

Session: 2021-22(Even)

|  |  |
| --- | --- |
| **Session:** | **2021-22 (Even)** |
| **Subject:** | **Distributed Operating System Practical(DOS)** |
| **Year:** | **IV** |
| **Semester:** | **VIII** |
| **Name of Student:** | **Palak Maheshwari** |
| **Batch:** | **B2** |

c ***Department of Computer Science & Engineering, S.B.J.I.T.M.R., Nagpur***

# Practical No.1

## **Aim:** Construct program to illustrate concept of client server application.

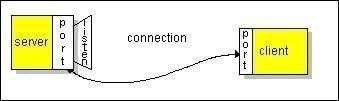
c ***Department of Computer Science & Engineering, S.B.J.I.T.M.R., Nagpur***

**AIM:** Construct program to illustrate concept of client server application.

**OBJECTIVES:**

* To know the concept of client server.
* To interpret the process of how communication between client and server is established.
* To know the concept of socket programming in distributed environment.

c ***Department of Computer Science & Engineering, S.B.J.I.T.M.R., Nagpur***



# Practical No.1

**AIM:** Construct program to illustrate concept of client server application

**INLAB**

**AIM**: Construct chat application to demonstrate the concept of echo client server application.

**OBJECTIVES:**

* To know the concept of client server.
* To interpret the process of how communication between client and server is established.
* To know the concept of socket programming in distributed environment.

**THEORY/ALGORITHM:**

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

**Creating a server program:**

The EchoServer example creates a server socket and waits for a client request. When it receives a client request, the server connects to the client and responds to it.

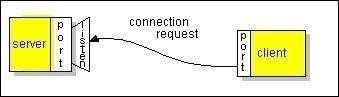
Ping Command

ping IP Address sends an ICMP *ECHO\_REQUEST* packet to the specified host. If the host responds, you get an ICMP packet back. “ping” an IP address is used to see if a machine is alive. If there is no response, there is something wrong.

Following are the steps to create echo server application

**1. Create and open a server socket.**

c ***Department of Computer Science & Engineering, S.B.J.I.T.M.R., Nagpur***



ServerSocket serverSocket = new ServerSocket(portNumber);

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

**1. Wait for the client request.**

Socket clientSocket = serverSocket.accept();

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

**1. Open an input stream and an output stream to the client.**

out = new PrintWriter(clientSocket.getOutputStream(), true);

in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

**1. Communicate with the client.**

Receive data from the client: (inputLine = in.readLine()) Send data to the client:

out.println(inputLine);

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

**Creating a client program:**

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

c ***Department of Computer Science & Engineering, S.B.J.I.T.M.R., Nagpur***

**CODE:**

**Client Side:**

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

**public class EchoClient**

**{**

**public static void main(String[] args)**

**{**

**try**

**{**

**Socket s = new Socket("127.0.0.1", 9999); BufferedReader r = new BufferedReader(new**

**InputStreamReader(s.getInputStream()));**

**PrintWriter w = new PrintWriter(s.getOutputStream(), true); BufferedReader con = new BufferedReader(new**

**InputStreamReader(System.in));**

**String line; do**

**{**

**line = r.readLine(); if ( line != null )**

**System.out.println(line); line = con.readLine(); w.println(line);**

**}**

**while ( !line.trim().equals("bye") );**

**}**

**catch (Exception err)**

**{**

**System.err.println(err);**

**}**

**}**

**}**

**Server Side:**

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

**public class EchoServer**

**{**

**private ServerSocket server; public EchoServer(int portnum)**

**{**

**try**

**{**

c ***Department of Computer Science & Engineering, S.B.J.I.T.M.R., Nagpur***

**server = new ServerSocket(portnum);**

**}**

**catch (Exception err)**

**{**

**System.out.println(err);**

**}**

**}**

**public void serve()**

**{**

**System.out.println("Server Started"); try**

**{**

**while (true)**

**{**

**Socket client = server.accept(); BufferedReader r = new BufferedReader(new**

**InputStreamReader(client.getInputStream()));**

**PrintWriter w = new PrintWriter(client.getOutputStream(), true);**

**w.println("Welcome to the Java EchoServer. Type 'bye' to**

**close.");**

**String line; do**

**{**

**line = r.readLine(); if ( line != null )**

**w.println("Got: "+ line);**

**}**

**while ( !line.trim().equals("bye") ); client.close();**

**}**

**}**

**catch (Exception err)**

**{**

**System.err.println(err);**

**}**

**}**

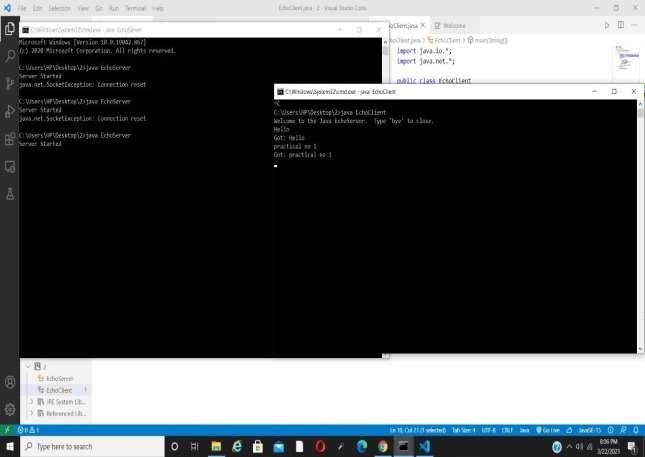
**public static void main(String[] args)**

**{**

**EchoServer s = new EchoServer(9999); s.serve();**

**}**

**}**



**OUTPUT:**

**CONCLUSION:**

In this practical we successfully constructed a program to demonstrate the concept of Client Server application.

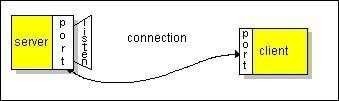
# Practical No. 2

## **Aim:** Construct program to illustrate concept of concurrent clientserver application.

**INLAB AIM:** Construct chat application to demonstrate the concept of concurrent client server application.

**OBJECTIVES:**

* + To know the concept of concurrent client server application.
  + To interpret the process of how concurrent communication between client and server is established.
  + To know the concept of socket programming in distributed environment.



**AIM:** Construct program to illustrate concept of concurrent client server application.

**INLAB AIM:** Construct chat application to demonstrate the concept of concurrent client serverapplication.

**OBJECTIVES:**

* + To know the concept of concurrent client server application.
  + To interpret the process of how concurrent communication between client and server is established.
  + To know the concept of socket programming in distributed environment.

**THEORY/ALGORITHM:**

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

**Creating a server program:**

The Chat Server example creates a server socket and waits for a client request. When itreceives client request, the server connects to the client and responds to it.

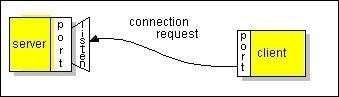
Following are the steps to create echo server application

**Create and open a server socket.**

ServerSocket serverSocket = new ServerSocket(portNumber);

The portNumber argument is the logical address through which the application communicates over the network. It's the port on which the server is running. You must

provide the port number through which the server can listen to the client. Don't select port numbers between 0 and 1,023 because they're reserved for privileged users (that is, super user or root). Add the server socket inside the try-with-resources block.



**Wait for the client request.**

Socket clientSocket = serverSocket.accept();

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

**Open an input stream and an output stream to the client.**

out = new PrintWriter(clientSocket.getOutputStream(), true);

in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

**Server Communicate with the client.**

Receive data from the client: (inputLine = in.readLine())Send data to the client:

out.println(inputLine);

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

**Creating a client program:**

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

Following are the various steps to create client.

**Create and open a client socket.**

Socket echoSocket = new Socket(hostName, portNumber);

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

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

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

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

Receive data from the server: (userInput = stdIn.readLine())Send data to the server: out.println(userInput);

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

**CODE:**

**Server Side Code:-**

**import java.net.\*; import java.io.\*; public class Server {**

**public static void main(String args[]) throws Exception,UnknownHostException{ ServerSocket ss=new ServerSocket(8088);**

**Socket s=ss.accept();;**

**DataInputStream din=new DataInputStream(s.getInputStream()); DataOutputStream dout=new DataOutputStream(s.getOutputStream()); BufferedReader br=new BufferedReader(new**

**InputStreamReader(System.in));**

**String str=&quot;&quot;,str2=&quot;&quot;;**

**while(str!=&quot;stop&quot;)**

**{**

**System.out.println(&quot;Waiting for client&#39;s Reply...&quot;); str=din.readUTF();**

**System.out.println(&quot;Client: &quot;+str); System.out.println(&quot;Enter Message:&quot;); str2=br.readLine();**

**dout.writeUTF(str2); dout.flush();**

**}**

**din.close();**

**s.close();**

**ss.close();**

**}**

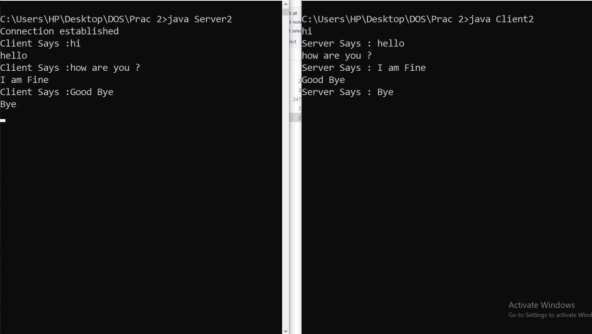
**}**

**Client Side Code:-**

**import java.net.\*; import java.io.\*; public class Client {**

**public static void main(String[] args) throws Exception { Socket s=new Socket(&quot;localhost&quot;,8088);**

**DataInputStream din=new DataInputStream(s.getInputStream()); DataOutputStream dout=new DataOutputStream(s.getOutputStream());BufferedReader br=new BufferedReader(new**



**InputStreamReader(System.in));**

**String str=&quot;&quot;,str2=&quot;&quot;; while(!str.equals(&quot;stop&quot;)){ System.out.println(&quot;\nEnter Response: &quot;); str=br.readLine();**

**dout.writeUTF(str); dout.flush();**

**System.out.println(&quot;Waiting for Server&#39;s Reply...&quot;); str2=din.readUTF();**

**System.out.println(&quot;Server says: &quot;+str2);**

**}**

**dout.close();**

* 1. **lose();**

**}**

**} OUTPUT:**

**CONCLUSION:**

In this practical we successfully constructed a program to illustrate concept of concurrent client server application.

# Practical No. 3

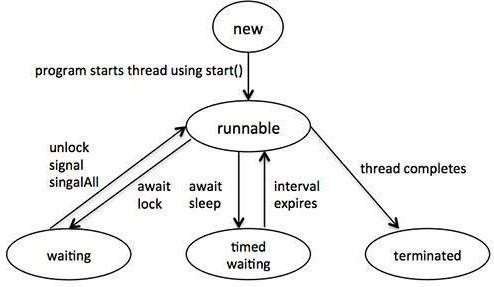
## **Aim:** Develop an application to illustrate the concept of multithreading.

**Practical No.3**

**INLAB AIM:** Develop an application that executes two threads. One thread display “HELLO WORLD” every 1000 milliseconds and another thread display “How Are You” every 2000 milliseconds.

**OBJECTIVES:**

* + - To Study concept of multithreading.
    - To create the concurrent execution for multiple queries at a time.



# Practical No.3

**AIM:** Develop an application to illustrate the concept of multithreading.

### INLAB

**AIM**: Develop an application that executes two threads. One thread display “HELLO WORLD” every 1000 milliseconds and another thread display “How Are You” every 2000 milliseconds.

**OBJECTIVES:**

* + - To Study concept of multithreading.
    - To create the concurrent execution for multiple queries at a time.

**THEORY/ALGORITHM:**

Multithreading in java is a process of executing multiple threads simultaneously. Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

**Life Cycle of a Thread:**

Following figure shows the variuos stages of thread also called as life cycle of thread.

Figure: Lifecycle of thread



A thread goes through various stages in its life cycle. For example, a thread is born, started,runs,and then dies. Following diagram shows complete life cycle of a thread.

**New:** A new thread begins its life cycle in the new state. It remains in this state until theprogram starts the thread. It is also referred to as a born thread.

**Runnable:** After a newly born thread is started, the thread becomes runnable. A thread inthis state is considered to be executing its task.

**Waiting:** Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only whenanother thread signals the waiting thread to continue executing.

**Timed waiting**: A runnable thread can enter the timed waiting state for a specified intervalof time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

**Terminated:** A runnable thread enters the terminated state when it completes its task orotherwise terminates.

**Thread Priorities:**

Every Java thread has a priority that helps the operating system determine the order inwhich threads are scheduled.

Java thread priorities are in the range between MIN\_PRIORITY (a constant of 1) and MAX\_PRIORITY (a constant of 10). By default, every thread is given priority NORM\_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependentant.

**CODE:**

**class Thread1 implements Runnable{ @Override**

**public void run() {**

**// TODO Auto-generated method stub**

**while (true) { try{**

**System.out.println("HELLO WORLD"); Thread.sleep(1000);**

**}catch(Exception e){**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**class Thread2 implements Runnable{ @Override**

**public void run() {**

**// TODO Auto-generated method stub while (true) {**

**try{**

**System.out.println("How Are You"); Thread.sleep(2000);**

**}catch(Exception e){ e.printStackTrace();**

**}**

**}**

**}**

**}**

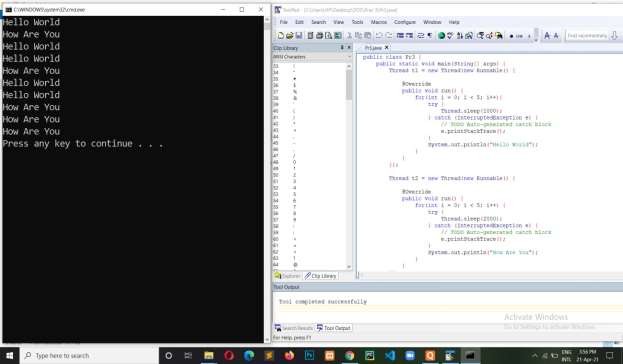
**public class MThread{**

**public static void main(String[] args) { Thread1 thread1 = new Thread1(); Thread2 thread2 = new Thread2(); new Thread(thread1).start();**

**new Thread(thread2).start();**

**}**

**}**



**OUTPUT:**

**CONCLUSION:**

In this practical we developed an application to illustrate the concept of multithreading and hence completed the execution successfully.

# Practical No. 4

## **Aim:** Build a program to demonstrate the concept of logical clock.

**Practical No.4**

**INLAB AIM:** Build a program to demonstrate the concept of logical clock synchronization indistributed environment using Lamport logical clock

**OBJECTIVES:**

* To study clock synchronization issues in distributed environment.
* To know the concept of logical clock.
* To study the need of logical clock.

# Practical No.4

**AIM:** Build a program to demonstrate the concept of Logical Clock.

### INLAB

**AIM**: Construct a program to demonstrate the concept of logical clock synchronization in distributed environment using Lamport logical clock.

**OBJECTIVES:**

* To Study clock synchronization issues in distributed environment.
* To know the concept of logical clock.
* To study the need of logical clock.

**THEORY/ALGORITHM:**

Distributed system is a collection of computer that are interconnected via some communication networks. Basically all the computers in distributed system are physically separated and they may be located apart from each other. Therefore all the process that interactwith each other needs to be synchronized in the context of time to achieve some goal.

There are certain limitations of distributed systems that leave impact on the design of distributed systems

Following are some inherent limitations:

* No global clock available
* No shared Memory

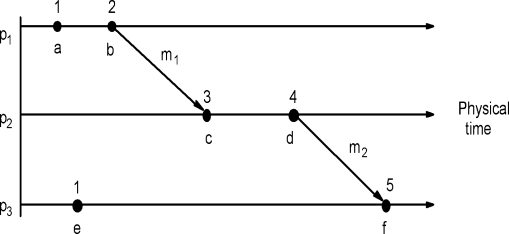
In distributed system, there is no common or global clock available. Since in some situation, the programs on different computers need to coordinate their actions by exchanging message. Due to the unavailability of notion of a global clock there are limits on the accuracy of the output. Fo that purpose temporal ordering of events is required for scheduling processes and it becomes very difficult for a distributed system.

In case of ordering events according to time, Lamport proposed a scheme using logical

clocks.

**Lamport’s Clock:**

Logical time and logical clocks:



A logical clock is a monotonically increasing software counter. It need not relate to aphysical clock.

Each process pi has a logical clock, Li which can be used to apply logical timestamps to

events.

* LC1: Li is incremented by 1 before each event at process pi
* LC2:
* (a) when process pi sends message m, it piggybacks t = Li
* (b) when pj receives (m,t) it sets Lj := max(Lj, t) and applies LC1 before timestamping the event receive (m)

**CODE:**

**import java.util.Scanner; import java.io.File;**

**import java.io.FileNotFoundException; import java.io.FileOutputStream; import java.io.PrintStream;**

**public class Lamport{ static int ev[]=new int[25];**

**static int lc[][]= new int[5][25]; static int p,slocation;**

**static String evin[][]=new String[5][25];**

**public static void main(String args[]) throws FileNotFoundException**

**{**

**int i,j,rseq,lctemp;**

**Scanner sc = new Scanner(System.in); PrintStream console = System.out; File file = new File("a1out.txt");**

**FileOutputStream fos = new FileOutputStream(file); PrintStream ps = new PrintStream(fos); System.out.println("Enter the number of process:"); p=sc.nextInt();**

**System.out.println("Enter the no of events per process:"); for(i=1;i<=p;i++)**

**{**

**ev[i]=sc.nextInt();**

**}**

**System.out.println("Enter internal events as chars other than 's'**

**and 'r' unless it's a send or receive:"); for(i=1;i<=p;i++)**

**{**

**System.out.println("For process:"+i); for(j=1;j<=ev[i];j++)**

**{**

**System.out.println("For event:"+j); evin[i][j]=sc.next();**

**lc[i][j]=-2; //Initialized all values of logical clocks as -2**

**for validation purpose**

**}**

**}**

**// print input entered by user System.out.println("Below is the entered Input:"); for(i=1;i<=p;i++)**

**{**

**System.out.print("P"+i+" : "); for(j=1;j<=ev[i];j++)**

**{**

**System.out.print(evin[i][j]+" ");**

**}**

**System.out.println();**

**}**

**// Initialization of Main Logic of Logical Clock for(i=1;i<=p;i++)**

**{**

**for(j=1;j<=ev[i];j++)**

**{**

**if((j==1)&&(evin[i][j].charAt(0)!='r'))**

**lc[i][j]=1;**

**else if(evin[i][j].charAt(0)!='r')**

**{**

**lc[i][j]=lc[i][j-1]+1;**

**}**

**else**

**{**

**rseq=Character.getNumericValue(evin[i][j].charAt(1));**

**lctemp=findlcs(rseq); if (lctemp==-5)**

**System.out.println("There is some problem in**

**lctemp value -5??");**

**if(lctemp<lc[i][j-1])**

**lc[i][j]=lc[i][j-1]+1;**

**else**

**}**

**}**

**}**

**lc[i][j]=lctemp+1;**

**//It will print logical clock value System.setOut(ps); for(i=1;i<=p;i++)**

**{**

**System.out.print("P"+i+" : "); for(j=1;j<=ev[i];j++)**

**{**

**System.out.print(lc[i][j]+" ");**

**}**

**System.out.println();**

**}**

**System.setOut(console); System.out.println();**

**//Print final output on console**

**System.out.println("Logical clock value for the above input is as below"); for(i=1;i<=p;i++)**

**{**

**System.out.print("P"+i+" : "); for(j=1;j<=ev[i];j++)**

**{**

**System.out.print(lc[i][j]+" ");**

**}**

**System.out.println();**

**}**

**sc.close();**

**}**

**//Recursive function which finds logical clock of the Send event static int findlcs(int rseq)**

**{**

**int i,j,slc=-5; //slog : Logical clock value of corresponding s for(i=1;i<=p;i++)//find lc(s) matches -2 then pass process id to**

**logclock() to calculate the value of of s**

**{**

**for(j=1;j<=ev[i];j++)**

**{**

**if(evin[i][j].charAt(0)=='s'&& Character.getNumericValue(evin[i][j].charAt(1))==rseq)**

**{**

**if(lc[i][j]!=-2)**

**return lc[i][j]; //return the corresponding sender logical clock**

**else**

**{**

**slocation=j; slc=logclock(i);**

**}**

**}**

**}**

**}**

**return slc;**

**}**

**static int logclock(int pr)**

**{**

**int j,rseq,lctemp,slc=-1; for(j=1;j<=ev[pr];j++)**

**{**

**if((j==1)&&(evin[pr][j].charAt(0)!='r'))**

**{**

**lc[pr][j]=1;**

**}**

**else if(evin[pr][j].charAt(0)!='r')**

**{**

**}**

**else**

**{**

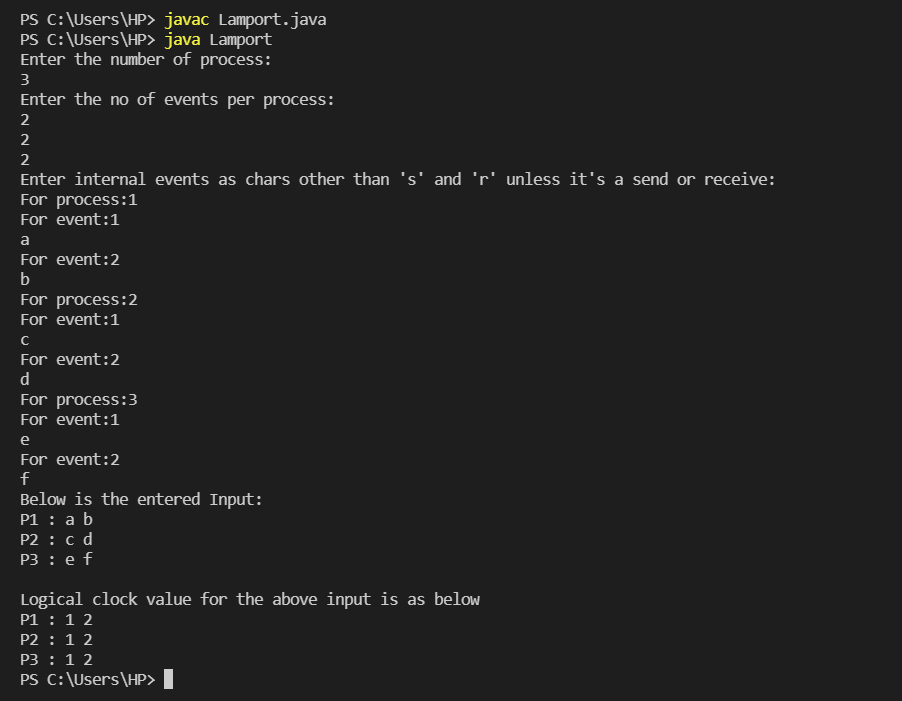
**lc[pr][j]=lc[pr][j-1]+1;**

**rseq=Character.getNumericValue(evin[pr][j].charAt(1)); lctemp=findlcs(rseq);**

**if(lctemp<lc[pr][j-1])**

**lc[pr][j]=lc[pr][j-1]+1;**

**else**



**lc[pr][j]=lctemp+1;**

**}**

**if(j==slocation && lc[pr][j]!=-2)**

**{**

**return lc[pr][j];**

**}**

**}**

**return slc;**

**}**

**}**

**OUTPUT:**

**CONCLUSION:**

In this practical we build a program to demonstrate the concept of Logical Clock and hence successfully completed its execution.

# Practical No. 5

## **Aim:** Construct a distributed application using the concept ofRemote method invocation (RMI).



**Practical No.5**

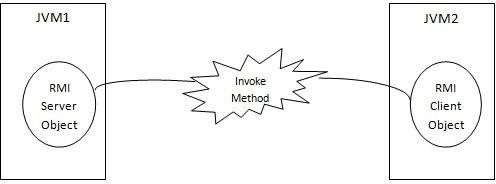
**INLAB AIM:** Construct a distributed application using remote method invocation (RMI) whereclient submit two strings to server and returns concatenation of given string.

**OBJECTIVES:**

To study RMI concepts.

To know how to invoke a method running in differ JVM from another JVM. To study how to write an application using RMI methodology.

# Practical No.5



**AIM:** Construct a distributed application using the concept of remote method invocation (RMI)

### INLAB

**AIM**: Construct a distributed application using remote method invocation (RMI) where clientsubmit two strings to server and returns concatenation of given string.

**OBJECTIVES:**

To study RMI concepts.

To know how to invoke a method running in differ JVM from another JVM. To study how to write an application using RMI methodology.

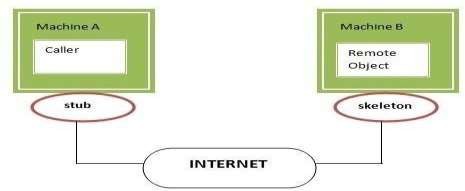
**THEORY/ALGORITHM:**

The main objective of RMI is to provide the facility of invoking method on server. This is done bycreating a RMI Client and RMI Server. RMI Client invokes a method defined on the RMI Server. **RMI terminology:**

Following figure shows the terminology of Remote Method Invocation.

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

RMI is used to communicate between two running java applications on different JVM (JavaVirtual Machines). The main motive behind RMI implementation is to invoke one method running on different JVM. The JVM Application, in which an invoked method is running out, is called RMI Server, where as the invoking application running on the different JVM is called RMI Client.



**RMI Client**: It is an object that invokes remote methods on an RMI Server.

**Stub**: A stub is proxy that stands for RMI Server on client side and handles remote method invocation on behalf of RMI Client.

**Skeleton**: It is a proxy that stands for RMI client on server side and handles remote method invocation on RMI Server on behalf of client.

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

Figure: Role of stub and skeleton in RMI

**Algorithm:**

**Step 1: Define Remote Interface**

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

**Step 2: Implementation of remote interface**

For implementation of remote interface, a class must either extend UnicastRemoteObject or use exportObject() method of UnicastRemoteObject class.

**Step 3: Create AddServer and host rmi service**

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

**Step 4: Create client application**

Client application contains a java program that invokes the lookup() method of the Naming class. This method accepts one argument, the rmi URL and returns a reference to an object of type

AddServerInterface. All remote method invocation is done on this object.

**CODE:**

**Calculator Code:**

**import java.rmi.\*;**

**interface Calculator extends Remote{**

**public int getAddition(int a, int b)throws RemoteException;**

**}**

**Calculator Impl Code:**

**import java.rmi.\*; import java.rmi.server.\*;**

**class CalculatorImpl extends UnicastRemoteObject implements Calculator{ CalculatorImpl()throws RemoteException{}**

**@Override**

**public int getAddition(int a, int b) throws RemoteException {**

**// TODO Auto-generated method stub return a + b;**

**}**

**}**

**Client Code:**

**import java.rmi.\*; import java.util.Scanner; public class MyClient{**

**public static void main(String args[])throws Exception{**

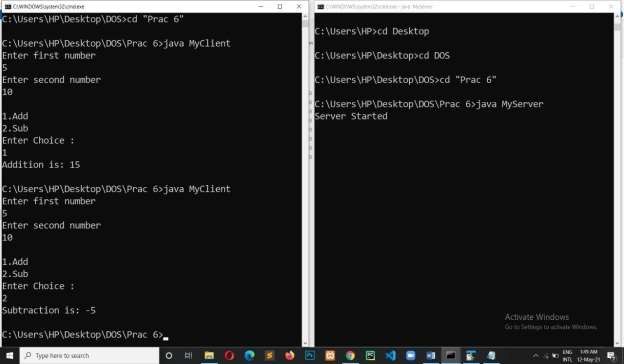
**Calculator c=(Calculator)Naming.lookup("rmi://localhost:6666/calc"); Scanner sc = new Scanner(System.in);**

**System.out.println("Enter first number"); int a = sc.nextInt(); System.out.println("Enter second number"); int b = sc.nextInt();**

**System.out.println("Addition is: "+c.getAddition(a, b)); sc.close();**

**}**

**}**



**Server Code:**

**import java.rmi.\*;**

**import java.rmi.registry.LocateRegistry; public class MyServer{**

**public static void main(String args[])throws Exception{ Remote r = new CalculatorImpl(); LocateRegistry.createRegistry(6666); Naming.rebind("rmi://localhost:6666/calc",r); System.out.println("Server Started");**

**}**

**}**

**OUTPUT:**

**CONCLUSION:**

In this practical we constructed a distributed application using the concept of remote method invocation (RMI) and hence completed the execution successfully.

# Practical No. 6

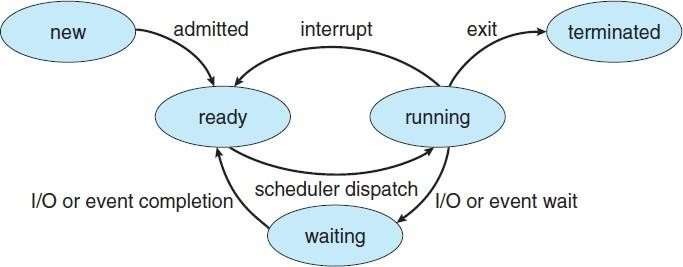
## **Aim:** Construct a program to implement concept of IPC

**Practical No.6**

**INLAB AIM:** Construct a program to implement the concept of IPC using pipe.

**OBJECTIVES:**

* + To Study concept of multiprocessing.
  + To create the concurrent execution for multiple queries at a time.



# Practical No.6

**AIM:** Construct a program to implement concept of IPC.

### INLAB

**AIM**: Construct a program to implement the concept of IPC using pipe.

**OBJECTIVES:**

* + To Study concept of multiprocessing.
  + To create the concurrent execution for multiple queries at a time.

**THEORY/ALGORITHM:**

Multitasking in java is a process of executing multiple processes simultaneously. Process is basically a program under execution. During Multitasking no. of processes will execute simultaneously, they may interact with one another during execution. The inter process communication is an example of multitasking with interaction.

**Life Cycle of a process:**

Following figure shows the variuos stages of thread also called as life cycle of process.

Figure: Lifecycle of process



A process goes through various stages in its life cycle. For example, a process is born, started,runs, and then dies. Following diagram shows complete life cycle of a process.

**New:** A new process begins its life cycle in the new state. It remains in this state untilthe program starts. It is also referred to as a born process.

**Runnable:** After a newly born process is started, the process becomes runnable. Aprocess in this state is considered to be executing.

**Waiting:** Sometimes, a process transitions to the waiting state while the process waitsfor another process to perform a task. A process transitions back to the runnable state only when another process signals the waiting thread to continue executing.

**Timed waiting**: A process can enter the timed waiting state for a specified interval of time. A process in this state transitions back to the runnable state when that time intervalexpires or when the event it is waiting for occurs.

**Terminated:** A runnable process enters the terminated state when it completes its taskor otherwise terminates.

**CODE:**

**Process1:**

**import java.io.PipedOutputStream;**

**public class Process1 implements Runnable { final PipedOutputStream output; Process1(PipedOutputStream output){**

**this.output = output; System.out.println("Process 1 Instantiated");**

**}**

**@Override**

**public void run() {**

**// TODO Auto-generated method stub try {**

**output.write("Hello world, pipe!".getBytes());**

**} catch (Exception e) {**

**//e.printStackTrace();**

**}**

**Process2:**

**import java.io.PipedOutputStream;**

**public class Process1 implements Runnable { final PipedOutputStream output; Process1(PipedOutputStream output){**

**this.output = output; System.out.println("Process 1 Instantiated");**

**}**

**@Override**

**public void run() {**

**// TODO Auto-generated method stub try {**

**output.write("Hello world, pipe!".getBytes());**

**} catch (Exception e) {**

**//e.printStackTrace();**

**}**

**}**

**}**

**Pipe Example:**

**import java.io.IOException; import java.io.PipedInputStream;**

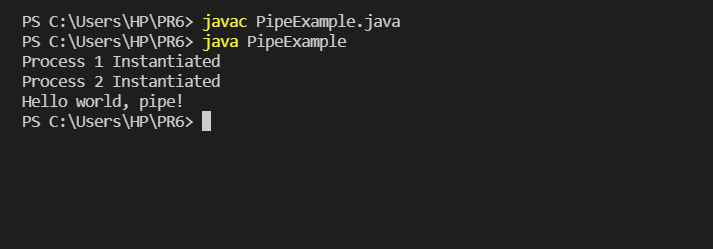
**import java.io.PipedOutputStream; public class PipeExample {**

**public static void main(String[] args) throws IOException {**

**PipedOutputStream output = new PipedOutputStream(); PipedInputStream input = new PipedInputStream(output);**

**Process1 p1 = new Process1(output); Process2 p2 = new Process2(input); new Thread(p1).start();**

**new Thread(p2).start();**



**OUTPUT:**

**CONCLUSION:**

In this practical we constructed a program to implement the concept of IPC and hence completed the execution successfully.

# Practical No. 7

## **Aim:** Build a program to demonstrate concept of distributed mutual exclusion.



**INLAB AIM:** Build a program to implement Suzuki Kasami algorithm.

**OBJECTIVES:**

To Study the concept of mutual exclusion in distributed environment. To study the working of Suzuki Kasami algorithm



**AIM:** Build a program to demonstrate concept of distributed mutual exclusion.

**INLAB**

**AIM**: Build a program to implement Suzuki Kasami algorithm

**OBJECTIVES:**

To Study the concept of mutual exclusion in distributed environment. To study the working of Suzuki Kasami algorithm

**THEORY/ALGORITHM:**

**Mutual exclusion:**

Concurrent access of processes to a shared resource or data is executed in mutuallyexclusive manner.

Only one process is allowed to execute the critical section (CS) at any given time. In a distributed system, shared variables (semaphores) or a local kernel cannot be used toimplement mutual exclusion.

Three basic approaches for distributed mutual exclusion:

1. Token based approach
2. Non-token based approach
3. Quorum based approach

**Token-based approach:**

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

**Suzuki Kasami’s broadcast algorithm**

Suzuki Kasami’s is token based mutual exclusion algorithm which can be used to achievemutual exclusion in distributed environment.

**Overview:**



If a process wants to enter the critical section, and it does not have the token, it broadcastsa request message to all other processes in the system

The processes that has the token will then send it to the requesting process However, if it is in CS, it gets to finish before sending the token

A process holding the token can continuously enter the critical section until the token isrequested

Request vector at process i:

RNi[k] contains the largest sequence number received from process k in a requestmessage

Token consists of vector and a queue:

LN[k] contains the sequence number of the latest executed request from process k Q is the queue of requesting process

**Requesting the critical section (CS):**

When a process i wants to enter the CS, if it does not have the token, it:

- Increments its sequence number Rni[i].

-Sends a request message containing new sequence number to all processes in thesystem.

When a process k receives the request(i,sn) message, it:

- Sets RNk[i] to MAX(RNk[i], sn).

-If sn < Rnk[i], the message is outdated.

If process k has the token and is not in CS (i.e., is not using token), and if RNk[i] ==LN[i]+1 (indicating an outstanding request) it sends the token to process i.

**Executing the CS:**

**-**A process enters the CS when it has acquired the token.

**Releasing the CS:**

When a process i leaves the CS, it:

-Sets LN[i] of the token equal to Rni[i].

-Indicates that its request RN i[i] has been executed.



For every process k whose ID is not in the token queue Q, it appends its ID to Q if Rni[ k]

== LN[k]+1

-Indicates that process k has an outstanding request

If the token queue Q is nonempty after this update, it deletes the process ID at the head of Qand sends the token to that process

-Gives priority to others’ requests

-Otherwise, it keeps the token

**Evaluation:**

0 to N messages required to enter CS No messages if process holds the token Otherwise N 1 requests, 1 reply

**CODE:**

**Echo Server: 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**

**{**

**BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream())); while(true)**

**{**

**System.out.println(in.readLine());**

**}**

**}**

**catch(Exception e)**

**{**

**}**

**}**

**}**

**Echo Client1:**

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

**public class EchoClientOne**

**{**

**public static void main(String args[]) throws IOException**

**{**

**Socket s=new Socket("localhost",7000);**

**PrintStream out=new PrintStream(s.getOutputStream()); ServerSocket ss = new ServerSocket(7001);**

**Socket s1 = ss.accept();**

**BufferedReader in1 = new BufferedReader(new InputStreamReader(s1.getInputStream()));**

**PrintStream out1 = new PrintStream(s1.getOutputStream());**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); String str="Token";**

**while(true)**

**{**

**if(str.equalsIgnoreCase("Token"))**

**{**

**System.out.println("Do you want to send some data"); System.out.println("Enter Yes or No"); str=br.readLine();**

**if(str.equalsIgnoreCase("Yes"))**

**{**

**System.out.println("Enter the data"); str=br.readLine();**

**out.println(str);**

**}**

**out1.println("Token");**

**}**

**System.out.println("Waiting for Token"); str=in1.readLine();**

**}**

**}**

**}**

**Echo Client2:**

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

**public class EchoClientTwo**

**{**

**public static void main(String args[])throws IOException**

**{**

**Socket s=new Socket("localhost",7000);**

**PrintStream out = new PrintStream(s.getOutputStream()); Socket s2=new Socket("localhost",7001);**

**BufferedReader in2 = new BufferedReader(new InputStreamReader(s2.getInputStream()));**

**PrintStream out2 = new PrintStream(s2.getOutputStream());**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); String str;**

**while(true)**

**{**

**System.out.println("Waiting for Token"); str=in2.readLine(); if(str.equalsIgnoreCase("Token"))**

**{**

**System.out.println("Do you want to send some data"); System.out.println("Enter Yes or No"); str=br.readLine();**

**if(str.equalsIgnoreCase("Yes"))**

**{**

**System.out.println("Enter the data"); str=br.readLine();**

**out.println(str);**

**}**

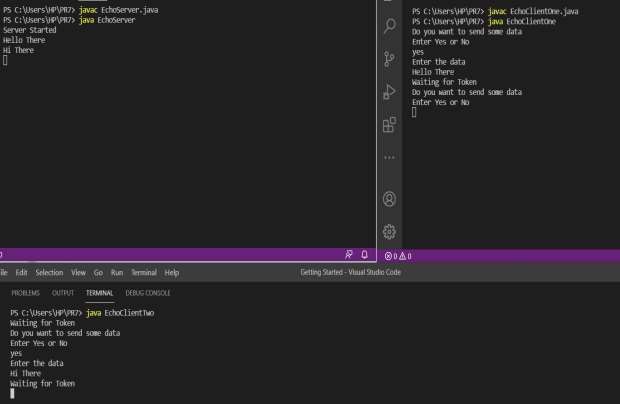
**out2.println("Token");**

**}**

**}**

**}**

**}**



**OUTPUT:**

**CONCLUSION:**

In this practical we built a program to demonstrate concept of distributed mutual exclusion and hence completed its execution successfully.

# Practical No. 8

## **Aim:** Implement a program to illustrate the concept of deadlock detection



**Practical No. 8**

**INLAB AIM:** Write a program to implement centralize deadlock detection algorithm.

**OBJECTIVES:**

To know the concept of deadlock.

To interpret the process of detecting deadlock.



# Practical No. 8

**AIM:** Implement a program to illustrate the concept of deadlock detection

**INLAB AIM**: Write a program to implement centralize deadlock detection algorithm.

**OBJECTIVES:**

To know the concept of deadlock.

To interpret the process of detecting deadlock.

**THEORY/ALGORITHM:**

A *deadlock* is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasingto function.

We can use a centralized deadlock detection algorithm and try to imitate the non- distributed algorithm. Although each machine maintains the resource graph for its own processes and resources, a central coordinator maintains the resource graph for the entire system. When the coordinator detects a cycle, it kills off one process to break the deadlock.

Unlike the centralized case, where all the information is automatically available in the right place, in a distributed system it has to be sent there explicitly. Each machine maintains the graph for its own processes and resources. Several possibilities exist for getting it there. First, whenever an arc is added or deleted from the resource graph, a message can be sent to the coordinator providing the update. Second, periodically, every process can send a list of arcs added or deleted since the previous update. This method requires fewer messages than the first one. Third, the coordinator can ask for information when it needs it.

**CODE:**

**import java.util.Scanner; class DeadLock{**

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

**Scanner sc = new Scanner(System.in); int found,flag=0,l,tp,tr,i,j,k=1,sum=0; int p[][] = new int[4][5];**

**int c[][] = new int[4][5]; int m[]=new int[5];**

**int r[]=new int[5]; int a[]= new int[5];**

**int temp[]=new int[5];**

**System.out.println("Enter total no of processes"); tp = sc.nextInt();**

**System.out.println("Enter total no of resources"); tr = sc.nextInt();**

**System.out.println("Enter claim (Max. Need) matrix\n"); for(i=0;i<tp;i++)**

**{**

**System.out.println("process: \n"+i); for(j=0;j<tr;j++){**

**c[i][j] = sc.nextInt();**

**}**

**}**

**System.out.println("Enter allocation matrix\n"); for(i=0;i<tp;i++)**

**{**

**System.out.println("process:\n"+i); for(j=0;j<tr;j++){**

**p[i][j] = sc.nextInt();**

**}**

**}**

**System.out.println("Enter resource vector (Total resources):\n"); for(i=0;i<tr;i++)**

**{**

**r[i] = sc.nextInt();**

**}**

**System.out.println("Enter availability vector (available resources):\n"); for(i=0;i<tr;i++)**

**{**

**a[i] = sc.nextInt();**

**temp[i]=a[i];**

**}**

**for(i=0;i<tp;i++)**

**{**

**sum=0; for(j=0;j<tr;j++)**

**{**

**sum+=p[i][j];**

**}**

**if(sum==0)**

**{**

**m[k]=i; k++;**

**}**

**}**

**for(i=0;i<tp;i++)**

**{**

**for(l=0;l<k;l++) if(i!=m[l])**

**{**

**flag=1; for(j=0;j<tr;j++) if(c[i][j]<temp[j])**

**{**

**flag=0; break;**

**}**

**}**

**if(flag==1)**

**{**

**m[k]=i; k++;**

**for(j=0;j<tr;j++) temp[j]+=p[i][j];**

**}**

**}**

**System.out.println("deadlock causing processes are:"); for(j=0;j<tp;j++)**

**{**

**found=0; for(i=0;i<k;i++)**

**{**

**if(j==m[i]) found=1;**

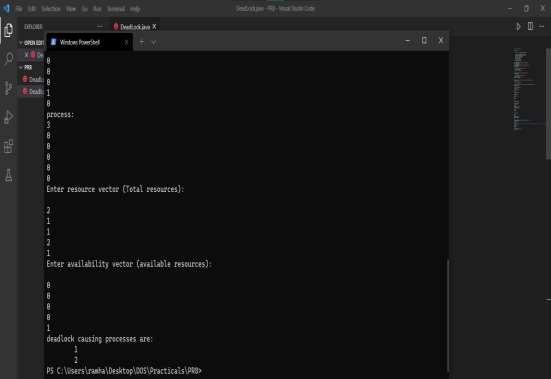
**}**

**if(found==0) System.out.println("\t"+j);**

**}**

**}**

**}**



**OUTPUT:**

**CONCLUSION:**

In this practical we implemented a program to illustrate the concept of deadlock detection and hence completed the execution successfully.

# Practical No. 9

## **Aim:** Build a program to illustrate the concept of distributeddeadlock detection



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

**OBJECTIVES:**

To interpret the concept of distributed deadlock. To know the process of detecting deadlock.



**AIM:** Build a program to illustrate the concept of distributed deadlock detection.

**INLAB**

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

**OBJECTIVES:**

To interpret the concept of distributed deadlock. To know the process of detecting deadlock.

**THEORY/ALGORITHM:**

A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasingto function.

The same conditions for deadlock in uniprocessors apply to distributed systems. Unfortunately, as in many other aspects of distributed systems, they are harder to detect, avoid, andprevent.

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

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

**Edge Chasing deadlock detection algorithm:**

if *Pi* is locally dependent on

itselfthen declare a deadlock else for all *Pj* and *Pk* such that

1 *Pi* is locally dependent upon *Pj* , and2 *Pj* is waiting on *Pk* , and

3 *Pj* and *Pk* are on different sites,send a probe (i, j, k) to the home site of *Pk*

**CODE:**

**Client:**

**import java.io.File;**

**import java.nio.CharBuffer;**

**import java.nio.MappedByteBuffer; import java.nio.channels.FileChannel;**

**import java.nio.channels.FileChannel.MapMode; import java.nio.file.StandardOpenOption;**

**public class Client {**

**public static void main( String[] args ) throws Throwable { File f = new File("file.txt");**

**FileChannel channel = FileChannel.open( f.toPath(), StandardOpenOption.READ, StandardOpenOption.WRITE, StandardOpenOption.CREATE );**

**MappedByteBuffer b = channel.map( MapMode.READ\_WRITE, 0, 4096 ); CharBuffer charBuf = b.asCharBuffer();**

**System.out.println("File name: "+f.getName()); System.out.println("Reading from file. ..");**

**// Prints 'Hello server' char c;**

**while( ( c = charBuf.get() ) != 0 ) { System.out.print( c );**

**}**

**System.out.println();**

**charBuf.put( 0, '\0' );**

**}**

**}**

**Server:**

**import java.io.File;**

**import java.nio.CharBuffer;**

**import java.nio.MappedByteBuffer; import java.nio.channels.FileChannel;**

**import java.nio.channels.FileChannel.MapMode; import java.nio.file.StandardOpenOption;**

**public class Server {**

**public static void main( String[] args ) throws Throwable { File f = new File("file.txt");**

**FileChannel channel = FileChannel.open( f.toPath(), StandardOpenOption.READ, StandardOpenOption.WRITE, StandardOpenOption.CREATE );**

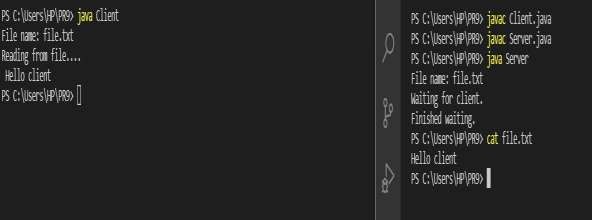
**MappedByteBuffer b = channel.map( MapMode.READ\_WRITE, 0, 4096 ); CharBuffer charBuf = b.asCharBuffer();**

**System.out.println("File name: "+f.getName()); char[] string = " Hello client\0".toCharArray(); charBuf.put( string );**

**System.out.println( "Waiting for client." ); while( charBuf.get( 0 ) != '\0' ); System.out.println( "Finished waiting." );**

**}**

**}**



**OUTPUT:**

**CONCLUSION:**

In this practical we built a program to illustrate the concept of distributed deadlock detection and hence completed the execution successfully.

# Practical No. 10

## **Aim:** Construct a program to implement commit protocol

**Practical No. 10**

**INLAB AIM:** Construct a program to implement two phase commit protocol.

**OBJECTIVES:**

* To know the concept of commit Protocol.
* To interpret the role of two phase commit protocol in distributed application.
* To recognize working of two phase commit protocol.

# Practical No. 10

**AIM:** Construct a program implement commit protocol.

### INLAB

**AIM**: Construct a program to implement the two-phase commit protocol.

**OBJECTIVES:**

* To know the concept of commit Protocol.
* To interpret the role of two phase commit protocol in designing distributed application.
* To recognize working of two phase commit protocol.

**THEORY/ALGORITHM:**

The two phase commit protocol is a distributed algorithm which lets all sites in a distributed system agree to [commi](http://courses.cs.vt.edu/~cs5204/fall00/distributedDBMS/duckett/tpcp.html)t a transaction. The protocol results in either all nodes committing the [transaction](http://courses.cs.vt.edu/~cs5204/fall00/distributedDBMS/duckett/tpcp.html) or aborting, even in the case of site failures and message losses.However, due to the work by Skeen and Stonebraker, the protocol will not handle more than one random site failure at a time. The two phases of the algorithm are broken into the COMMIT- REQUEST phase, where the COORDINATOR attempts to prepare all the COHORTS, and the COMMIT phase, where the COORDINATOR completes the transactions at all COHORTS.

**Two phase commit protocol**

It works in two phase as given below.

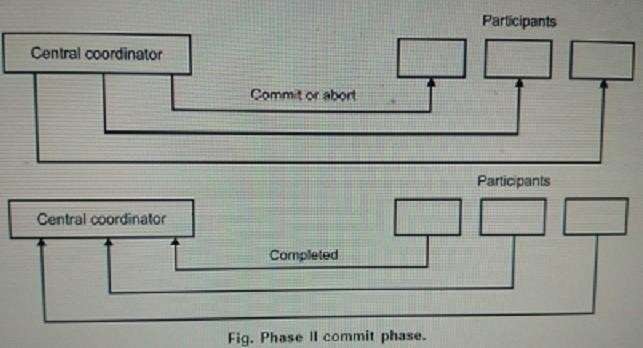
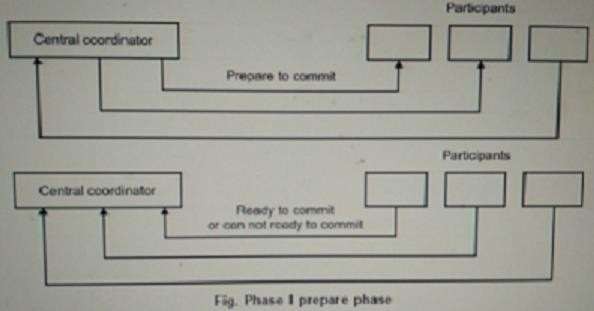
**- Phase 1 (Prepare Phase /Voting Phase)**

* In this phase coordinator sends a prepare to commit message to all participants of transaction.

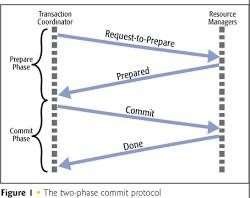
In response to that each participant sends a ready to commit or can not ready to commit messageback to coordinator.

**Phase 2 (Commit / Decision Phase)**

If coordinator receives a ready to commit reply from all participants in phase I it send abort message to all participants.



Assuming that coordinator has sent commit message, all participants now commit transaction and send complete message to coordinator. If any participant fail during commit then participant send abort message to coordinator. If coordinator receive at least one abort it ask all participants to abort otherwise whole transaction is successful.



**CODE:**

**Client:**

**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\nCommited... "); 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**

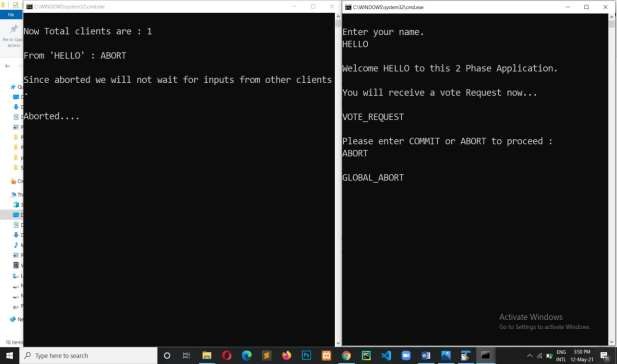
**}//while ser.closed=true; clientSocket.close();**

**}**

**catch(IOException e)**

**{**

**};}}**



**OUTPUT:**

**CONCLUSION:**

In this practical we constructed a program to implement commit protocol and hence completed the execution successfully.

# Practical No. 11

## **Aim:** Case study on Supply Chain Management.

**Practical No. 11**

**INLAB AIM:** Write a case study on Supply Chain Management.

**OBJECTIVES:**

* + To know the concept of supply chain.
  + To be aware about supply chain management.
  + To recognize the utility of supply chain management.

# Practical No. 11

**AIM:** Case study on Supply Chain Management

**INLAB AIM:** Write a case study on Supply Chain Management

**OBJECTIVES:**

* + To know the concept of supply chain.
  + To be aware about supply chain management.
  + To recognize the utility of supply chain management.

**THEORY:**

At the most fundamental level, supply chain management (SCM) is management of the flow of goods, data, and finances related to a product or service, from the procurement of raw materials to the delivery of the product at its final destination. Although many people equate the supply chain with logistics, logistics is actually just one component of the supply chain. Today’s digitally based SCM systems include material handling and software for all parties involved in product or service creation, order fulfilment, and information tracking―such as suppliers, manufacturers, wholesalers, transportation and logistics providers, and retailers.

Supply chain activities span procurement, product lifecycle management, supply chain planning (including inventory planning and the maintenance of enterprise assets and production lines), logistics (including transportation and fleet management), and order management. SCM can also extend to the activities around global trade, such as the management of global suppliers and multinational production processes.

**The history of SCM**

Supply chains have existed since ancient times, beginning with the very first product or service created and sold. With the advent of industrialization, SCM became more sophisticated, allowing companies to do a more efficient job of producing and delivering goods and services. For example, Henry Ford’s standardization of automobile parts was a game-changer that allowed for the mass production of goods to meet the demands of a growing customer base. Over time, incremental changes (such as the invention of computers) have brought additional levels of sophistication to SCM systems. However, for generations, SCM essentially remained a linear, siloed function that was managed by supply chain specialists.

The internet, technology innovation, and the explosion of the demand-driven global economy has changed all that. Today’s supply chain is no longer a linear entity. Rather, it’s a complex collection of disparate networks that can be accessed 24 hours a day. At the center of these networks are consumers expecting their orders to be fulfilled―when they want them, the way they want them.

We now live in a time of unprecedented global business and trade, not to mention continual technology innovation and rapidly changing customer expectations. Today’s best supply chain

strategies call for a demand-driven operating model that can successfully bring people, processes, and technology together around integrated capabilities to deliver goods and services with extraordinary speed and accuracy.

Though SCM has always been an enterprise fundamental, the supply chain today is more vital than ever as a marker for business success. Companies that can effectively manage their supply chain to adapt to today’s volatile and ever-changing, technology-driven business environment are the ones that will survive and thrive.

**SCM and the cloud**

With today’s SCM parameters, the cloud is a natural ally, in part because cloud-based applications are inherently more flexible and adaptable to change. It’s very difficult to adjust on-premises and custom- coded applications in response to the fluctuating circumstances that regularly occur in today’s enterprise environment, such as an unexpected sourcing issue. Cloud solutions are also inherently architected to make better use of the technologies that are becoming pervasive in the Industry 4.0 model. Retrofitting your environment so these technologies can function on legacy applications is both complicated and expensive.

Another significant benefit of integrating the cloud into your SCM system is that you can adopt elements of cloud-based SCM depending on your specific business needs, without undertaking a full-scale migration. Many companies find themselves with a short-term need to rationalize their move to the cloud. The best SCM systems help you extract more value from your current assets and customize your cloud integration to suit your SCM needs, both now and into the future.

***The future of SCM***

The supply chain of the future is all about responsiveness and the customer experience― understood and managed within a network rather than a linear model. Every node of the network must be attuned and flexible to the needs of the consumer while also being capable of addressing factors such as sourcing, trade policies, modes of shipment, and more.

Advanced technology will increasingly be used to improve transparency and visibility throughout this network, as well as to further enable connectivity and SCM utilization. The entire SCM planning function will become more intelligent to take consumer demands into account. The ability to adapt will be a mandate.

In the past, supply chain planning has been a periodic business exercise. Heading into the future, it will be continuous. Future SCM systems will also bring tighter alignment between planning and execution, which is not a current state for most enterprises. The need for speed and accuracy in SCM is only going to increase. Make sure your supply chain is ready for the future by supporting it with an intelligent SCM system.

# Practical No. 12

## **Aim:** Build a program to create web services.



**Practical No. 12**

**INLAB AIM:** Build a program for web service creation and utilization.

**OBJECTIVES:**

To know the concept of web service.

To realize process of web service creation. To recognize the utility of web services.



# Practical No. 12

**AIM:** Build a program to create web services.

### INLAB

**AIM**: Build a program for web service creation and utilization.

**OBJECTIVES:**

To know the concept of web service.

To realize process of web service creation.

To recognize the utility of different web services.

**THEORY/ALGORITHM:**

Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, andsupply chains. These applications can be local, distributed, or web-based. Web services are built ontop of open standards such as TCP/IP, HTTP, Java, HTML, and XML. A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system.

A complete web service is, therefore, any service that −

Is available over the Internet or private (intranet) networks Uses a standardized XML messaging system

Is not tied to any one operating system or programming language Is self-describing via a common XML grammar

Is discoverable via a simple find mechanism

**Characteristics of Web Services:**

XML-Based Loosely Coupled Coarse-Grained

Supports Remote Procedure Calls(RPCs) Supports Document Exchange



**Components of Web Services:**

The basic web services platform is XML + HTTP. All the standard web services work using thefollowing components

SOAP (Simple Object Access Protocol)

UDDI (Universal Description, Discovery and Integration) WSDL (Web Services Description Language)

**How Does a Web Service Work?**

A web service enables communication among various applications by using open standards such asHTML, XML, WSDL, and SOAP. A web service takes the help of −

XML to tag the data

SOAP to transfer a message

WSDL to describe the availability of service.

**CODE:**

**Hello Servelet:**

**import java.io.IOException; import java.io.PrintWriter;**

**import javax.servlet.ServletException; import javax.servlet.http.HttpServlet;**

**import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse;**

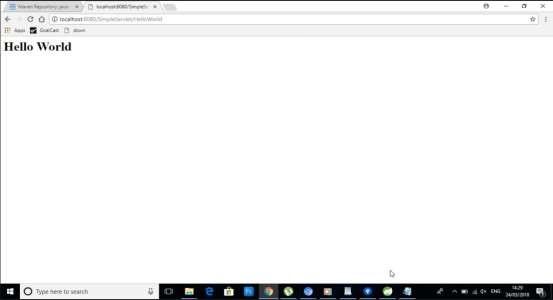
**public class HelloServlet extends HttpServlet{ @Override**

**protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {**

**PrintWriter printWriter=resp.getWriter(); printWriter.append("<h1>Hello World</h1>");**

**}**

**}**



**XML:**

**<?xml version="1.0" encoding="UTF-8"?>**

**<web-app xmlns:xsi=**[**"http://www.w3.org/2001/XMLSchem**](http://www.w3.org/2001/XMLSchema-instance)**a**[**-instance"**](http://www.w3.org/2001/XMLSchema-instance) **xmlns=**[**"http://java.sun.com/xml/ns/javaee"**](http://java.sun.com/xml/ns/javaee) **xsi:schemaLocation=**[**"http://java.sun.com/xml/ns/javaee**](http://java.sun.com/xml/ns/javaee)[**http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd**](http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd)**" version="2.5">**

**<display-name>SimpleServlet</display-name>**

**<welcome-file-list>**

**<welcome-file>index.html</welcome-file>**

**<welcome-file>index.htm</welcome-file>**

**<welcome-file>index.jsp</welcome-file>**

**<welcome-file>default.html</welcome-file>**

**<welcome-file>default.htm</welcome-file>**

**<welcome-file>default.jsp</welcome-file>**

**</welcome-file-list>**

**<servlet>**

**<servlet-name>HelloServlet</servlet-name>**

**<servlet-class>com.ark.simple.servlet.example.HelloServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>HelloServlet</servlet-name>**

**<url-pattern>/HelloServlet</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**OUTPUT:**

**CONCLUSION:**

In this practical we constructed a program to create web services and hence performed the practical successfully.