of the ring (there are no failures).

Note how the non-participant and participant states are used so that duplicate messages arising when two processes start an election at the same time are extinguished as soon as possible, and always before the 'winning' election result has been announced.

If only a single process starts an election, then the worst-performing case is when its anti-clockwise neighbour has the highest identifier. A total of $N - 1$ messages are then required to reach this neighbour, which will not announce its election until its identifier has completed another circuit, taking a further N messages. The elected message is then sent N times, making $3N - 1$ messages in all. The turnaround time is also $3N - 1$, since these messages are sent sequentially.

Bully Algorithm:

The bully algorithm allows processes to crash during an election, although it assumes that message delivery between processes is reliable. Unlike the ring-based algorithm, this algorithm assumes that the system is synchronous: it uses timeouts to detect a process failure. Another difference is that the ring-based algorithm assumed that processes have minimal a priori knowledge of one another: each knows only how to communicate with its neighbour and none knows the identifiers of the other processes. The bully algorithm, on the other hand, assumes that each process knows which processes have higher identifiers, and that it can communicate with all such processes.

There are three types of message in this algorithm: an election message is sent to announce an election; an answer message is sent in response to an election message and a coordinator message is sent to announce the identity of the elected process – the new 'coordinator'. A process begins an election when it notices, through timeouts, that the coordinator has failed. Several processes may discover this concurrently. Since the system is synchronous, we can construct a reliable failure detector. There is a maximum message transmission delay, T_{trans} , and a maximum delay for processing a message $T_{process}$. Therefore, we can calculate a time $T = 2T_{trans} + T_{process}$ that is an upper bound on the time that can elapse between sending a message to another process and receiving a response. If no response arrives within time T , then the local failure detector can report that the intended recipient of the request has failed.

The process that knows it has the highest identifier can elect itself as the coordinator simply by sending a coordinator message to all processes with lower identifiers. On the other hand, a process with a lower identifier can begin an election by sending an election message to those processes that have a higher identifier and awaiting answer messages in response. If none arrives within time T , the process considers itself the coordinator and sends a coordinator message to all processes with lower identifiers announcing this. Otherwise, the process waits a further period T' for a coordinator message to arrive from the new coordinator. If none arrives, it begins another election. If a process p_i receives a coordinator message, it sets its variable $elect_i$ to the identifier of the coordinator contained within it and treats that process as the coordinator. If a process receives an election message, it sends back an answer message and begins another election – unless it has begun one already.

When a process is started to replace a crashed process, it begins an election. If it has the highest process identifier, then it will decide that it is the coordinator and announce this to the other processes. Thus it will become the coordinator, even though the current coordinator is functioning. It is for this reason that the algorithm is called the ‘bully’ algorithm.

Task 1: Write a program to implement any one election algorithm

Election.c

```
#include<stdio.h>
#include<conio.h>
#include<process.h>

Struct proc
{
    int live;
    int identifier;
}

process[10];
intn,cordinator=1;

/***** DISPLAY PROCESSES *****/

void display()
{
    int i;
    printf("\n PROCESSES ARE\n\n");
```

```
        printf("Processes ");
        for(i=1;i<=n;i++)
        {
            printf("P%d\t",i);
        }
        printf("\nlive ");

        for(i=1;i<=n;i++)
        {
            printf("%d\t",process[i].live);
        }
        printf("\nidentifier ");
        for(i=1;i<=n;i++)
        {
            printf("%d\t",process[i].identifier);
        }

        /***** BULLY ALGORITHM *****/
void bully()
{
    int ch,c,id,i=0,cordinator,init,max=-99;
    cordinator=i;
    for(i=1;i<=n;i++)
    {

        if(process[cordinator].identifier<process[i].identifier&&process[i].live==1)
            cordinator=i;
    }
    printf("\n\n CURRENT CO-ORDINATOR IS=P%d",cordinator);
    while(ch!=4)
    {
        printf("\n\n *** BULLY ALGORITHM ***");
        printf("\n1.Crash a Process\n2.Activate Process\n3.Display\n4.Exit");
        printf("\nENTER UR CHOICE");
```



```
scanf("%d",&ch);
switch(ch)
{
    case 1:
        printf("\n Enter the process id to crash");
        scanf("%d",&id);
        if(process[id].live==0)
        {
            printf("\n Already crashed process");
        }
    else
    {
        process[id].live=0;
        printf("\n process P%d is crashed",id);
        if(id==cordinator)
        {
            while(1)
            {
                printf("\n Enter process id who intiates election");
                scanf("%d",&init);
                if(process[init].live==0)
                {
                    printf("\n the selected process is crashed");
                }
            }
        }
        else
        {
            for(i=1;i<=n;i++)
            {
                if(i!=init&&&
                rocess[i].identifier>process[init].identifier)
                printf("\n Election MSG sent from %d to %d",init,i);
            }
            for(i=1;i<=n;i++)
```

```
        {
            if(i!=init)
            {

if(process[i].identifier>process[init].identifier&&process[i].live!=0)

                {

                    printf("\n OK from %d to %d",i,init);

                }

            }

        }
for(i=1;i<=n;i++)
{
    if(max<process[i].identifier && process[i].live!=0)
    {
        cordinator=i;
        max=process[i].identifier;
    }
}
printf("\n\n NEW CO-ORDINATOR
IS=P%d",cordinator);
break;
    }
}
}
break;
case 2:
    printf("\n Enter process id to activate");
    scanf("%d",&id);
    if(process[id].live==1)
    {

        printf("\n Process %d is already active",id);

    }
```

```
        else
        {
            process[id].live=1;
            printf("\n Process %d activated",id);
        }
        if(process[id].identifier>process[cordinator].identifier)
        {
            cordinator=id;
            printf("\n NEW CO-ORDINATOR IS=P%d\n\n",id);
        }
        break;
    case 3:
        display();
        break;
    case 4:
        break;
    }
}

}

/***** RING ALGORITHM *****/
void ring()
{
    int ch,c,id,i=0,init,max=-99,last;
    for(i=1;i<=n;i++)
    {
        if(process[cordinator].identifier<process[i].identifier&&process[i].live==1)
            cordinator=i;
    }
    printf("\n\n CURRENT CO-ORDINATOR IS=P%d",cordinator);
    while(ch!=4)
    {
        printf("\n\n\n *** RING ALGORITHM ***");
        printf("\n1.Crash a Process\n2.Activate Process\n3.Display\n4.Exit");
    }
}
```

```
printf("\nENTER UR CHOICE");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\n Enter the process id to crash");
scanf("%d",&id);
if(process[id].live==0)
{
printf("\n Already crashed process");
}
else
{
process[id].live=0;
printf("\n process P%d is crashed",id);
if(id==cordinator)
{
while(1)
{
printf("\n Enter process id who intiates
election");
scanf("%d",&init);
if(process[init].live==0)
{
printf("\n the selected process is
crashed");
}
else
{
last=init;
printf("\nElection MSG sent from
=%d",last);
for(i=init+1;i<=n;i++)
```

```
        {
            if(i!=init)
                printf("->%d",i);
        }
        for(i=1;i<init;i++)
        {
            if(i!=init)
                printf("->%d",i);
            last=i;
        }
        for(i=init+1;i<=n;i++)
        {
            if(max<process[i].identifier && process[i].live==1)
            {
                cordinator=i;
                max=process[i].identifier;
            }
        }
        for(i=1;i<=init;i++)
        {
            if(max<process[i].identifier && process[i].live==1)
            {
                cordinator=i; max=process[i].identifier;
            }
        }
        printf("\n\n NEW CO-ORDINATOR
        IS=P%d",cordinator);
        break;
    }
}
}
break;
```

```
        case 2:
            printf("\n Enter process id to activate");
            scanf("%d",&id);
            if(process[id].live==1)
            {
                printf("\n Process %d is already active",id);
            }
            else
            {
                process[id].live=1;
                printf("\n Process %d activated",id);
                if(process[id].identifier>process[cordinator].identifier)
                {
                    printf("\n NEW CO-ORDINATOR IS=P%d\n\n",id);
                    cordinator=id;
                }
            }
            break;
        case 3:
            display();
            break;
        case 4:
            break;
    }
}

void main()
{
    intch,i,c;
    clrscr();
    printf("\n ENTER NO. OF PROCESSES");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
```

```
{
    printf("\nEnter P%d process live or not(0/1)",i);
    scanf("%d",&process[i].live);
    printf("\nEnter P%d process identifier",i);
    scanf("%d",&process[i].identifier);
}
display();
while(1)
{
    printf("\n\n\n**** ELECTION ALGORITHM ****");
    printf("\n1.BULLY ALGORITHM\n2.RING
ALGORITHM\n3.EXIT");
    printf("\n\n ENTER UR CHOICE");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
            bully();
            break;
        case 2:
            ring();
            break;
        case 3:
            exit(0);
    }
}
}
```

Output:

```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

ENTER NO. OF PROCESSES
5
EnterP1 process live or not(0/1)1
EnterP1 process identifier1
EnterP2 process live or not(0/1)1
EnterP2 process identifier2
EnterP3 process live or not(0/1)0
EnterP3 process identifier3
EnterP4 process live or not(0/1)1
EnterP4 process identifier4
EnterP5 process live or not(0/1)1
EnterP5 process identifier5
```

Creating Processes

```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

PROCESSES ARE

Processes P1    P2    P3    P4    P5
live 1 1        0     1     1     5
identifier 1    2     3     4     5
```

List of Processes

```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

**** ELECTION ALGORITHM ****
1.BULLY ALGORITHM
2.RING ALGORITHM
3.EXIT

ENTER UR CHOICE:
```

Algorithm Selection

1. Bully Algorithm

```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

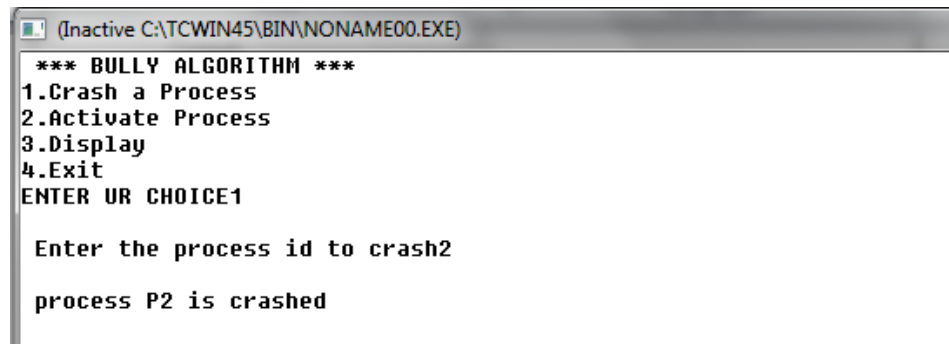
CURRENT CO-ORDINATOR IS=P5
```

Displaying Current Coordinator

```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

*** BULLY ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE
```

Operations of Bully Algorithm



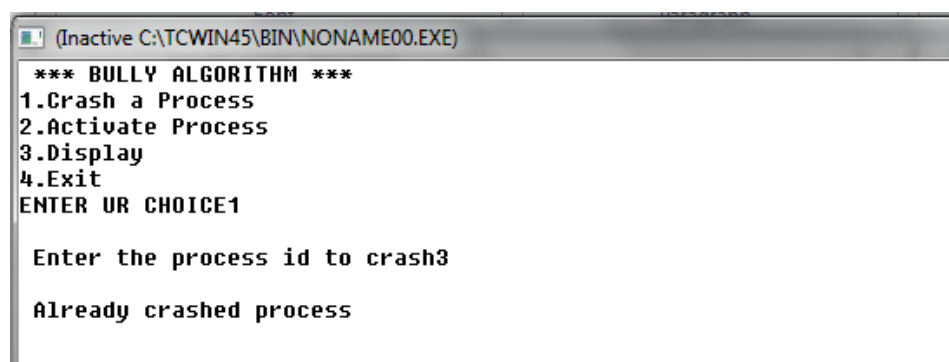
```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

*** BULLY ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE1

Enter the process id to crash2

process P2 is crashed
```

Crashing an Active Process



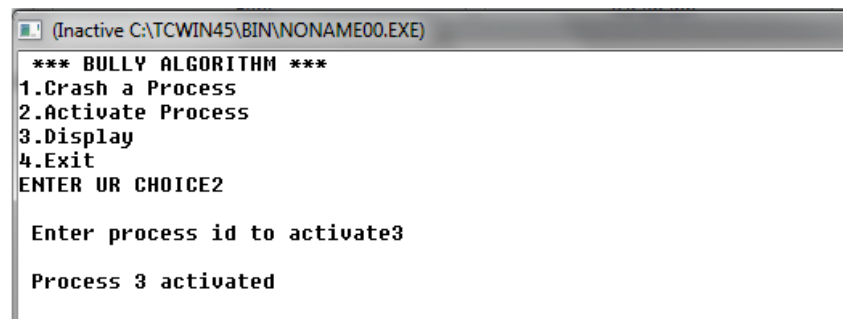
```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

*** BULLY ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE1

Enter the process id to crash3

Already crashed process
```

Crashing a Crashed Process



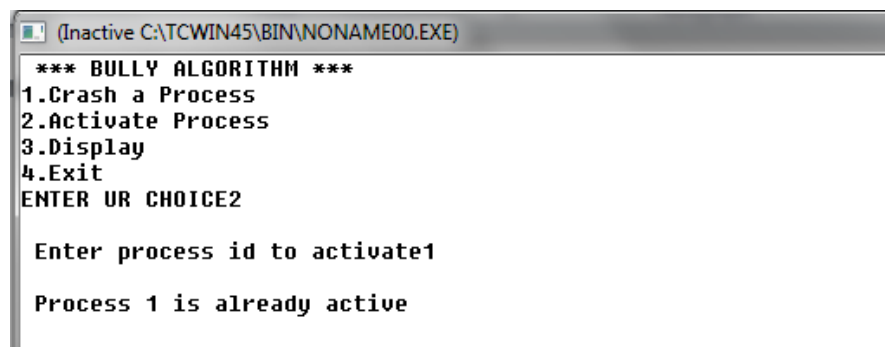
```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

*** BULLY ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE2

Enter process id to activate3

Process 3 activated
```

Activating a Crashed Process



```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

*** BULLY ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE2

Enter process id to activate1

Process 1 is already active
```

Activating Already Active Process

```

(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

*** BULLY ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE1

Enter the process id to crash5

process P5 is crashed
Enter process id who initiates election3

Election MSG sent from 3 to 4
Election MSG sent from 3 to 5
OK from 4 to 3

NEW CO-ORDINATOR IS=P4

```

Crashing the Co-ordinator and Election

```

(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

*** BULLY ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE3

PROCESSES ARE

Processes P1    P2    P3    P4    P5
live 1  0      1      1      0
identifier 1    2      3      4      5

```

Displaying Process Status

```

(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

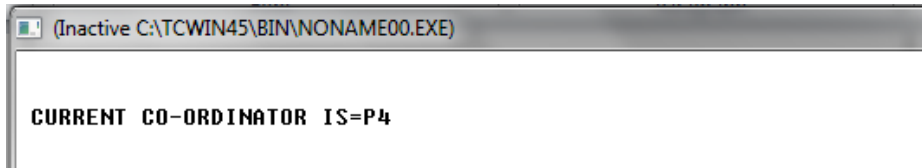
*** BULLY ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE4

**** ELECTION ALGORITHM ****
1.BULLY ALGORITHM
2.RING ALGORITHM
3.EXIT

```

Exiting Bully Algorithm and Entering Main Menu

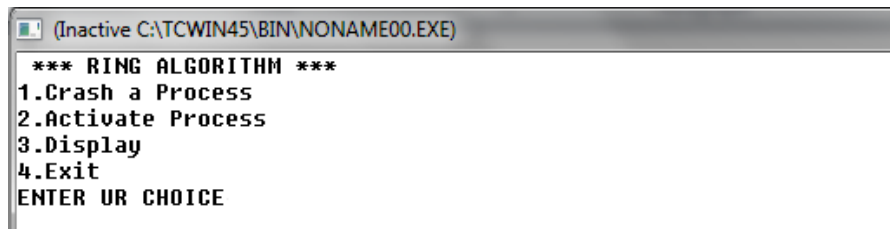
2. Ring Algorithm



```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

CURRENT CO-ORDINATOR IS=P4
```

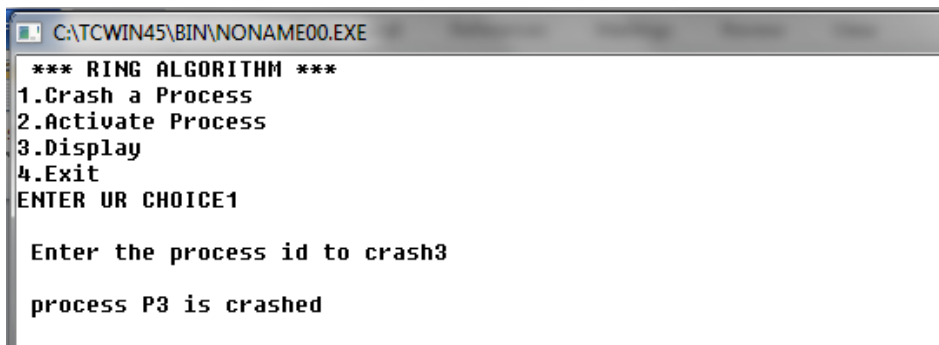
Displaying Co-ordinator



```
(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

*** RING ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE
```

Operations of Algorithm



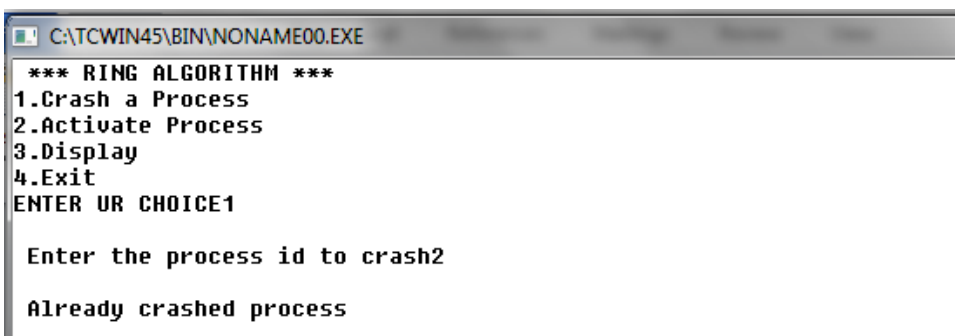
```
C:\TCWIN45\BIN\NONAME00.EXE

*** RING ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE1

Enter the process id to crash3

process P3 is crashed
```

Crashing an Active Process



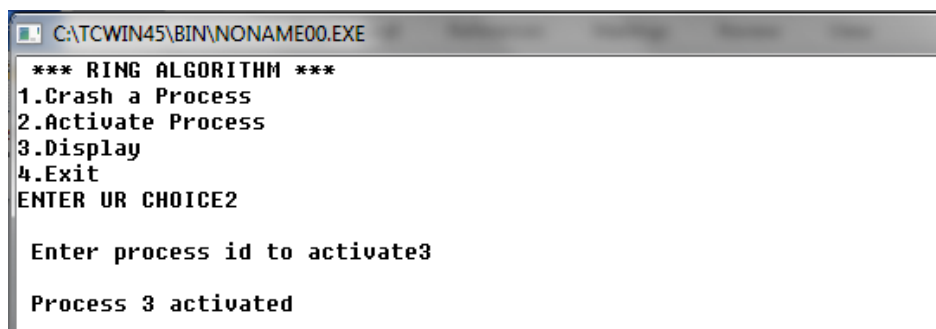
```
C:\TCWIN45\BIN\NONAME00.EXE

*** RING ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE1

Enter the process id to crash2

Already crashed process
```

Crashing an Already Crashed Process



```
C:\TCWIN45\BIN\NONAME00.EXE

*** RING ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE2

Enter process id to activate3

Process 3 activated
```

Activating a Crashed Process

```

C:\TCWIN45\BIN\NONAME00.EXE
*** RING ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE2

Enter process id to activate1

Process 1 is already active

```

Activating already active Process

```

C:\TCWIN45\BIN\NONAME00.EXE
*** RING ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE1

Enter the process id to crash4

process P4 is crashed
Enter process id who initiates election3

Election MSG sent from =3 ->4 ->5->1->2

NEW CO-ORDINATOR IS=P3

```

Crashing the Co-ordinator and Election

```

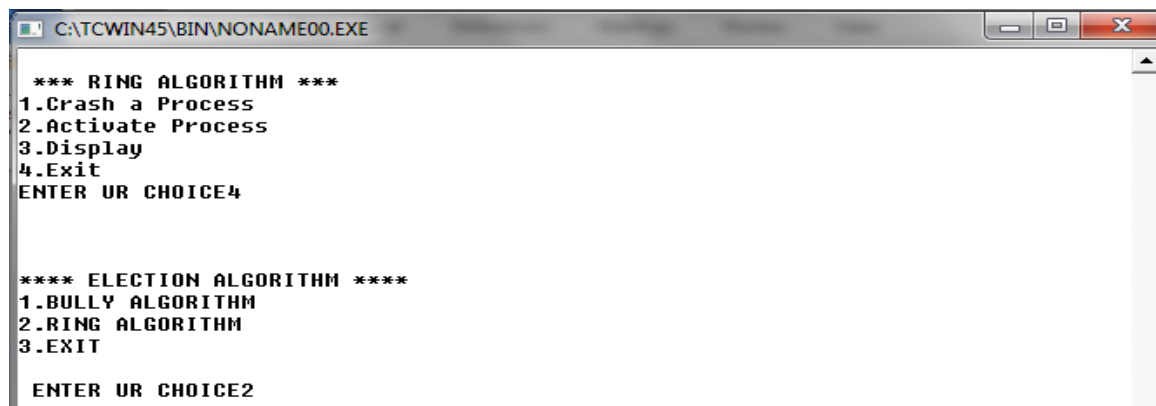
C:\TCWIN45\BIN\NONAME00.EXE
*** RING ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE3

PROCESSES ARE

Processes P1    P2    P3    P4    P5
live 1  0      1      0      0
identifier 1    2      3      4      5

```

Displaying Process Status



```
C:\TCWIN45\BIN\NONAME00.EXE

*** RING ALGORITHM ***
1.Crash a Process
2.Activate Process
3.Display
4.Exit
ENTER UR CHOICE4

**** ELECTION ALGORITHM ****
1.BULLY ALGORITHM
2.RING ALGORITHM
3.EXIT
ENTER UR CHOICE2
```

Exiting Ring Algorithm and Entering Main Menu

Exercise:

1. Implement the ring based election algorithm. Consider the previous lab's implementation of ring based mutual exclusion and modify it to perform an election algorithm.
2. Implement the bully election algorithm.

Lab 7: Implementation of clock synchronization algorithms.**Task 1: Write a program for implementation of clock synchronization algorithms.****SCServer.java**

```
import java.io.*;
import java.net.*;
import java.sql.*;
public class SCServer
{
    public static void main(String args[])throws Exception
    {
        InetAddress lclhost;
        lclhost=InetAddress.getLocalHost();
        long maxtime,skewtime,datatime;
        String maxtimestr,skewtimestr;
        BufferedReader br;
        CIntServer ser=new CIntServer(lclhost);
        System.out.println("Enter the maximum time");
        br = new BufferedReader(new InputStreamReader(System.in));
        maxtimestr=br.readLine();
        System.out.println("Enter the maximum skew time");
        br = new BufferedReader(new InputStreamReader(System.in));
        skewtimestr=br.readLine();
        maxtime=Long.parseLong(maxtimestr);
        skewtime=Long.parseLong(skewtimestr);
        while(true)
        {
            datatime = System.currentTimeMillis();
            long G = datatime-maxtime-skewtime;
            System.out.println("G =" +G);
            ser.setTimeStamp(new Timestamp(G));
            ser.recPort(8001);
            ser.recData();
        }
    }
}

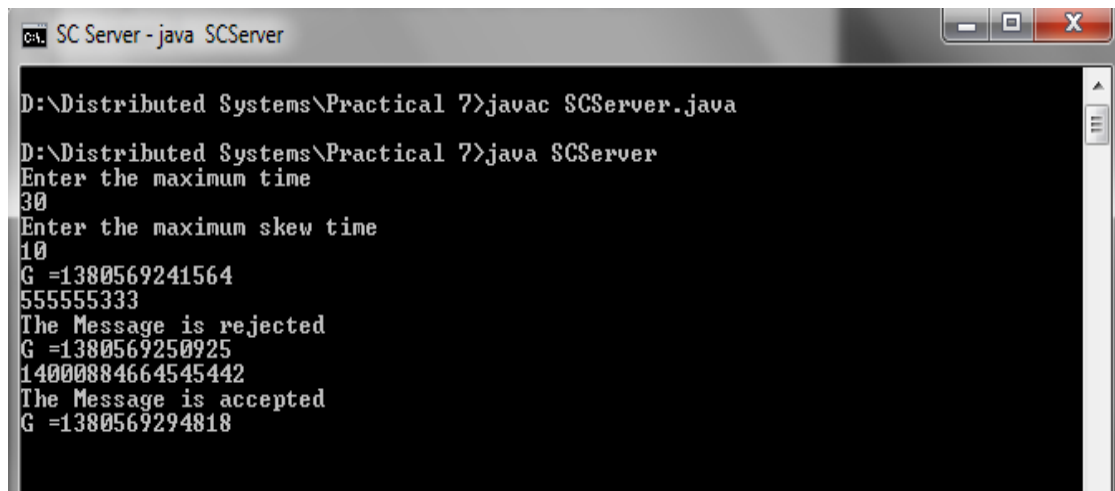
class CIntServer
{
    InetAddress lclhost;
    int recport;
```

```
Timestamp obtmp;
ClnServer(InetAddress lclhost)
{
    this.lclhost = lclhost;
}
void recPort(int recport)
{
    this.recport = recport;
}
void setTimeStamp(Timestamp obtmp)
{
    this.obtmp = obtmp;
}
void recData()throws Exception
{
    String msgstr="";
    DatagramSocket ds;
    DatagramPacket dp;
    BufferedReader br;
    byte buf[] = new byte[256];
    ds = new DatagramSocket(recport);
    dp = new DatagramPacket(buf,buf.length);
    ds.receive(dp);
    ds.close();
    msgstr = new String(dp.getData(),0,dp.getLength());
    System.out.println(msgstr);
    Timestamp obtmp = new Timestamp(Long.parseLong(msgstr));
    if(this.obtmp.before(obtmp) == true)
    {
        System.out.println("The Message is accepted");
    }
    else
    {
        System.out.println("The Message is rejected");
    }
}
}
```

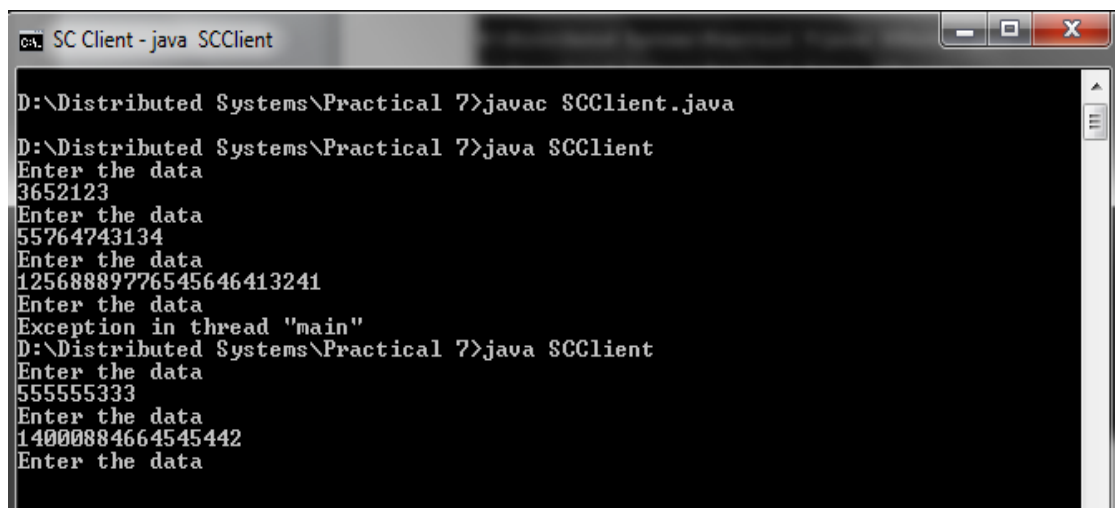
SCClient.java

```
import java.io.*;
import java.net.*;
public class SCClient
```

```
{
    public static void main(String args[])throws Exception
    {
        InetAddress lclhost;
        lclhost=InetAddress.getLocalHost();
        while(true)
        {
            Client cntl=new Client(lclhost);
            cntl.sendPort(9001);
            cntl.sendData();
        }
    }
}
class Client
{
    InetAddress lclhost;
    int senport;
    Client(InetAddress lclhost)
    {
        this.lclhost=lclhost;
    }
    void sendPort(int senport)
    {
        this.senport=senport;
    }
    void sendData()throws Exception
    {
        DatagramPacket dp;
        DatagramSocket ds;
        BufferedReader br;
        br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the data");
        String str=br.readLine();
        ds = new DatagramSocket(senport);
        dp = new
        DatagramPacket(str.getBytes(),str.length(),lclhost,senport-1000);
        ds.send(dp);
        ds.close();
    }
}
```


Output:

```
SC Server - java SCSERVER
D:\Distributed Systems\Practical 7>javac SCServer.java
D:\Distributed Systems\Practical 7>java SCServer
Enter the maximum time
30
Enter the maximum skew time
10
G =1380569241564
555555333
The Message is rejected
G =1380569250925
14000884664545442
The Message is accepted
G =1380569294818
```

Server Window

```
SC Client - java SCClient
D:\Distributed Systems\Practical 7>javac SCClient.java
D:\Distributed Systems\Practical 7>java SCClient
Enter the data
3652123
Enter the data
55764743134
Enter the data
12568889776545646413241
Enter the data
Exception in thread "main"
D:\Distributed Systems\Practical 7>java SCClient
Enter the data
555555333
Enter the data
14000884664545442
Enter the data
```

Client Window

Lab 8: Implementation of two phases commit protocol.**Task 1: Write a program to implement two phase commit protocol.****Server.java**

```
import java.io.*;
import java.net.*;
import java.util.*;
public class Server
{
    boolean closed=false,inputFromAll=false;
    List<clientThread> t;
    List<String> data;
    Server()
    {
        t = new ArrayList<clientThread>();
        data= new ArrayList<String>();
    }
    public static void main(String args[])
    {
        Socket clientSocket = null;
        ServerSocket serverSocket = null;
        int port_number=1111;
        Server ser=new Server();
        try
        {
            serverSocket = new ServerSocket(port_number);
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
    }
}
```

```
while(!ser.closed)
{
    try
    {
        clientSocket = serverSocket.accept();
        clientThread th=new clientThread(ser,clientSocket);
        (ser.t).add(th);
        System.out.println("\nNow Total clients are : "+(ser.t).size());
        (ser.data).add("NOT_SENT");
        th.start();
    }
    catch (IOException e)
    {
    }
}
try
{
    serverSocket.close();
}
catch(Exception e1)
{
}
}
}
class clientThread extends Thread
{
    DataInputStream is = null;
    String line;
    String destClient="";
    String name;
    PrintStream os = null;
    Socket clientSocket = null;
    String clientIdentity;
```

```
Server ser;
public clientThread(Server ser,Socket clientSocket)
{
    this.clientSocket=clientSocket;
    this.ser=ser;
}
public void run()
{
    try
    {
        is = new DataInputStream(clientSocket.getInputStream());
        os = new PrintStream(clientSocket.getOutputStream());
        os.println("Enter your name.");
        name = is.readLine();
        clientIdentity=name;
        os.println("Welcome "+name+" to this 2 Phase
Application.\nYou will receive a vote Request now...");
        os.println("VOTE_REQUEST\nPlease enter COMMIT or
ABORT to proceed : ");
        for(int i=0; i<(ser.t).size(); i++)
        {
            if((ser.t).get(i)!=this)
            {
                ((ser.t).get(i)).os.println("---A new user
"+name+"
entered the Appilcation---");
            }
        }
        while (true)
        {
            line = is.readLine();
            if(line.equalsIgnoreCase("ABORT"))
            {
```

```
        System.out.println("\nFrom '"+clientIdentity+"' :  
        ABORT\n\nSince aborted we will not wait for inputs  
        from other clients.");  
        System.out.println("\nAborted....");  
        for(int i=0; i<(ser.t).size(); i++)  
        {  
            ((ser.t).get(i)).os.println("GLOBAL_ABORT");  
            ((ser.t).get(i)).os.close();  
            ((ser.t).get(i)).is.close();  
        }  
        break;  
    }  
    if(line.equalsIgnoreCase("COMMIT"))  
    {  
        System.out.println("\nFrom '"+clientIdentity+"' :COMMIT");  
        if((ser.t).contains(this))  
        {  
            (ser.data).set((ser.t).indexOf(this), "COMMIT");  
            for(int j=0;j<(ser.data).size();j++)  
            {  
                if(!(((ser.data).get(j)).equalsIgnoreCase("NOT SENT")))  
                {  
                    ser.inputFromAll=true;  
                    continue;  
                }  
                else  
                {  
                    ser.inputFromAll=false;  
                    System.out.println("\nWaiting for inputs from other clients.");  
                    break;  
                }  
            }  
        }  
        if(ser.inputFromAll)
```

```
        {
            System.out.println("\n\nCommitted....");
            for(int i=0; i<(ser.t).size(); i++)
            {
                ((ser.t).get(i)).os.println("GLOBAL_COMMIT");
                ((ser.t).get(i)).os.close();
                ((ser.t).get(i)).is.close();
            }
            break;
        }
    } //if t.contains
} //commit
} //while
ser.closed=true;
clientSocket.close();
}
catch(IOException e)
{
};
}
}
```

Client.java

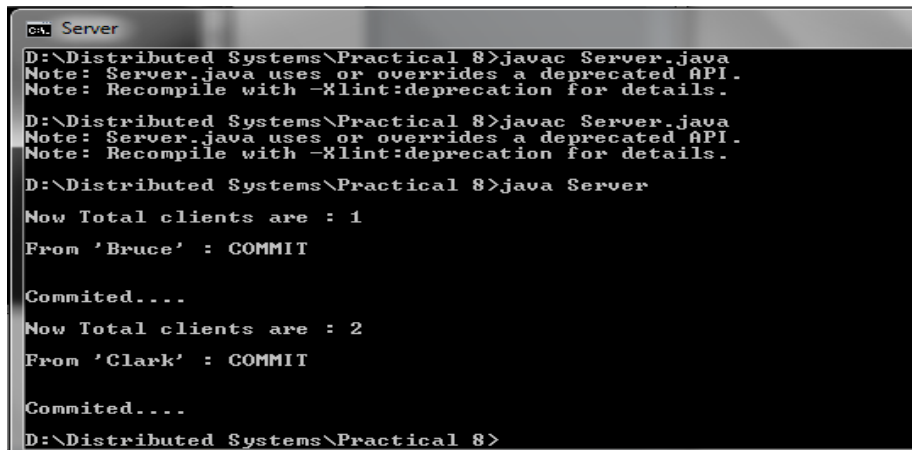
```
import java.io.*;
import java.net.*;
public class Client implements Runnable
{
    static Socket clientSocket = null;
    static PrintStream os = null;
    static DataInputStream is = null;
    static BufferedReader inputLine = null;
    static boolean closed = false;
    public static void main(String[] args)
    {
        int port_number=1111;
        String host="localhost";
        try
```

```
        {
            clientSocket = new Socket(host, port_number);
            inputLine = new BufferedReader(new
            InputStreamReader(System.in));
            os = new PrintStream(clientSocket.getOutputStream());
            is = new DataInputStream(clientSocket.getInputStream());
        }
        catch (Exception e)
        {
            System.out.println("Exception occurred : "+e.getMessage());
        }
        if (clientSocket != null && os != null && is != null)
        {
            try
            {
                new Thread(new Client()).start();
                while (!closed)
                {
                    os.println(inputLine.readLine());
                }
                os.close();
                is.close();
                clientSocket.close();
            }
            catch (IOException e)
            {
                System.err.println("IOException: " + e);
            }
        }
    }
}

public void run()
{
    String responseLine;
    try
    {
        while ((responseLine = is.readLine()) != null)
        {
            System.out.println("\n"+responseLine);
            if (responseLine.equalsIgnoreCase("GLOBAL_COMMIT")==true ||
            responseLine.equalsIgnoreCase("GLOBAL_ABORT")==true )
            {
                break;
            }
        }
    }
}
```

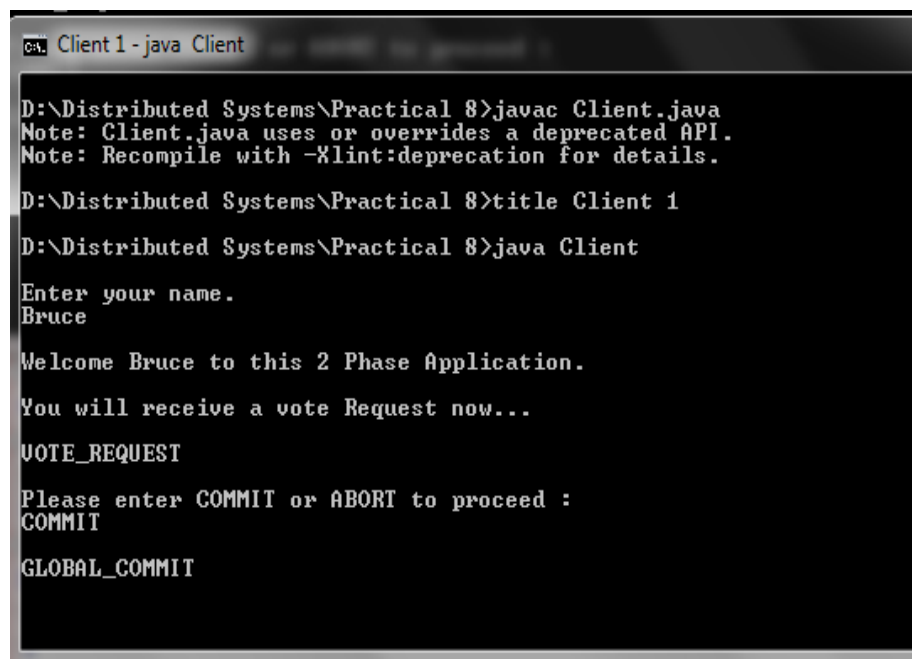
```
        }  
    }  
    closed=true;  
}  
catch (IOException e)  
{  
    System.err.println("IOException: " + e);  
}  
}  
}
```

Output:



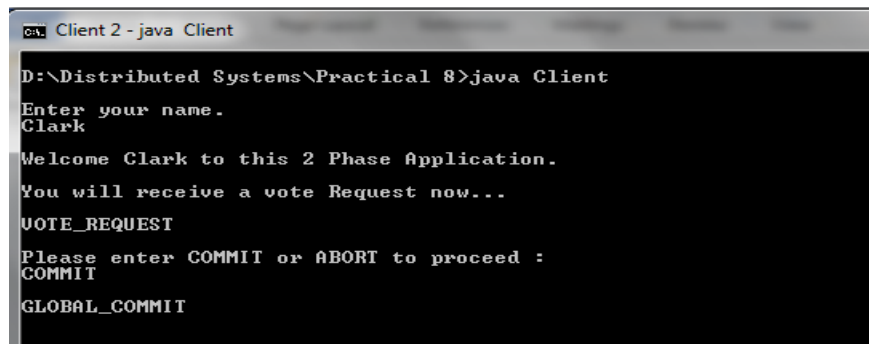
```
C:\ Server  
D:\Distributed Systems\Practical 8>javac Server.java  
Note: Server.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
D:\Distributed Systems\Practical 8>javac Server.java  
Note: Server.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
D:\Distributed Systems\Practical 8>java Server  
Now Total clients are : 1  
From 'Bruce' : COMMIT  
  
Committed....  
Now Total clients are : 2  
From 'Clark' : COMMIT  
  
Committed....  
D:\Distributed Systems\Practical 8>
```

Server Window



```
C:\ Client 1 - java Client  
D:\Distributed Systems\Practical 8>javac Client.java  
Note: Client.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
D:\Distributed Systems\Practical 8>title Client 1  
D:\Distributed Systems\Practical 8>java Client  
Enter your name.  
Bruce  
  
Welcome Bruce to this 2 Phase Application.  
You will receive a vote Request now...  
VOTE_REQUEST  
  
Please enter COMMIT or ABORT to proceed :  
COMMIT  
  
GLOBAL_COMMIT
```

Client Window 1



```
C:\ Client 2 - java Client

D:\Distributed Systems\Practical 8>java Client
Enter your name.
Clark
Welcome Clark to this 2 Phase Application.
You will receive a vote Request now...
VOTE_REQUEST
Please enter COMMIT or ABORT to proceed :
COMMIT
GLOBAL_COMMIT
```

Client Window 2

The end!