

Smt. Chandibai Himathmal Mansukhani College

USCS3P01:USCS303-Operating System (OS) Practical-04

Process Communication

Contents

USCS3P01:USCS303-Operating System (OS) Practical-04	1
Process Communication.....	1
Practical Date : 6 th August,2021 (Friday)	2
Practical Aim : (Producer – Consumer Problem ,RMI)	2
Process Communication	2
Producer – Consumer Problem	2
Using Shared Memory.....	3
1. Producer – Consumer Solution Using Shared Memory	5
Question – 1	5
Write a java program for producer – consumer using shared memory	5
Implement Producer – Consumer solution using shared memory	5
Source Codes	5
Output:	8
Using Message Passing.....	9
2. Producer – Consumer Solution Using Message Passing	9
Question – 2	9
Write a java program for producer – consumer using message passing.....	9
Implement Producer – Consumer solution using Message Passing	10
Source Code :	10
Output:	12
Remote Method Invocation	12
3. Implement Remote Method Invocation (RMI)Calculator	14
Question – 3	14
Write a java program for adding , subtracting , multiplying and dividing two numbers.....	14
Source codes	14
Output:	20

Smt. Chandibai Himathmal Mansukhani College

Practical Date : 6 th August,2021 (Friday)

Practical Aim : (Producer – Consumer Problem ,RMI)

Producer – Consumer Problem

Process Communication

Content:

- ❖ Solution for Producer – Consumer Problem using shared memory and message passing
- ❖ Communication in Client Server environment using Remote Method Invocation (RMI).

Process:

- ❖ Producer – Consumer problem without threads and without synchronization.
- ❖ Implement remote method Invocation (RMI)

Prior Knowledge:

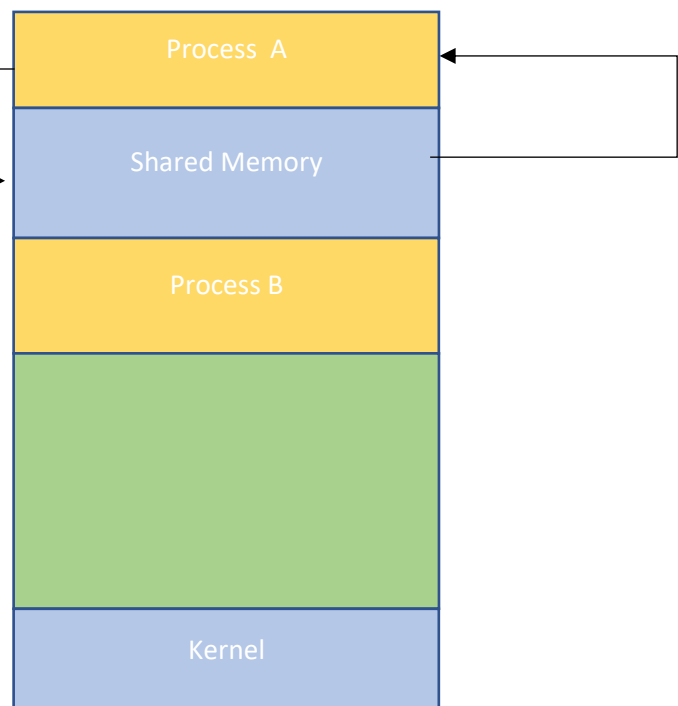
- ❖ Concept of shared memory , message passing , interfaces and remote method Invoaction.
- ❖ Message passing and Remote proceduring cells are the most common methods of interprocess communication in distributed systems.
- ❖ A less frequently used but no less valuable method is distributed shared memory

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 print spooling. A word processor spools data to 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.

Using Shared Memory

- Shared memory is memory that may be simultaneously accessed by multiple processes with an intent to provide communication among them or avoid redundant copies.
- Shared memory is an efficient means of passing data between processes.



Smt. Chandibai Himathmal Mansukhani College

1. Producer – Consumer Solution Using Shared Memory

Question – 1

Write a java program for producer – consumer using shared memory

Implement Producer – Consumer solution using shared memory

Source Codes

Code 1:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

```
public interface P4_PC_SM_Buffer_SJ  
{  
  
    //Procedures call this method  
    public void insert(String item);  
  
    //Consumers call this method  
    public String remove();  
  
}//interface ends
```

Smt. Chandibai Himathmal Mansukhani College

Code 2:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

```
public class P4_PC_SM_BufferImpl_SJ implements P4_PC_SM_Buffer_SJ
{
    private static final int BUFFER_SIZE = 5;
    private String[] elements;
    private int in,out,count;

    public P4_PC_SM_BufferImpl_SJ()
    {
        count = 0;
        in = 0;
        out = 0;
        elements = new String[BUFFER_SIZE];
    }

    public void insert(String item)
    {
        while(count==BUFFER_SIZE);

        elements[in] = item;
        in = (in + 1)%BUFFER_SIZE;
        ++count;
        System.out.println("Item Produced " + item + " at position " + in + " having
total items " + count);
    }
    public String remove()
    {
        String item;
        while(count==0);

        item = elements[out];
        out = (out + 1)%BUFFER_SIZE;
        --count;
        System.out.println("Item Consumed " + item + " from position " + out + "
remaining total items " + count);
    }
}
```

Smt. Chandibai Himathmal Mansukhani College

```
        return item;
    }
}
```

Code 3:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

```
public class P4_PC_SM_SJ
{
```

```
    public static void main(String[] args){
        P4_PC_SM_BufferImpl_SJ bufobj = new P4_PC_SM_BufferImpl_SJ();

        System.out.println("\n=====PRODUCER producing the
ITEMS=====\\n");
        bufobj.insert("Name: Sahil Jadhav");
        bufobj.insert("CHMCS: Batch - B2");
        bufobj.insert("PRN: 2020016400783091");
        bufobj.insert("USCSP301 - USCS303:0S Practical - P4");

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

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

Smt. Chandibai Himathmal Mansukhani College

Output:

```
Command Prompt
Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\91766>cd C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q1_PC_SM_SJ

C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q1_PC_SM_SJ>set path="C:\Program Files\Java\jdk-16.0.1\bin"

C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q1_PC_SM_SJ>javac P4_PC_SM_SJ.java

C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q1_PC_SM_SJ>java P4_PC_SM_SJ

=====PRODUCER producing the ITEMS=====

Item Produced Name: Sahil Jadhav at position 1 having total items 1
Item Produced CHMCS: Batch - B2 at position 2 having total items 2
Item Produced PRN: 2020016400783091 at position 3 having total items 3
Item Produced USCSP301 - USCS303:0S Practical - P4 at position 4 having total items 4

=====CONSUMER consuming the ITEMS=====

Item Consumed Name: Sahil Jadhav from position 1 remaining total items 3
Name: Sahil Jadhav
Item Consumed CHMCS: Batch - B2 from position 2 remaining total items 2
CHMCS: Batch - B2
Item Consumed PRN: 2020016400783091 from position 3 remaining total items 1
PRN: 2020016400783091
Item Consumed USCSP301 - USCS303:0S Practical - P4 from position 4 remaining total items 0
USCSP301 - USCS303:0S Practical - P4

C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q1_PC_SM_SJ>_
```


Using Message Passing

- Message passing is the basic of most inter- process communication in distributed systems.
- It is at the 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 these processes.
- Communication in the message passing paradigm , in its simplest form , is performed using the send() and received() 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 a parameter for the send() primitive would enable the receiver to acknowledge the message. The receive() primitive requires the name of the anticipated sender and should provide a storage buffer for the message.

Smt. Chandibai Himathmal Mansukhani College

2. Producer – Consumer Solution Using Message Passing

Question – 2

Write a java program for producer – consumer using message passing

Implement Producer – Consumer solution using Message Passing

Source Code :

Code 1:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

```
public interface P4_PC_MP_Channel_SJ<E>
{
    // Send a message to the channel
    public void send(E item);

    // Receive a message from the channel
    public E receive();
} // interface ends
```

Code 2:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

import java.util.Vector;

```
public class P4_PC_MP_MessageQueue_SJ<E> implements
P4_PC_MP_Channel_SJ<E>
```

```
{
```

```
    private Vector<E> queue;
```

```
    public P4_PC_MP_MessageQueue_SJ(){
```

Smt. Chandibai Himathmal Mansukhani College

```
        queue = new Vector<E>();
    }

    // This implements a nonblocking send
    public void send(E item){
        queue.addElement(item);
    } // send() ends

    // This implements a nonblocking receive
    public E receive(){
        if(queue.size()==0)
            return null;
        else
            return queue.remove(0);
    } // receive() ends
} class ends
```

Code 3:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

```
import java.util.Date;
public class P4_PC_MP_SJ
{
    public static void main(String args[])
    {
        P4_PC_MP_Channel_SJ<Date> mailBox = new
P4_PC_MP_MessageQueue_SJ<Date>();

        int i = 0;

        do
        {
```

Smt. Chandibai Himathmal Mansukhani College

```
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);

}

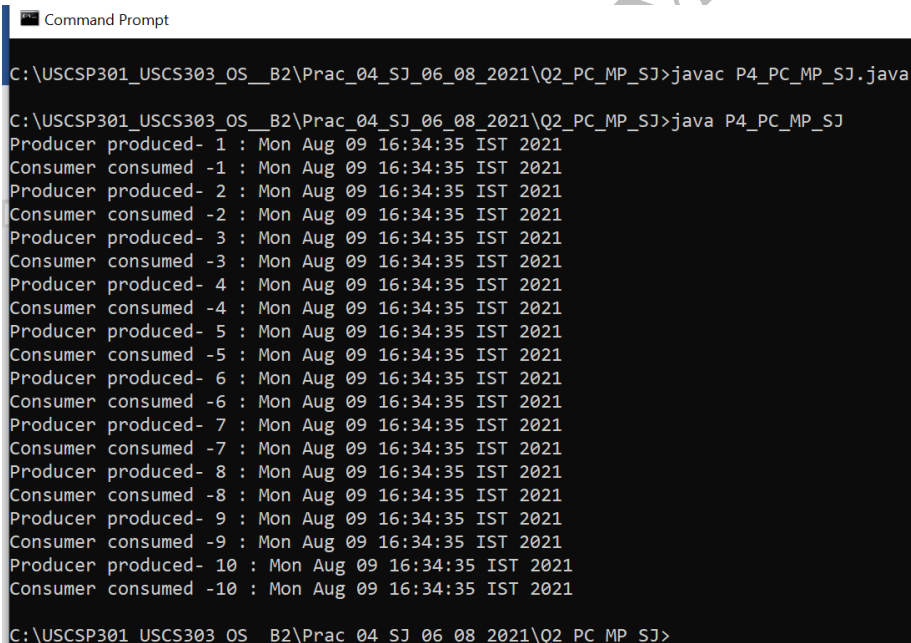
    i++;

}while(i<10);

}

}
```

Output:



```
Command Prompt

C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q2_PC_MP_SJ>javac P4_PC_MP_SJ.java

C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q2_PC_MP_SJ>java P4_PC_MP_SJ
Producer produced- 1 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -1 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 2 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -2 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 3 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -3 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 4 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -4 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 5 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -5 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 6 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -6 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 7 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -7 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 8 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -8 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 9 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -9 : Mon Aug 09 16:34:35 IST 2021
Producer produced- 10 : Mon Aug 09 16:34:35 IST 2021
Consumer consumed -10 : Mon Aug 09 16:34:35 IST 2021

C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q2_PC_MP_SJ>
```

Smt. Chandibai Himathmal Mansukhani College

Remote Method Invocation

Remote Procedure Calls

- Message passing leaves the programming with the burden of the explicit 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 blocks at the call() until the reply is received.
- The remote procedure is the server 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() when 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() generates a client stub which marshals the arguments into a message and sends the message to the server machine. On receipt of the message the server stub is generated and extracts 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. Chandibai Himathmal Mansukhani College

Question – 3

Write a java program for adding , subtracting , multiplying and dividing two numbers.

3. Implement Remote Method Invocation (RMI) Calculator

Step 1:

Creating the Remote Interface

This file defines the remote interface that is provided by the server. It contains four methods that accepts two Integer arguments and returns 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 and interface uses remote uses remote methods.

All remote methods can throw a RemoteException.

Source Codes :

Code 1:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

import java.rmi.*;

public interface P4_RMI_CalcServerIntf_SJ 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;

}//interface ends

Smt. Chandibai Himathmal Mansukhani College

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 from remote machines.

Code 2:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

import java.rmi.*;

import java.rmi.server.*;

**public class P4_RMI_CalcServerImpl_SJ extends UnicastRemoteObject implements
P4_RMI_CalcServerIntf_SJ**

{

public P4_RMI_CalcServerImpl_SJ()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. Chandibai Himathmal Mansukhani College

```
        return a * b;
    }

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

Step 3:

Creating the server

This file contains 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 associates a name with an object reference. The first argument to the rebind() method is a string that names the server. Its second argument is a reference to an interface of CalcServerImpl

Code 3:

//Name: Sahil Jadhav

//Batch No:B2

//PRN:2020016400783091

//Date:06-08-2021

```
import java.net.*;
import java.rmi.*;
public class P4_RMI_CalcServer_SJ
{
    public static void main(String args[])
    {
        try
        {
            P4_RMI_CalcServerImpl_SJ csi = new P4_RMI_CalcServerImpl_SJ();
            Naming.rebind( "CSBO" , csi );
        }
    }
}
```


Smt. Chandibai Himathmal Mansukhani College

```
        catch(Exception e){  
System.out.println( "Exception : " + e);  
        }//catch ends  
    }//main ends  
}//class end
```

Step 4:

Creating the Client

- This file implements the client side of this distributed application. It accepts three 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.
- This application begins by forming a string that follows the URL syntax.
- This URL uses the rmi protocol. The string includes the IP address or name of the server and the string “CSBO”. The program that invokes the lookup() method of the Naming class. This method accepts one argument , the rmi URL , and returns a reference to an object of type CalcServerIntf. All remote method invocations can then be directed to this object.

Code 4:

```
//Name: Sahil Jadhav  
//Batch No:B2  
//PRN:2020016400783091  
//Date:06-08-2021  
import java.rmi.*;  
public class P4_RMI_CalcClient_SJ  
{  
    public static void main(String args[])  
    {  
        try {  
            String CSURL = "rmi://" + args[0] + "/CSBO";  
            P4_RMI_CalcServerIntf_SJ CSIntf = (P4_RMI_CalcServerIntf_SJ)  
Naming.lookup(CSURL);  
            System.out.println(" The first number is: " + args[1]);
```

Smt. Chandibai Himathmal Mansukhani College

```
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 Operation-----");

        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 versions of Java. However, if we work in a legacy environment, then we can use rmic compiler, as shown here, to build a stub.

```
rmic CalcsServerImpl
```

Smt. Chandibai Himathmal Mansukhani College

Step 6:

Install Files on the Client and Server Machines

- Copy P4_RMI_calcClient_JD.class , P4_RMI_CalcServerImpl_JD_Stub.class(if needed),and P4_RMI_CalcServerintf_JD.class to a directory on the client machine.
- Copy CalcServerintf.class , P4_RMI_CalcServerImpl_JD.class , P4_RMI_CalcServerImpl_JD_Stub.class (if needed), and P4_RMI_CalcServer_JD.class to a directory on the server machine.

Step 7:

Start the RMI Registry on the ServerMachine

- The JDK provides a program called rmiregistry , which executes on the server machine.

It maps names to object references. Start the RMI Registry form 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. Chandibai Himathmal Mansukhani College

Step 8:

Start the Server

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

Java P4_RMI_CalcServer_MD

Output:

Terminal 1:

```
C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q3_RMI_SJ>javac *.java
C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q3_RMI_SJ>start rmiregistry
C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q3_RMI_SJ>javac P4_RMI_CalcServer_SJ.java
C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q3_RMI_SJ>java P4_RMI_CalcServer_SJ
```

Terminal 2:

```
C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q3_RMI_SJ>javac P4_RMI_CalcClient_SJ.java
C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q3_RMI_SJ>java P4_RMI_CalcClient_SJ 127.0.0.1 87 42
The first number is: 87
The second number is: 42
-----Arithmetic Operation-----
Addition : 87 + 42 = 129
Subtraction : 87 - 42 = 45
Multiplication : 87 * 42 = 3654
Division : 87 / 42 = 2
C:\USCSP301_USCS303_OS_B2\Prac_04_SJ_06_08_2021\Q3_RMI_SJ>_
```