Content

**USCSP301-USCS303:Operating System(OS)Practical-04**

**Practical-04:**Process Communication **Practical Aim:**Producer-Consumer Problem,RMI Process Communication....................................................................1 Producer-Consumer Problem............................................................2 Using Shared Memory.......................................................................3 1.Producer-Consumer Solution using Shared Memory

Using Message Passing......................................................................4 2.Producer-Consumer Solution using Message Passing

Remote Procedure Calls.....................................................................5 3.Remote method Invocation(RMI)Calculator

Process Communication

1.Process often need to communicate with eachother.

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

3.Inter-processes communication provides a means for process to co-operate and compete.

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

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

Producer-Consumer Problem

1.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.

2.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 the data onto the compact disc.

Producer-Consumer Problem Solution

Using Shared Memory

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

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

|  |
| --- |
| Process A |
| Shared memory |
| Process B |
|  |
| Kernal |

Question-01:

Write a java program for producer-consumer problem using shared memory.

Source Code:1

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

public interface P4\_PC\_SM\_Buffer\_SS

{

//Producer call this method

public void insert(String item);

//Consumers call this method

public String remove();

}//interface ends

Source code:2

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

public class P4\_PC\_SM\_BufferImpl\_SS implements

P4\_PC\_SM\_Buffer\_SS

{

private static final int BUFFER\_SIZE = 5;

private String[] elements;

private int in,out,count;

public P4\_PC\_SM\_BufferImpl\_SS() //constructor initializing the variables to initial value

{

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

Source code:3

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

public class P4\_PC\_SM\_SS

{

public static void main(String[]args){

P4\_PC\_SM\_BufferImpl\_SS bufobj = new P4\_PC\_SM\_BufferImpl\_SS();

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

bufobj.insert("Name:Shraddha Sawant");

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

bufobj.insert("PRN:2020016400773862");

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

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

String element=bufobj.remove();

System.out.println(element);

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

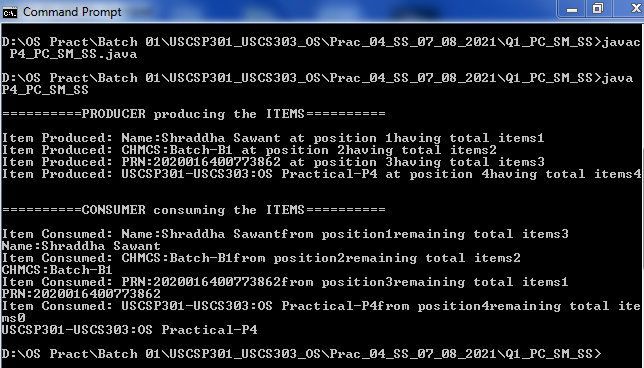
System.out.println(bufobj.remove());

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

}//main ends

}//class ends

Output:



Producer-Consumer Problem Solution

Using Message Passing

1.Message passing is the basis of most iner-process communication in distributed systems.

2.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.

3.communication in the message passing paradigm,in it simplest form,is performed using the send() and receive() primitives.The syntax is generally of the form:

Send(receiver,message)

Receive(sender,message)

4.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.

Question:2

Write a java program for producer-consumer problem using message passing.

Source code:1

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

public interface P4\_PC\_MP\_Channel\_SS<E>

{

//send a message to the channel

public void send(E item);

//Receive a message from the channel

public E receive();

}//interface ends

Source code:2

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

import java.util.Vector;

public class P4\_PC\_MP\_MessageQueue\_SS<E> implements P4\_PC\_MP\_Channel\_SS<E>

{

private Vector<E> queue;

public P4\_PC\_MP\_MessageQueue\_SS(){

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

Source code:3

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

import java.util.Date;

public class P4\_PC\_MP\_SS

{

public static void main(String args[])

{

//Producer and consumer process

P4\_PC\_MP\_Channel\_SS<Date> mailBox=new P4\_PC\_MP\_MessageQueue\_SS<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);

}

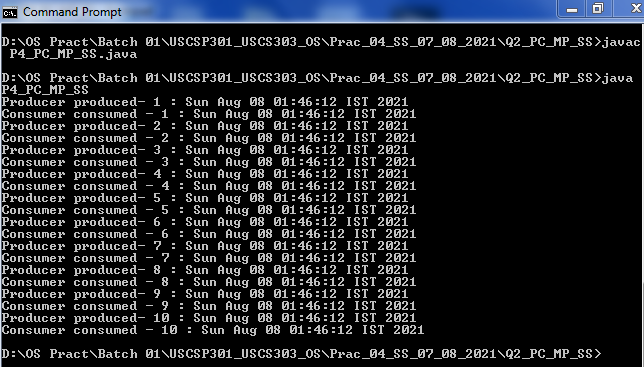
i++;

}while(i<10);

}//main ends

}//class ends

Output:



Remote Method Invocation

Remote Procedure Calls

1.Message passing leaves the programmer 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.

2.Syntax:

3.The syntax of a remote procedure call is generally of the form:

Call procedure\_name(value\_arguments;result\_arguments)

4.The client process blocks at the call() until the reply is received.

5.The remote procedure is the server processes which has already begun executing on a remote machine.

6.It blocks at the receive() until it receives a message and parameters from the sender.

7.The server then sends a reply() when it has finished its task.

Remote Method Invocation

Remote Procedure Calls

1.Message passing leaves the programmer 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.

2.Syntax:

3.The syntax of a remote procedure call is generally of the form:

Call procedure\_name(value\_arguments;result\_arguments)

4.The client process blocks at the call() until the reply is received.

5.The remote procedure is the server processes which has already begun executing on a remote machine.

6.It blocks at the receive() until it receives a message and parameters from the sender.

7.The server then sends a reply() when it has finished its task.

8.The syntax is as follows:

Receive procedure\_name(in value\_parameters; out result\_parameters)

Reply(caller, result\_parameters)

9.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 servermachine. 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.

Question:3

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

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 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 and interface uses remote uses remote methods.All remote methods can throw a RemoteException.

Source code:1

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

import java.rmi.\*;

public interface P4\_RMI\_CalcServerIntf\_SS 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

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 UnicasrRemoteObjest,which provides functionality that is needed to make objects available from remote machines.

Source Code:3

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

import java.rmi.\*;

import java.rmi.server.\*;

public class P4\_RMI\_CalcServerImpl\_SS extends UnicastRemoteObject implements P4\_RMI\_CalcServerIntf\_SS

{

public P4\_RMI\_CalcServerImpl\_SS() 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

{

return a\*b;

}

public int divide(int a,int b) throws RemoteException

{

return a/b;

}

}//class ends

STEP 3:

Creating the Remote Interface

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 straight that names the server.Its second argument is a reference to an interface of CalcServerImpl.

Source Code:3

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 8TH Aug 2021

//PRAC-04: PROCESS COMMUNICATION

import java.net.\*;

import java.rmi.\*;

public class P4\_RMI\_CalcServer\_SS

{

public static void main(String args[])

{

try

{

P4\_RMI\_CalcServerImpl\_SS csi=new P4\_RMI\_CalcServerImpl\_SS();

Naming.rebind("CSB1",csi);

}//try ends

catch(Exception e) {

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

}//catch ends

}//main ends

}//class ends

STEP 4:

Creating the Client

1.This file implements the client side of this distributes 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.

2.This application begins by forming a string that follows the URL syntax.

3.This URL uses the rmi protocol.The string includes the IP address or name of the server and the string “CSB1”.The program that invokes the lookup() method of the Naming class.This method accepts one argument,the rmi URL,and returns a reference to ab object of type CalcServerinf.All remote method invocations can then be directed to this object.

Source Code:4

import java.rmi.\*;

public class P4\_RMI\_CalcClient\_SS

{

public static void main(String args[])

{

try{

String CSURL = "rmi://"+args[0]+"/CSB1";

P4\_RMI\_CalcServerIntf\_SS CSIntf= (P4\_RMI\_CalcServerIntf\_SS) 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 require 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.

Rimc CalcServerImpl

STEP 6:

Install Files on the Client and Server Machines

1.Copy P\_RMI\_CalcClient\_SS.class, P4\_RMI\_CalcServerImpl\_SS\_Stub.class(if needed), and P4\_RMI\_CalcServerIntf\_SS.class to a directory on the client machine.

2.Copy CalcServerIntf.class, P4\_RMI\_CalcServerImpl\_SS.class,

P4\_RMI\_CalcServerImpl\_SS\_Stub.class(if needed), and

P4\_RMI\_CalcServer\_SS.class to a directory on the server machine.

STEP 7:

Start the RMI Register on the ServerMachine

1.The JDK provides a program called rmiregistry,which executes on the server machine.It maps names to object reference.Start the RMI Registry from the command line, as shown here

Start rmiregistry

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

STEP 8:

Start the Server

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

Java P4\_RMI\_CalcServer\_SS

STEP 9:

Start the Client

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

Java P4\_RMI\_CalcClient\_SS 127.0.0.1 20 02

OUTPUT:

