

# Smt.Chandabai Himathmal Mansukhani

USCS 3P01 :USCS303-Operating System(OS) Practical-04

DATE: 07-08-2021

## Process Communication

### Contents

USCS3P01:USCS303-Operating System (OS)

Aim.....	2
Process Communication.....	4
Producer-Consumer Problem.....	5
Using Shared Memory.....	6
Using Message Passing.....	11
Remote Procedure Calls.....	14

# Smt.Chandabai Himathmal Mansukhani

A) Aim

**Producer consumer problem solution using  
message passing**

- **Message passing is the basis of most interprocess communication in distributed systemS.**
- **City Centre lowest level of abstraction and requires the application programmer to be able to identify the destination process, the message , the source process and the data types expected from this processor.**
- **Communication in the message passing array paradigm , in its simplest form ,is performed using the send and receive primitives the syntax is generally of the form: send(receiver,message) receive (sender ,message)**
- **The Send() primitive requires the name of the destination process and the message data as parameters. the addition of the name of the sender as parameter for the send() primitive would enable the receiver to acknowledge the message. The receive() primitive requires the name of the anticipated centre and should provide storage buffer for the message.**

# Smt.Chandabai Himathmal Mansukhani

## Remote method invocation

### Remote Procedure calls

- Message passing leaves the programmer with the burden of the explain control of the movement of data. Remote Procedure calls ( RPC) relieves this burden by increasing the level of abstraction and providing semantics similar to a local procedure call.
- Syntax :
- The syntax of a remote procedure call is generally of the form : call procedure\_name (value\_arguments; result\_arguments)
- The client process block at the call() until the reply is received.
- The remote procedure is the sensor processes which has already begun executing on a remote machine .
- It blocks at the receive() until it receives a message and parameters from the sender.
- The server then sends a reply where it has finished Its task.
- The syntax is as follows receive procedure\_name( in value\_parameters; out result parameters) . reply( caller, result\_parameters)
- In the simplest case the execution of the call() generate a client stub which marshals the arguments into a message and sends the message to the server machine. On the server machine the server is blocked awaiting the message. On receipt of the message the service stub is generated and extract the parameters from the message and passes the parameters and control to the procedure. The results are returned to the client with the same procedure in reverse.

# Smt.Chandabai Himathmal Mansukhani

## c) Process communication

Processes often need to communicate with each other .

This is complicated in distributed systems by the fact that the communicating processes may be on different workstations.

Inter-process communication provides a means for processes to cooperate and complete.

Message passing and remote procedure calls are the most common methods of inter-process communication in distributed systems.

A less frequently used but no less valuable method is distributed shared memory.

Producer-consumer

Producer-consumer problem

In a producer/consumer relationship, the producer portion of an application generates data and stores it in a shared object, and the consumer portion of an application reads data from the shared object.

One example of a common producer/consumer relationship is a print spooling. A wordprocessor spools data from a buffer (typically a file) and that data is subsequently consumed by the printer as it prints the document . Similarly an application that copies data onto compact discs places data in a fixed-size buffer that is emptied as the CD-RW drive burns the data onto the compact disc.

# Smt.Chandabai Himathmal Mansukhani

d) **Producer consumer problem solution**  
**using message passing**

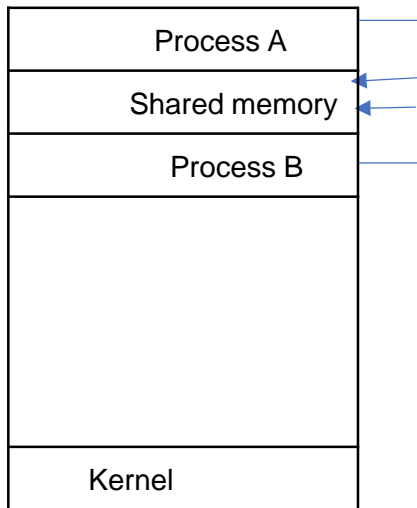
- **Message passing is the basis of most interprocess communication in distributed systemS.**
- **City Centre lowest level of abstraction and requires the application programmer to be able to identify the destination process, the message , the source process and the data types expected from this processor.**
- **Communication in the message passing array paradigm , in its simplest form ,is performed using the send and receive primitives the syntax is generally of the form: send(receiver,message) receive (sender ,message)**
- **The Send() primitive requires the name of the destination process and the message data as parameters. the addition of the name of the sender as parameter for the send() primitive would enable the receiver to acknowledge the message. The receive() primitive requires the name of the anticipated centre and should provide storage buffer for the message.**

# Smt.Chandabai Himathmal Mansukhani

## e) USING SHARED MEMORY

Shared memory is a memory that may be simultaneously accessed by multiple processes with an intent to provide communication among them r avoid redundant copies .

Shared memory is an efficient means of passing data between processes.



# Smt.Chandabai Himathmal Mansukhani

Question - 01

Write a Java Program for producer-consumer problem using shared memory memory .

File 1: **P4\_PC\_SM\_Buffer\_RS**

//Name: SHIVHARI CHAVAN

//Batch: B2

//PRN: 2018016401247454

//Date: 13/8/2021

//Prac-04: Process Communication.

```
public interface P4_PC_SM_Buffer_RS
{
    // producer call this method
    public void insert (String item);
    // consumers call this method
    public String remove();
} // interface ends
```

File 2: **P4\_PC\_SM\_BufferImpl\_RS**

//Name: SHIVHARI CHAVAN

//Batch:B2

//PRN:2018016401247454

//Date:13/8/2021

//Prac-04:Process Communication.

```
public class P4_PC_SM_BufferImpl_RS implements P4_PC_SM_Buffer_RS
{
    private static final int BUFFER_SIZE= 5;
    private String[] elements;
    private int in, out, count;
    public P4_PC_SM_BufferImpl_RS() //constructor initialising the variables to initial
    value
    {
```

# Smt.Chandabai Himathmal Mansukhani

```
count = 0;
in = 0;
out = 0;
elements= new String[BUFFER_SIZE];
} // Constructor ends
// producers call this method
public void insert(String item)
{
    while (count== BUFFER_SIZE)
    ; // do nothing as there is no free space
    // add an item to the buffer
    elements[in] =item;
    in = (in+ 1) % BUFFER_SIZE;
    ++count;
    System.out.println ("item produced"+ item + "at position" + in + "having total items
"+ count);
} // insert() ends
//Consumers call this method
public String remove()
{
    String item;
    while( count ==0)
    ; // do nothing as there is nothing to consume
    // remove an item from the buffer
    item= elements[out];
    out= (out + 1) % BUFFER_SIZE; --count;
    System.out.println(" item consumed:" + item + "from position"+ out + "remaining
total items" +count);
    return item;
} // remove ends
} // class ends
```



# Smt.Chandabai Himathmal Mansukhani

File 3: P4\_PC\_SM\_RS

//Name: SHIVHARI CHAVAN

//Batch:B2

//PRN: 2018016401247454

//Date:13/8/2021

//Prac-04:Process Communication.

```
public class P4_PC_SM_RS{

public static void main(String[] args) {

P4_PC_SM_BufferImpl_RS bufobj = new P4_PC_SM_BufferImpl_RS();

System.out.println("\n=====PRODUCER producing the ITEMS =====\n");

bufobj.insert("Name : SHIVHARI CHAVAN");

bufobj.insert("CHMCS : Batch - B1");

bufobj.insert("PRN :2018016401247454 ");

bufobj.insert("USCSP301 - USCS303 : OS Practical -P4");

System.out.println("\n=====CONSUMER consuming the ITEMS =====\n");

String element = bufobj.remove();
```

# Smt.Chandabai Himathmal Mansukhani

```
System.out.println(element);

System.out.println(bufobj.remove());

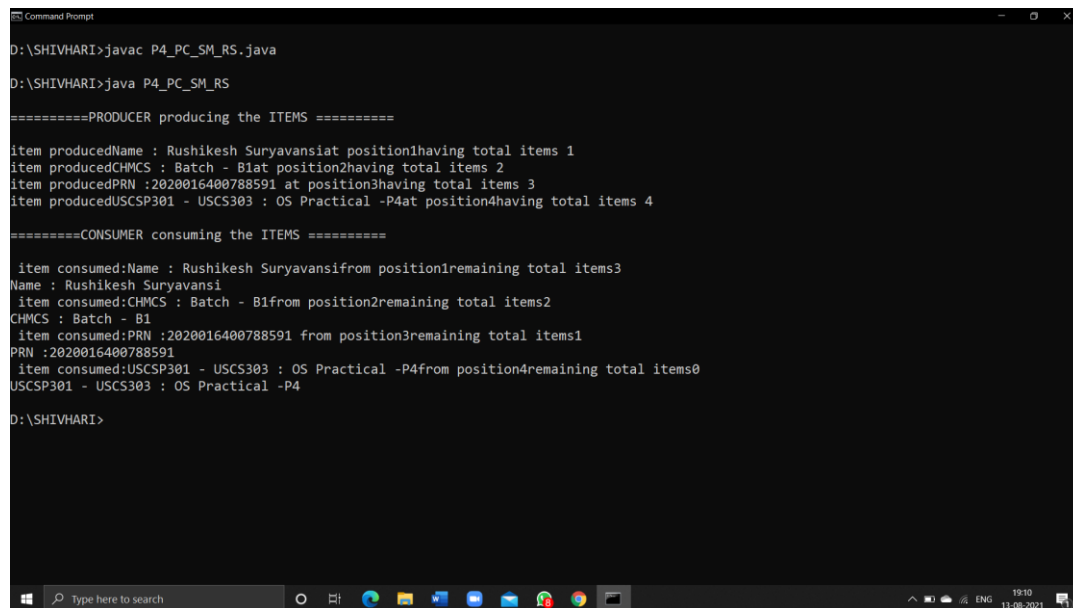
System.out.println(bufobj.remove());

System.out.println(bufobj.remove());

} //main ends

} // class ends
```

## OUTPUT:



```
Command Prompt
D:\SHIVHARI>javac P4_PC_SM_RS.java
D:\SHIVHARI>java P4_PC_SM_RS

=====PRODUCER producing the ITEMS =====
item producedName : Rushikesh Suryavansiat position1having total items 1
item producedCHMCS : Batch - B1at position2having total items 2
item producedPRN :2020016400788591 at position3having total items 3
item producedUSCSP301 - USCS303 : OS Practical -P4at position4having total items 4

=====CONSUMER consuming the ITEMS =====
item consumedName : Rushikesh Suryavansifrom position1remaining total items3
Name : Rushikesh Suryavansi
item consumed:CHMCS : Batch - B1from position2remaining total items2
CHMCS : Batch - B1
item consumed:PRN :2020016400788591 from position3remaining total items1
PRN :2020016400788591
item consumed:USCSP301 - USCS303 : OS Practical -P4from position4remaining total items0
USCSP301 - USCS303 : OS Practical -P4
D:\SHIVHARI>
```

# Smt.Chandabai Himathmal Mansukhani

f)using message passing:

File 1: **P4\_PC\_MP\_Channel\_RS**

//Name: SHIVHARI CHAVAN

//Batch:B2

//PRN: 2018016401247454

//Date:13/8/2021

//Prac-04:Process Communication.

```
public interface P4_PC_MP_Channel_RS<E>
```

```
{
```

```
//send a message to the channel
```

```
public void send( E item);
```

```
// receive a message from the channel
```

```
public E receive();
```

```
} // interface ends
```

File 2: **P4\_PC\_MP\_MessageQueue\_RS**

//Name: SHIVHARI CHAVAN

//Batch:B2

//PRN: 2018016401247454

//Date:13/8/2021

//Prac-04:Process Communication.

```
import java.util.Vector;
```

```
public class P4_PC_MP_MessageQueue_RS<E> implements P4_PC_MP_Channel_RS<E>
```

```
{
```

# Smt.Chandabai Himathmal Mansukhani

```
private Vector<E> queue;

public P4_PC_MP_MessageQueue_RS()

{queue = new Vector<E>();

}

//this implement or non blocking send

public void send ( E item)

{

queue.addElement(item);

} // send() ends

// this implements a non blocking receive

public E receive() {

if (queue.size() ==0)

return null;

else
```

# Smt.Chandabai Himathmal Mansukhani

```
return queue.remove(0);
```

```
} // receive() ends
```

```
} // class ends
```

# Smt.Chandabai Himathmal Mansukhani

File 3: **P4\_PC\_MP\_RS**

//Name: SHIVHARI CHAVAN

//Batch:B2

//PRN:2018016401247454

//Date:13/8/2021

//Prac-04:Process Communication.

```
import java.util.Date;

public class P4_PC_MP_RS
{
    public static void main (String args[])
    {
        // producer and consumer process
        P4_PC_MP_Channel_RS<Date> mailBox = new P4_PC_MP_MessageQueue_RS<Date>();
        int i=0;
        do
        {
            Date message= new Date();
            System.out.println("producer produced-"+ (i+1) + ":" + message);
            mailBox.send(message);
            Date rightNow =mailBox.receive();
            if (rightNow != null)
            {
                System.out.println("consumer consumed -" + (i+1) + ":" + rightNow);
            }
            i++;
        } while(i<10);
    } // main ends
} //class ends
```

**OUTPUT:**

# Smt.Chandabai Himathmal Mansukhani

```
Command Prompt
D:\SHIVHARI>javac P4_PC_MP_RS.java

D:\SHIVHARI>java P4_PC_MP_RS
producer produced-1:Fri Aug 13 19:14:17 IST 2021
consumer consumed -1:Fri Aug 13 19:14:17 IST 2021
producer produced-2:Fri Aug 13 19:14:17 IST 2021
consumer consumed -2:Fri Aug 13 19:14:17 IST 2021
producer produced-3:Fri Aug 13 19:14:17 IST 2021
consumer consumed -3:Fri Aug 13 19:14:17 IST 2021
producer produced-4:Fri Aug 13 19:14:17 IST 2021
consumer consumed -4:Fri Aug 13 19:14:17 IST 2021
producer produced-5:Fri Aug 13 19:14:17 IST 2021
consumer consumed -5:Fri Aug 13 19:14:17 IST 2021
producer produced-6:Fri Aug 13 19:14:17 IST 2021
consumer consumed -6:Fri Aug 13 19:14:17 IST 2021
producer produced-7:Fri Aug 13 19:14:17 IST 2021
consumer consumed -7:Fri Aug 13 19:14:17 IST 2021
producer produced-8:Fri Aug 13 19:14:17 IST 2021
consumer consumed -8:Fri Aug 13 19:14:17 IST 2021
producer produced-9:Fri Aug 13 19:14:17 IST 2021
consumer consumed -9:Fri Aug 13 19:14:17 IST 2021
producer produced-10:Fri Aug 13 19:14:17 IST 2021
consumer consumed -10:Fri Aug 13 19:14:17 IST 2021

D:\SHIVHARI>
```

# Smt.Chandabai Himathmal Mansukhani

g) Remote Procedure Calls

**Question -03 write a Java Program for adding ,subtracting, multiplying and dividing two numbers.**

**Steps for writing RMI program:**

□ **Step 1 creating the remote interface**

**This file define the remote interface that is provided by the server. It contains four methods that accepts two integer argument and return their sum ,difference ,product and quotient. All remote interfaces must extend the remote interface, which is part of java.rmi. Remote defines no members. Its purpose is simply to indicate that an interface uses remote methods. All remote methods can throw a RemoteException.**

File 1: **P4\_RMI\_CalcServerIntf\_RS**

//Name: SHIVHARI CHAVAN

//Batch: B2

//PRN: 2018016401247454

//Date: 13/8/2021

//Prac-04:Process Communication.

```
import java.rmi.*;

public interface P4_RMI_CalcServerIntf_RS extends
Remote
{

    int add(int a,int b) throws RemoteException;
    int subtract(int a,int b) throws
RemoteException; int multiply(int a,int b)
throws RemoteException; int divide(int a,int
b) throws RemoteException;
```



# Smt.Chandabai Himathmal Mansukhani

```
} //interface ends
```

## □ Step 2: Implementing the remote interface.

**This file implements the remote interface. The implementation of all the four methods is straight forward. All remote methods must extend UnicastRemoteObject, which provides functionality that is needed to make objects available for remote machines.**

File 2: **P4\_RMI\_CalcServerImpl\_RS**

//Name: SHIVHARI CHAVAN

//Batch:B2

//PRN: 2018016401247454

//Date:13/8/2021

//Prac-04:Process Communication.

```
import java.rmi.*; import java.rmi.server.*; public class P4_RMI_CalcServerImpl_RS extends
UnicastRemoteObject implements P4_RMI_CalcServerIntf_RS
```

```
{
    public P4_RMI_CalcServerImpl_RS() throws RemoteException {
    }
    public int add (int a,int b) throws RemoteException
    {
        return a+b;
    }
    public int subtract(int a,int b) throws RemoteException
    {
        return a-b;
    }
    public int multiply(int a,int b) throws RemoteException
    {
```

# Smt.Chandabai Himathmal Mansukhani

```
return a*b;
}
public int divide(int a,int b) throws RemoteException
{
    return a/b;
}

} //class ends
```

## □ Step 3 : Creating the Server

**This file contain the main program for the server machine. Its primary function is to update the RMI registry on that machine. This is done by using the rebind() method of the Naming class (found in java.rmi). that method associate a name with an object reference. The first argument to be rebind() method is a string that need to server. Its second argument is a reference to an instance of CalcServerImpl.**

File 3: **P4\_RMI\_CalcServer\_RS**

//Name: SHIVHARI CHAVAN

//Batch:B2

//PRN: 2018016401247454

//Date: 13/8/2021

//Prac-04:Process Communication.

```
import java.net.*; import java.rmi.*;
public class P4_RMI_CalcServer_RS
{ public static void main(String args[])
{
```

# Smt.Chandabai Himathmal Mansukhani

```
try
{
    P4_RMI_CalcServerImpl_RS csi = new P4_RMI_CalcServerImpl_RS();

    Naming.rebind("CSB2",csi);

} // try ends

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

} // catch ends

} //main ends

} // class ends
```

## □ Step 4: creating the client

**This file implement the client side of this distributed application. it accepts 3 command-line arguments. The first is the IP address or name of the server machine. The second and third arguments are the two numbers that are to be operated.**

**The application begins by forming a string that follows the URL syntax. This URL uses the RMI protocol. The string include includes the IP address or name of the server and the server “CSB2”. The program then invokes the lookup() method of the Naming class. This method accepts one argument, the rmi URL, and returns a reference to an object of type CalcServerInf. All remote Method invocation can be directed to this object.**

File 4: **P4\_RMI\_CalcClient\_RS**

//Name: SHIVHARI CHAVAN

//Batch:B2

//PRN:2018016401247454

//Date:13/8/2021

//Prac-04:Process Communication.

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

```
public class P4_RMI_CalcClient_RS
```

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

```
try{ String CSURL="rmi://" +args[0]+"/CSB2";
```

# Smt.Chandabai Himathmal Mansukhani

```
P4_RMI_CalcServerIntf_RS CSIntf = (P4_RMI_CalcServerIntf_RS) Naming.lookup(CSURL);
System.out.println("The first number is: " + args[1]); int x =
Integer.parseInt(args[1]);
System.out.println("The second number is: " + args[2]);
int y = Integer.parseInt(args[2]);
System.out.println("=====Arithmetic Operations=====");
System.out.println("Addition: "+x+"+"+y+"="+ CSIntf.add(x,y));
System.out.println("Subtraction: "+ x + " - "+ y + " = " +CSIntf.subtract(x,y));
System.out.println("Multiplication: " + x + " * "+ y + " = "+CSIntf.multiply(x,y));
System.out.println("Division: "+x+"/"++y+" = " +CSIntf.divide(x,y));
} // try ends catch(Exception
e){
    System.out.println("Exception: " + e);
} // catch ends
} // main ends
} // class ends
```

## Step 5 : manually generate a stub, if required

Prior to Java 5, stubs needed to be built manually by using `rmic`. This step is not required for modern version of Java. However, if we work in a agency environment, then we can use the `mic` compiler, as shown here, to build a stub.

`rmic CalcServerImpl`

□

## Step 6 : install files on the client and server machines.

Copy `P4_RMI_CalcClient_RS.class` ,`P4_RMI_CalcServerImpl_RS_Stub.class` ( if needed) and `P4_RMI_CalcServerIntf_RS.class` and to a directory on the client machine.

Copy `CalcServerIntf.class` , `P4_RMI_CalcServerImpl_RS.class` , `P4_RMI_CalcServerImpl_RS_Stub.class` (is needed), and `P4_RMI_CalcServer_RS_Stub.class` to a directory on the server machine.

## □ Step 7 :start the RMI registry on the server machine

The JDK provides a program called `rmiregistry`, which executes on the server machine. It maps names to object references. Start the RMI register Registry the command line, as shown here: start `rmiregistry`

When this command returns, a new window gets created. Leave this window open until we are done experimenting with the RMI example.

# Smt.Chandabai Himathmal Mansukhani

## □ Step 8 :start the server

The server code is started from the command line, as shown here :

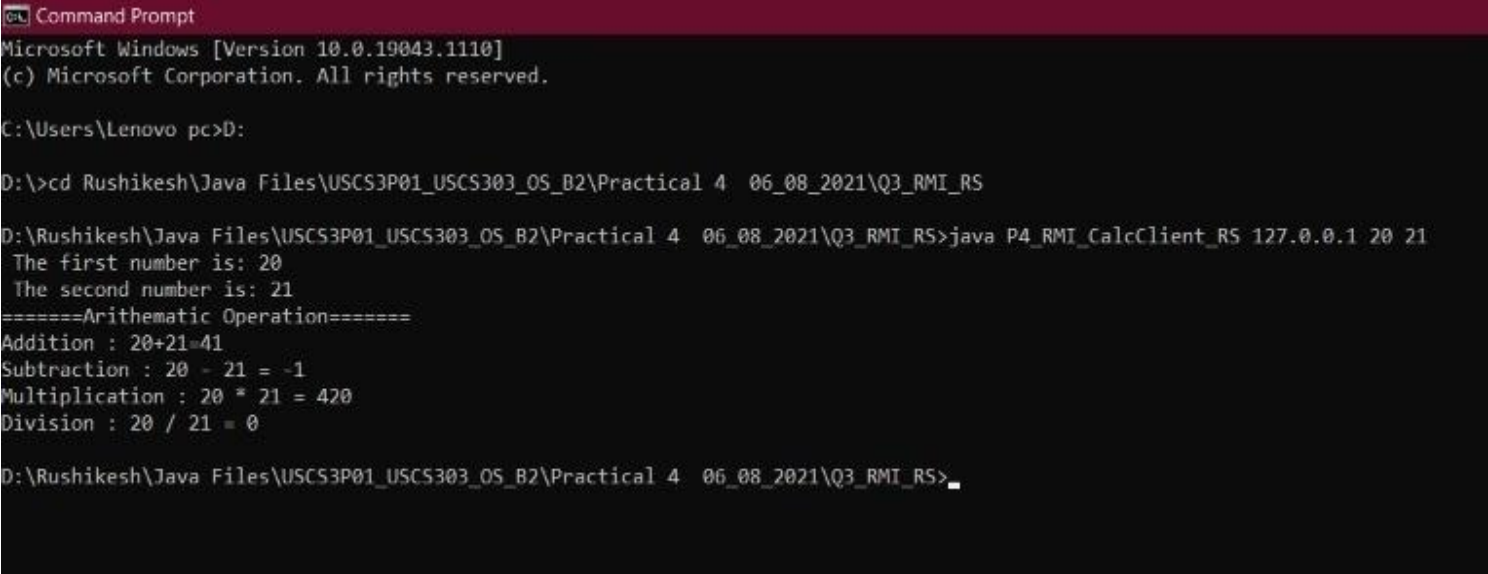
Java P4\_RMI\_CalcServer\_RS

## □ Step 9 :start the client

The client code is started from the command line, as shown here:

Java P4\_RMI\_CalcClient\_RS 127.0.0.1 15 5

## OUTPUT 1:



```
Command Prompt
Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo pc>D:

D:\>cd Rushikesh\Java Files\USCS3P01_USCS303_OS_B2\Practical 4 06_08_2021\Q3_RMI_RS

D:\Rushikesh\Java Files\USCS3P01_USCS303_OS_B2\Practical 4 06_08_2021\Q3_RMI_RS>java P4_RMI_CalcClient_RS 127.0.0.1 20 21
The first number is: 20
The second number is: 21
=====Arithmetic Operation=====
Addition : 20+21=41
Subtraction : 20 - 21 = -1
Multiplication : 20 * 21 = 420
Division : 20 / 21 = 0

D:\Rushikesh\Java Files\USCS3P01_USCS303_OS_B2\Practical 4 06_08_2021\Q3_RMI_RS>
```

## OUTPUT 2:

# Smt.Chandabai Himathmal Mansukhani

```
Command Prompt - java P4_RMI_CalcServer_RS
Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo pc>D:

D:\>cd Rushikesh\Java Files\USCS3P01_USCS303_OS_B2\Practical 4 06_08_2021\Q3_RMI_RS

D:\Rushikesh\Java Files\USCS3P01_USCS303_OS_B2\Practical 4 06_08_2021\Q3_RMI_RS>javac *.java

D:\Rushikesh\Java Files\USCS3P01_USCS303_OS_B2\Practical 4 06_08_2021\Q3_RMI_RS>start rmiregistry

D:\Rushikesh\Java Files\USCS3P01_USCS303_OS_B2\Practical 4 06_08_2021\Q3_RMI_RS>java P4_RMI_CalcClient_RS
Exception : java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0

D:\Rushikesh\Java Files\USCS3P01_USCS303_OS_B2\Practical 4 06_08_2021\Q3_RMI_RS>java P4_RMI_CalcServer_RS
```

..