S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

(An Autonomous Institute, Affiliated to RTMNU, Nagpur)

॥ विद्या धनम् सर्वधनः प्रधानम् ॥

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Vision: To become a center for quality education in the field of Computer Science & Engineering and to create competent professionals.

Session 2020-2021(Even)

Session:	**2020-21 (Even)**
Subject:	Distributed Operating System Practical (DOS)
Year:	IV Year
Semester:	8th Sem
Name of Student:	Rahul Mool
Batch:	B2

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.

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:

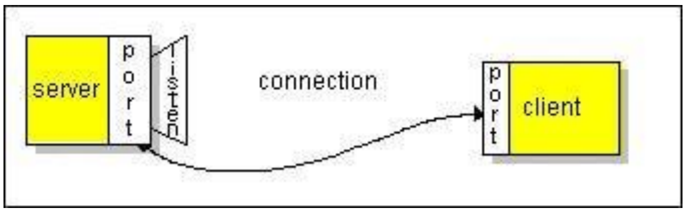
- To know the concept of client server.
- To interpret the process of how communication between client and server is established.
- To know the concept of socket programming in distributed environment.

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.



Following are the steps to create echo server application

1. **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.

2. **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.

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

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

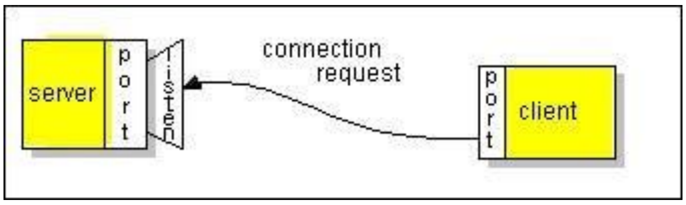
4. **Communicate with the client.**

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

5. **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.



Following are the various steps to create client.

1. **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).

2. **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()));

3. **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);

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

CODE:

<u>EchoClient.java</u>

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

public class EchoClient
{
        public static void main(String[] args)
        {
                try
                {
                        Socket s = new Socket("127.0.0.1", 5000);
                        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)
```

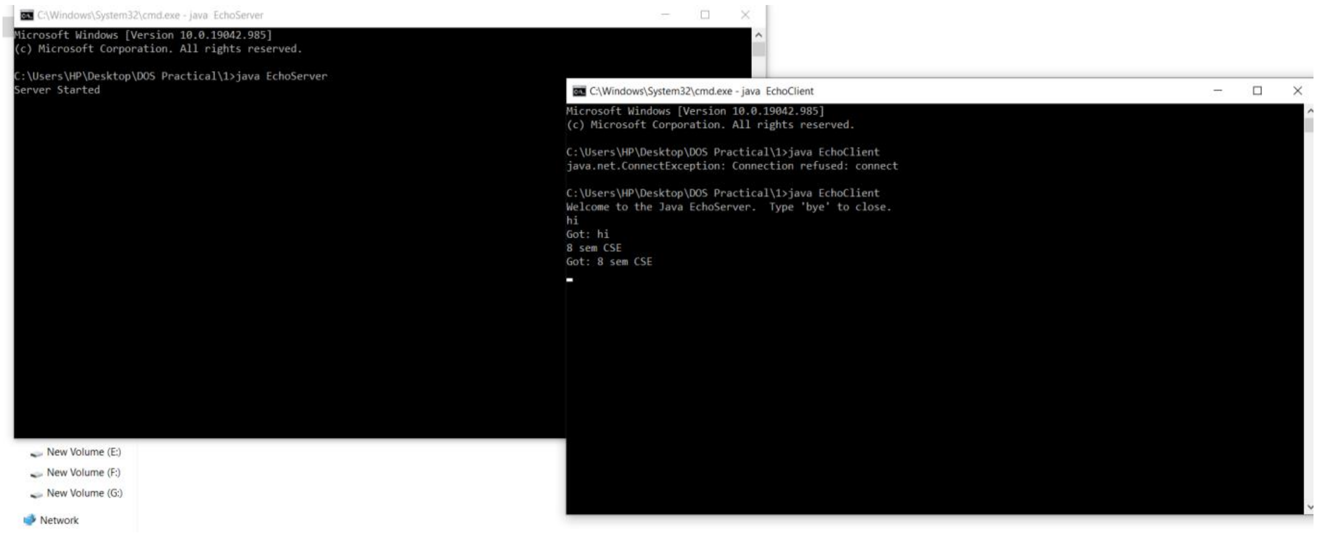
```
		{
			System.err.println(err);
		}
	}
}
```

EchoServer.java

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

public class EchoServer
{
	private ServerSocket server;
	public EchoServer(int portnum)
	{
		try
		{
			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);
				}
```

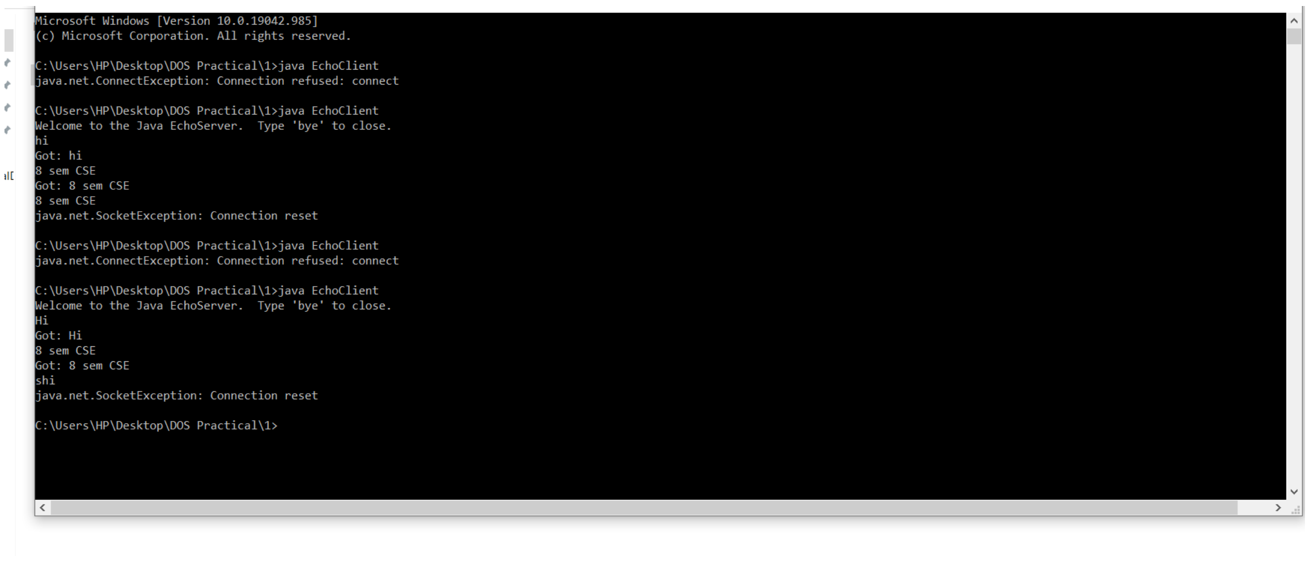
```
                    while ( !line.trim().equals("bye") );
                    client.close();
                }
            }
            catch (Exception err)
            {
                System.err.println(err);
            }
        }

        public static void main(String[] args)
        {
            EchoServer s = new EchoServer(5000);
            s.serve();
        }


    }
```

OUTPUT:



```
C:\Windows\System32\cmd.exe - java  EchoServer
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\1>java EchoServer
Server Started
```

```
C:\Windows\System32\cmd.exe - java  EchoClient
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\1>java EchoClient
java.net.ConnectException: Connection refused: connect

C:\Users\HP\Desktop\DOS Practical\1>java EchoClient
Welcome to the Java EchoServer.  Type 'bye' to close.
hi
Got: hi
8 sem CSE
Got: 8 sem CSE
```



```
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\1>java EchoClient
java.net.ConnectException: Connection refused: connect

C:\Users\HP\Desktop\DOS Practical\1>java EchoClient
Welcome to the Java EchoServer.  Type 'bye' to close.
hi
Got: hi
8 sem CSE
Got: 8 sem CSE
8 sem CSE
java.net.SocketException: Connection reset

C:\Users\HP\Desktop\DOS Practical\1>java EchoClient
java.net.ConnectException: Connection refused: connect

C:\Users\HP\Desktop\DOS Practical\1>java EchoClient
Welcome to the Java EchoServer.  Type 'bye' to close.
Hi
Got: Hi
8 sem CSE
Got: 8 sem CSE
shi
java.net.SocketException: Connection reset

C:\Users\HP\Desktop\DOS Practical\1>
```

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 client server 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 server application.

OBJECTIVES:

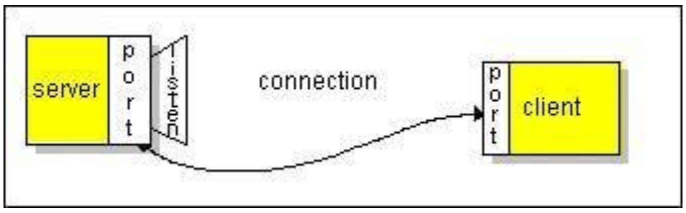
- To know the concept of concurrent client server application.
- To interpret the process of how concurrent communication between client and server is established.
- To know the concept of socket programming in distributed environment.

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 ChatServer 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.



Following are the steps to create echo server application

6. **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.

7. **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.

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

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

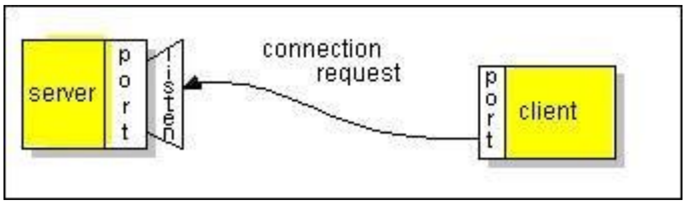
9. **Server Communicate with the client.**

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

10. **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.



Following are the various steps to create client.

5. **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).

6. **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()));

7. **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);

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

CODE:

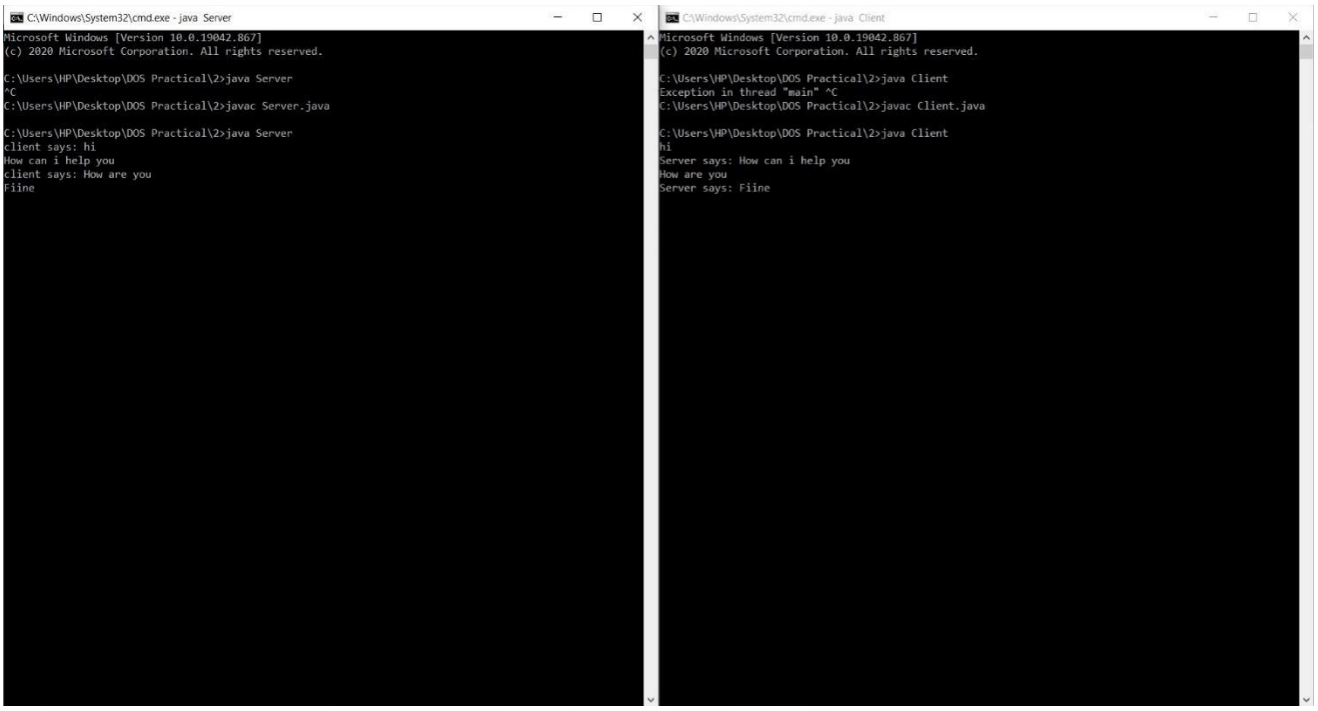
Client.java

```
import java.net.*;
import java.io.*;
class Client{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}}
```

Server.java

```
import java.net.*;
import java.io.*;
class Server{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
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="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

OUTPUT:


C:\Windows\System32\cmd.exe - java Server
C:\Windows\System32\cmd.exe - java Client

```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\2>java Server
^C
C:\Users\HP\Desktop\DOS Practical\2>javac Server.java

C:\Users\HP\Desktop\DOS Practical\2>java Server
client says: hi
How can i help you
client says: How are you
Fiine
```

```
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\2>java Client
Exception in thread "main" ^C
C:\Users\HP\Desktop\DOS Practical\2>javac Client.java

C:\Users\HP\Desktop\DOS Practical\2>java Client
hi
Server says: How can i help you
How are you
Server says: Fiine
```

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.

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.

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.

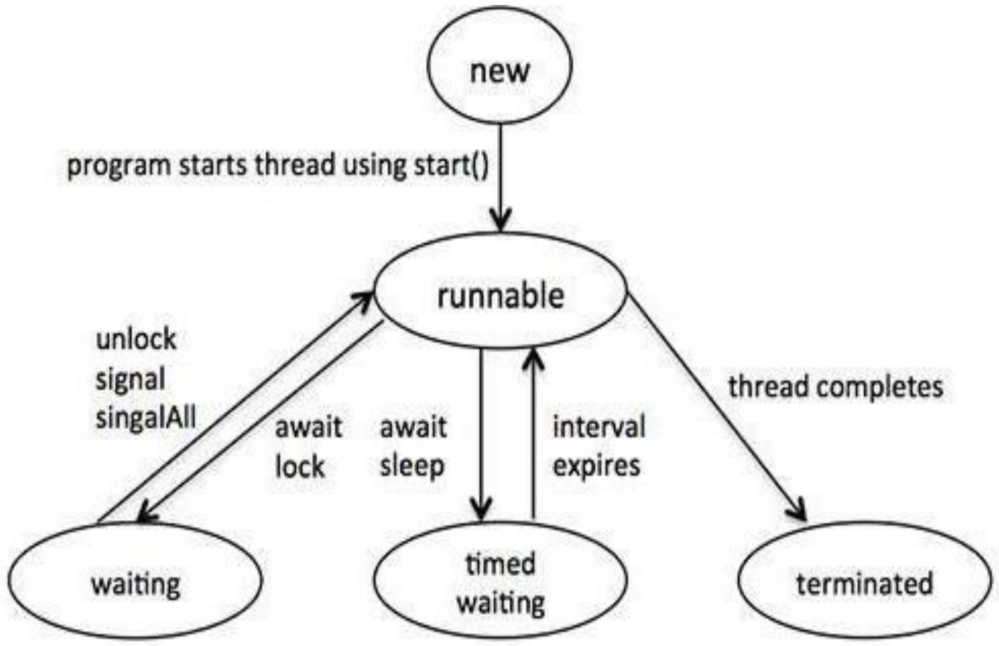


Figure: Lifecycle of thread

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 the program 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 in this 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 when another thread signals the waiting thread to continue executing.

- **Timed waiting**: A runnable thread can enter the timed waiting state for a specified interval of 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 or otherwise terminates.

Thread Priorities:

- Every Java thread has a priority that helps the operating system determine the order in which 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).

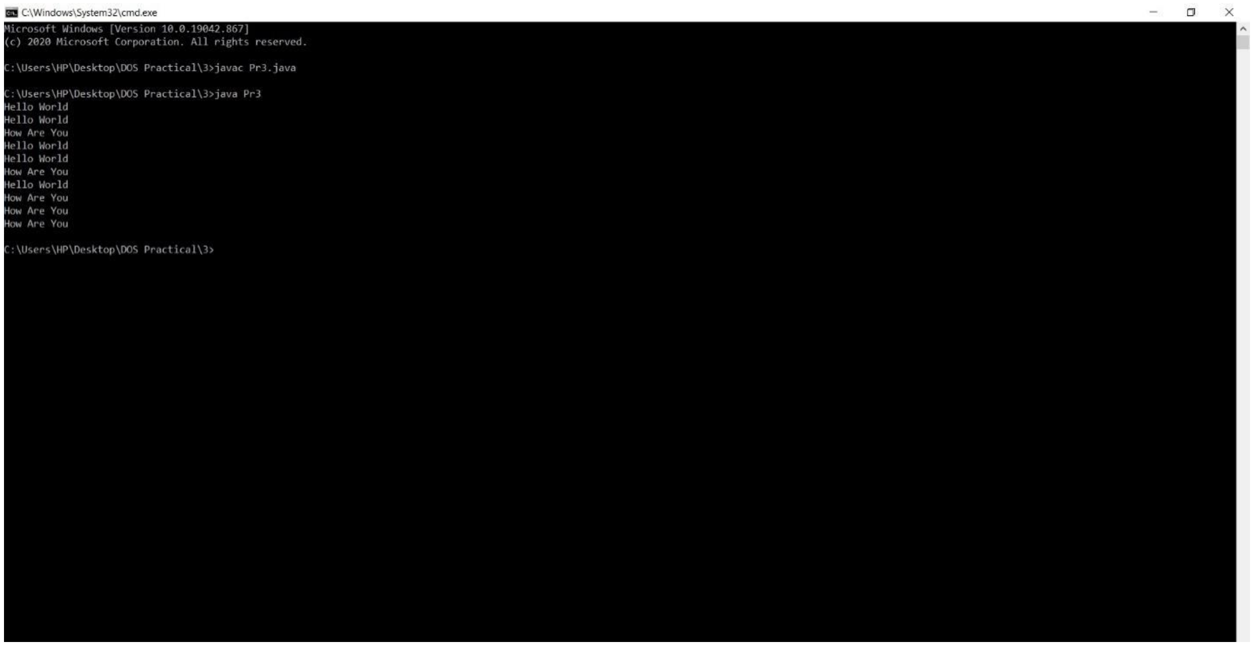
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependentant.

CODE:

<u>Pr3.java</u>

```
public class Pr3 {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {

            @Override
            public void run() {
                for(int i = 0; i < 5; i++){
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                System.out.println("Hello World");
            }
        }
    });

    Thread t2 = new Thread(new Runnable() {

        @Override
        public void run() {
            for(int i = 0; i < 5; i++) {
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                System.out.println("How Are You");
            }
        }
    });

    t1.start();
    t2.start();
  }
}
```

OUTPUT:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\3>javac Pr3.java

C:\Users\HP\Desktop\DOS Practical\3>java Pr3
Hello World
Hello World
How Are You
Hello World
Hello World
How Are You
Hello World
How Are You
How Are You
How Are You

C:\Users\HP\Desktop\DOS Practical\3>
```

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

.

INLAB AIM: Build 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.

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 interact with 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.

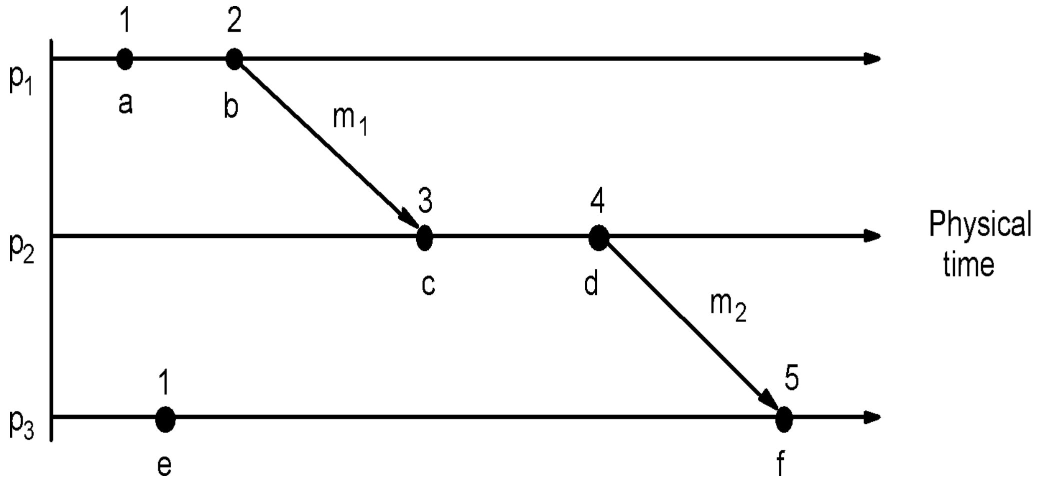
Lamport's Clock:

Logical time and logical clocks:

- A logical clock is a monotonically increasing software counter. It need not relate to a physical clock.
- Each process pi has a logical clock, Li which can be used to apply logical timestamps to

events.

- LC1: L_i is incremented by 1 before each event at process p_i
- LC2:
- (a) when process p_i sends message m, it piggybacks $t = L_i$
- (b) when pj receives (m,t) it sets $L_j := \max(L_j, t)$ and applies LC1 before times tamping the event receive (m)



CODE:

<u>Lamport.java</u>

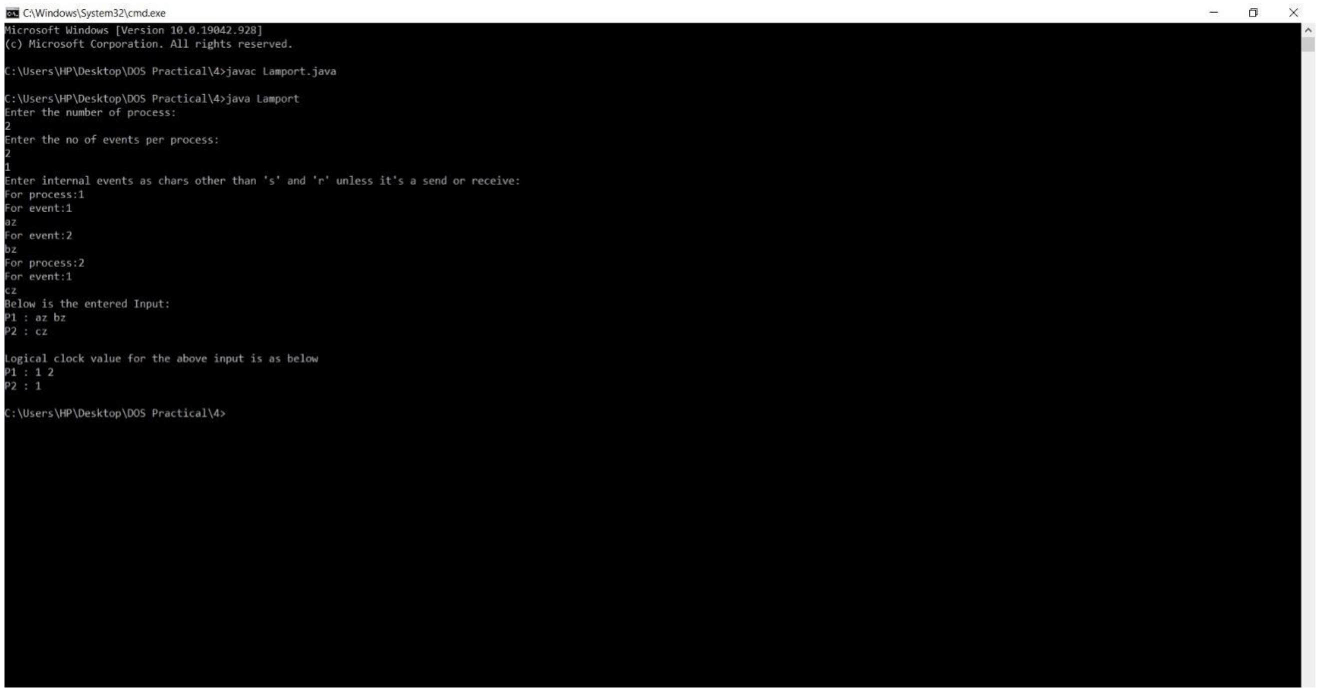
```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;


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')
                {
                    lc[pr][j]=lc[pr][j-1]+1;
                }
                else
                {
                    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:



C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\4>javac Lamport.java

C:\Users\HP\Desktop\DOS Practical\4>java Lamport
Enter the number of process:
2
Enter the no of events per process:
2
1
Enter internal events as chars other than 's' and 'r' unless it's a send or receive:
For process:1
For event:1
az
For event:2
bz
For process:2
For event:1
cz
Below is the entered Input:
P1 : az bz
P2 : cz

Logical clock value for the above input is as below
P1 : 1 2
P2 : 1

C:\Users\HP\Desktop\DOS Practical\4>
```

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 of Remote method invocation (RMI).

INLAB AIM: Construct a distributed application using remote method invocation (RMI) where client 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.

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 client 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.

THEORY/ALGORITHM:

The main objective of RMI is to provide the facility of invoking method on server. This is done by creating 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.

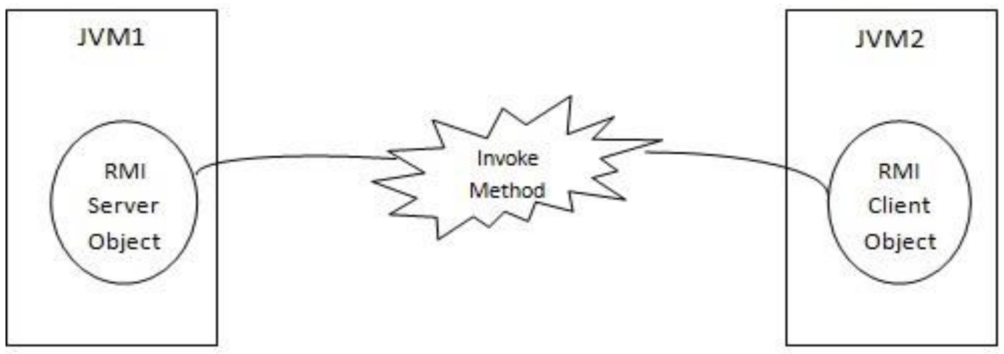


Figure: Method is invoked by client on the server using RMI

RMI is used to communicate between two running java applications on different JVM (Java Virtual 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.

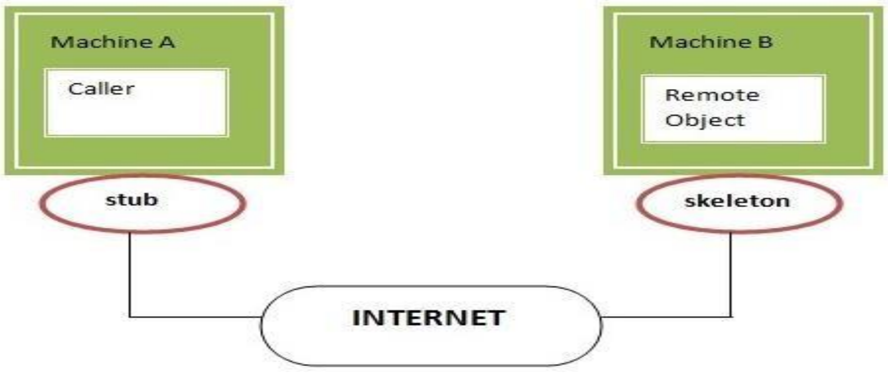


Figure: Role of stub and skeleton in RMI

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:

Remote.java

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

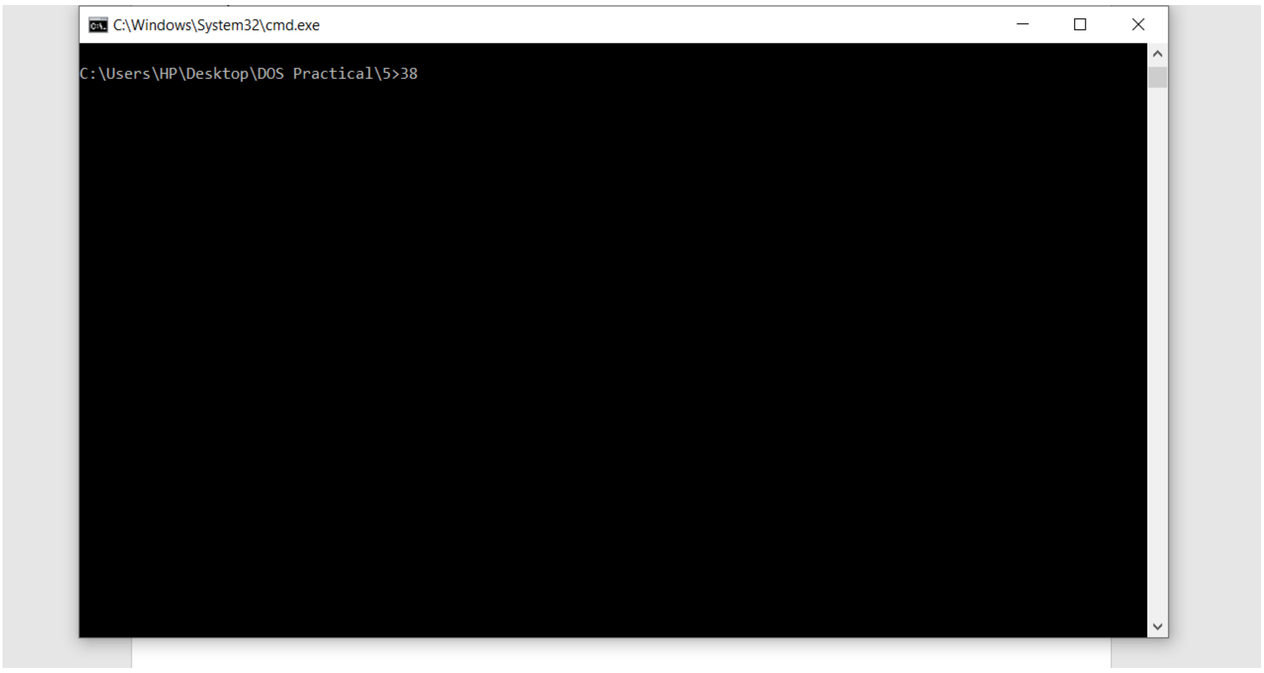
MyServer.java

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

MyClient.java

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

OUTPUT:



```
C:\Users\HP\Desktop\DOS Practical\5>38
```

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

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.

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.

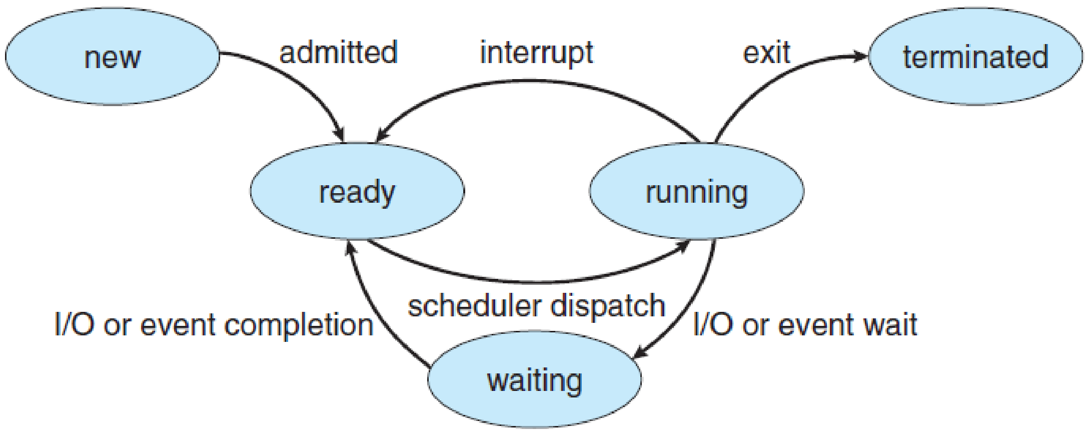


Figure: Lifecycle of process

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 until the program starts. It is also referred to as a born process.

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

- **Waiting:** Sometimes, a process transitions to the waiting state while the process waits for 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.

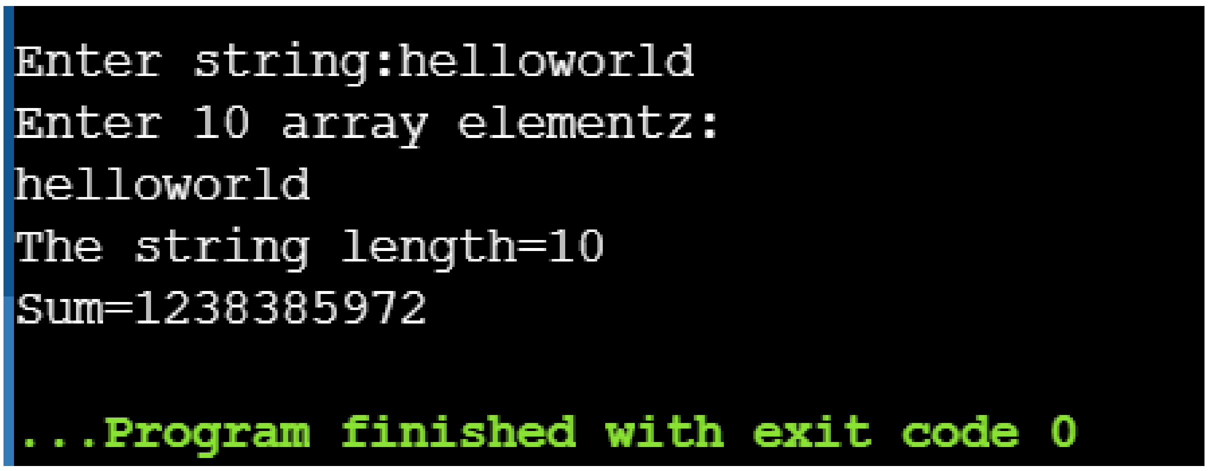
- **Timed waiting**: A process can enter the timed waiting state for a specified interval of time. A process in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

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

CODE:

```
#include<stdio.h>
 #include<unistd.h>
 #include<string.h>
 void main()
 {
  int pid[2],pid1[2],pid2[2],pid3[2],pid4[2];
  int a[20],i,l,s=0;
  char str[20];
  pipe(pid);
  pipe(pid1);
  pipe(pid2);
  pipe(pid3);
  pipe(pid4);
 if(fork()==0)
 {
  sleep(5);
  close(pid[1]);
  read(pid[0],str,sizeof(str));
  for(i=0,l=0;str[i]!='\0';i++)
  l=l+1;
  close(pid3[0]);
  write(pid3[1],&l,sizeof(l));
  sleep(6);
  printf("Enter %d array elementz:",l);
  for(i=0;i<l;i++)
  scanf("%d",&a[i]);
  close(pid1[0]);
  write(pid1[1],a,sizeof(a));
  close(pid4[0]);
 write(pid4[1],&l,sizeof(l));
}
else if(fork()==0)
{
 sleep(2);
 close(pid1[1]);
 close(pid4[1]);
 read(pid4[0],&l,sizeof(l));
 read(pid1[0],a,sizeof(a));
 for(i=0;i<l;i++)
 s=s+a[i];
 close(pid2[0]);
 write(pid2[1],&s,sizeof(s));
}
else
{
 printf("\nEnter string:");
 scanf("%s",str);
 close(pid[0]);
 write(pid[1],str,sizeof(str));
 sleep(7);
 close(pid3[1]);
 read(pid3[0],&l,sizeof(l));
 printf("\nThe string length=%d",l);
 sleep(8);
 close(pid2[1]);
 read(pid2[0],&s,sizeof(s));
 printf("\nSum=%d",s);
}
}
```

OUTPUT:



```
Enter string:helloworld
Enter 10 array elementz:
helloworld
The string length=10
Sum=1238385972

...Program finished with exit code 0
```

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 mutually exclusive 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 to implement 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 achieve mutual exclusion in distributed environment.

Overview:

- If a process wants to enter the critical section, and it does not have the token, it broadcasts a 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 is requested
- Request vector at process i:
- RNi[k] contains the largest sequence number received from process k in a request message
- 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 the system.
- 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 Q and 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:

<u>EchoServer.java</u>

```
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)
{
}
}
}
```

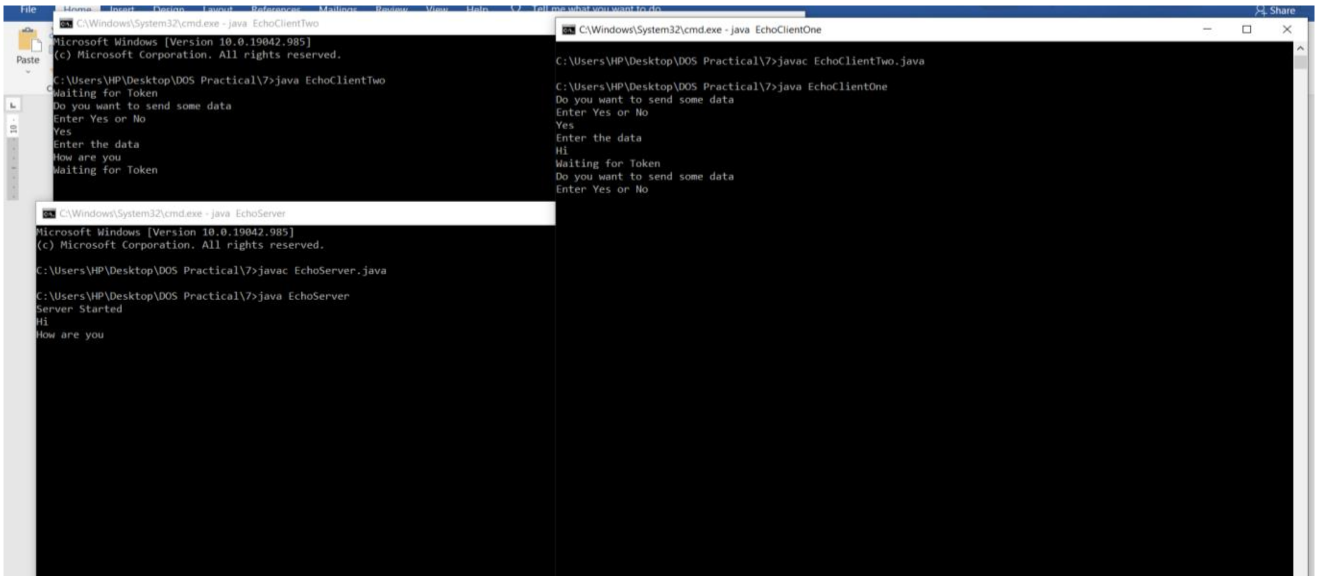
EchoClientOne.java

```
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();
}
}
}
```

EchoClientTwo.java

```
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:



```
C:\Windows\System32\cmd.exe - java  EchoClientTwo
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\7>java EchoClientTwo
Waiting for Token
Do you want to send some data
Enter Yes or No
Yes
Enter the data
How are you
Waiting for Token
```

```
C:\Windows\System32\cmd.exe - java  EchoServer
Microsoft Windows [Version 10.0.19042.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\Desktop\DOS Practical\7>javac EchoServer.java

C:\Users\HP\Desktop\DOS Practical\7>java EchoServer
Server Started
Hi
How are you
```

```
C:\Windows\System32\cmd.exe - java  EchoClientOne

C:\Users\HP\Desktop\DOS Practical\7>javac EchoClientTwo.java

C:\Users\HP\Desktop\DOS Practical\7>java EchoClientOne
Do you want to send some data
Enter Yes or No
Yes
Enter the data
Hi
Waiting for Token
Do you want to send some data
Enter Yes or No
```

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

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.

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 ceasing to 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.

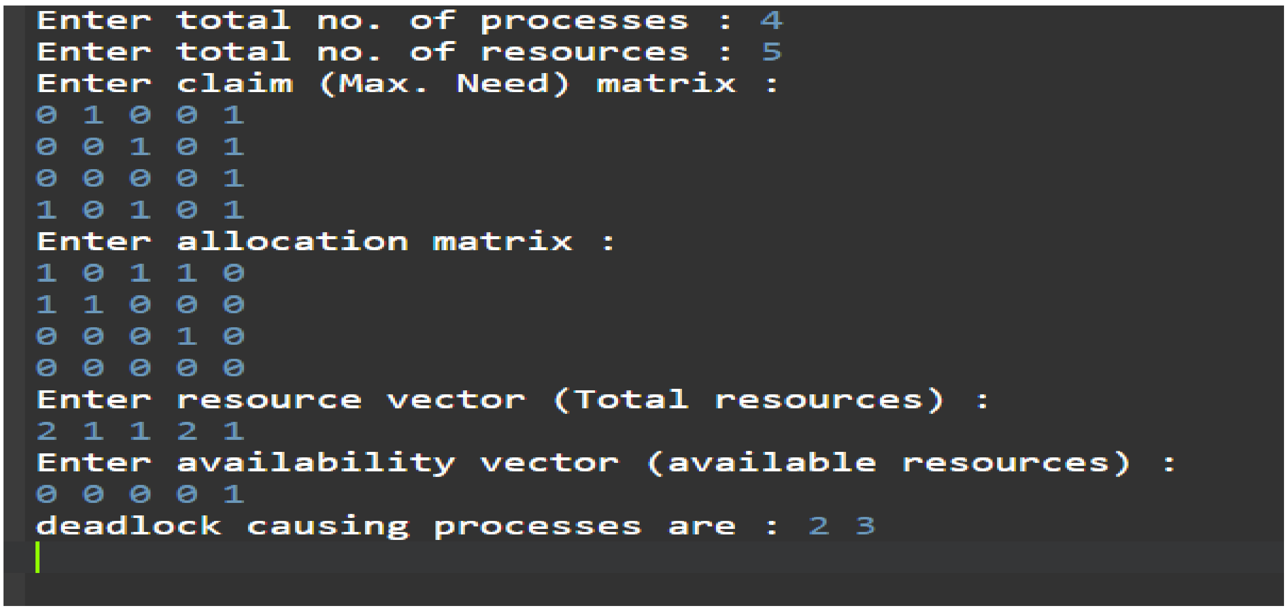
Unlike the centralized case, where all the information is automatically available in the right place, in a distributed system it has to be sent there explicitly. Each machine maintains the graph for its own processes and resources. Several possibilities exist for getting it there. First, whenever an 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:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int found,flag,l,p[4][5],tp,tr,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;
clrscr();
printf("Enter total no of processes");
scanf("%d",&tp);
printf("Enter total no of resources");
scanf("%d",&tr);
```

```
printf("Enter claim (Max. Need) matrix\n");
for(i=1;i<=tp;i++)
{
 printf("process %d:\n",i);
 for(j=1;j<=tr;j++)
 scanf("%d",&c[i][j]);
}
printf("Enter allocation matrix\n");
for(i=1;i<=tp;i++)
{
 printf("process %d:\n",i);
 for(j=1;j<=tr;j++)
 scanf("%d",&p[i][j]);
}
printf("Enter resource vector (Total resources):\n");
for(i=1;i<=tr;i++)
{
 scanf("%d",&r[i]);
}
printf("Enter availability vector (available resources):\n");
for(i=1;i<=tr;i++)
{
 scanf("%d",&a[i]);
 temp[i]=a[i];
}
for(i=1;i<=tp;i++)
{
 sum=0;
 for(j=1;j<=tr;j++)
 {
sum+=p[i][j];
 }
 if(sum==0)
 {
m[k]=i;
k++;
 }
}
for(i=1;i<=tp;i++)
{
 for(l=1;l<k;l++)
 if(i!=m[l])
 {
flag=1;
for(j=1;j<=tr;j++)
if(c[i][j]<temp[j])
{
 flag=0;
 break;
}
}
 if(flag==1)
 {
m[k]=i;
k++;
for(j=1;j<=tr;j++)
temp[j]+=p[i][j];
 }
}
printf("deadlock causing processes are:");
for(j=1;j<=tp;j++)
{
 found=0;
 for(i=1;i<k;i++)
 {
if(j==m[i])
found=1;
 }
 if(found==0)
 printf("%d\t",j);
}
getch();
}
```

OUTPUT:



```
Enter total no. of processes : 4
Enter total no. of resources : 5
Enter claim (Max. Need) matrix :
0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1
Enter allocation matrix :
1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
Enter resource vector (Total resources) :
2 1 1 2 1
Enter availability vector (available resources) :
0 0 0 0 1
deadlock causing processes are : 2 3
```

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 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.

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 ceasing to 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, and prevent.

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 of the global TWF graph, and a deadlock is detected when a probe message returns to the process that initiated 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 itself

then declare a deadlock

else for all *Pj* and *Pk* such that

1 *Pi* is locally dependent upon *Pj* , and

2 *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*

On the receipt of a probe (i, j, k), the site takes the following actions:

```
if
        1 Pk is blocked, and
        2 dependentk (i) is false, and
        3 Pk has not replied to all requests Pj
then
begin
        dependent k (i) = true;
if k=i
        then declare that Pi is deadlocked
else for all Pm and Pn such that
        (a') Pk is locally dependent upon Pm, and
        (b') Pm is waiting on Pn, and
        (c') Pm and Pn are on different sites, send a probe (i, m, n) to the home site of Pn
end.
```

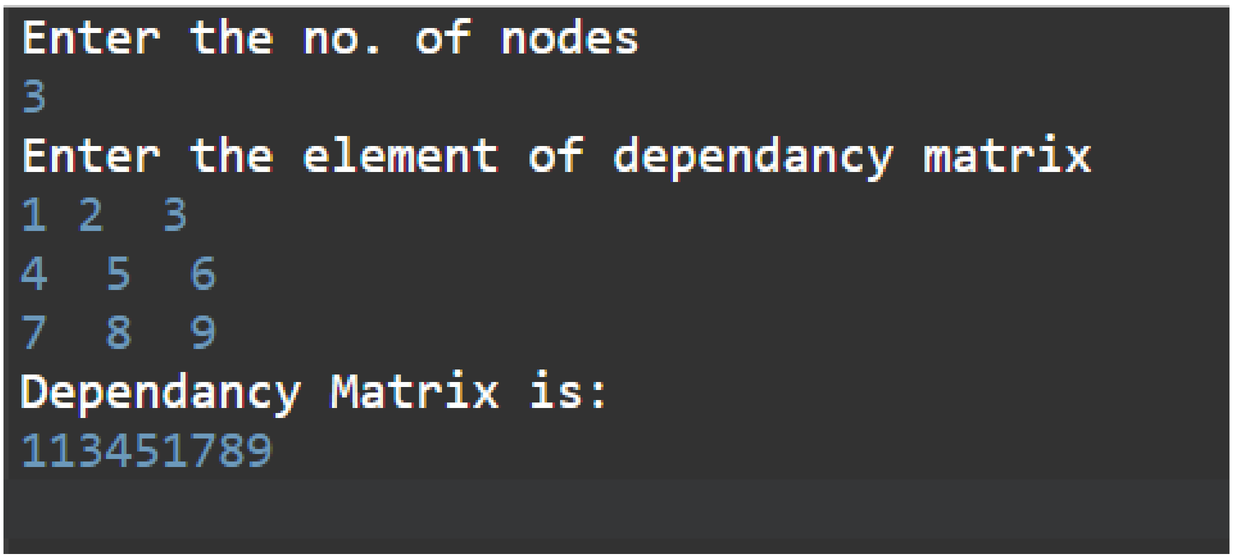
CODE:

```
#include<conio.h>

#include<stdio.h>

void main()

{

int p[10],n,i,p1,s1,sp1,sp2;

// clrscr();

printf("Enter total no. of sites\n");
```

```
scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("Enter total no. of process in S%d\n",i+1);

scanf("%d",&p[i]);

}

for(i=0;i<n;i++)

{

printf("Total no. of process in S%d are %d\n",i+1,p[i]);

}

printf("Enter the site no. and process id for which deadlock detection shold be initiated\n");

scanf("%d %d",&s1,&p1);

printf("Enter the the processes of two different sites connected with requesting edge\n");

scanf("%d %d",&sp1,&sp2);

printf("Probe message is (%d,%d,%d)",p1,sp1,sp2);

if(p1==sp2)

{

printf("Deadlock detected");

}

else

{

getch();

}
```

```
}
```

OUTPUT:



```
Enter the no. of nodes
3
Enter the element of dependancy matrix
1 2  3
4  5  6
7  8  9
Dependancy Matrix is:
113451789
```

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

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.

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 commit a transaction. The protocol results in either all nodes committing the transaction 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 message back to coordinator.

Phase 2 (Commit / Decision Phase)

```
    If coordinator receives a ready to commit reply from all participants in phase I
then
       it send commit message to all participants
else
```

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.

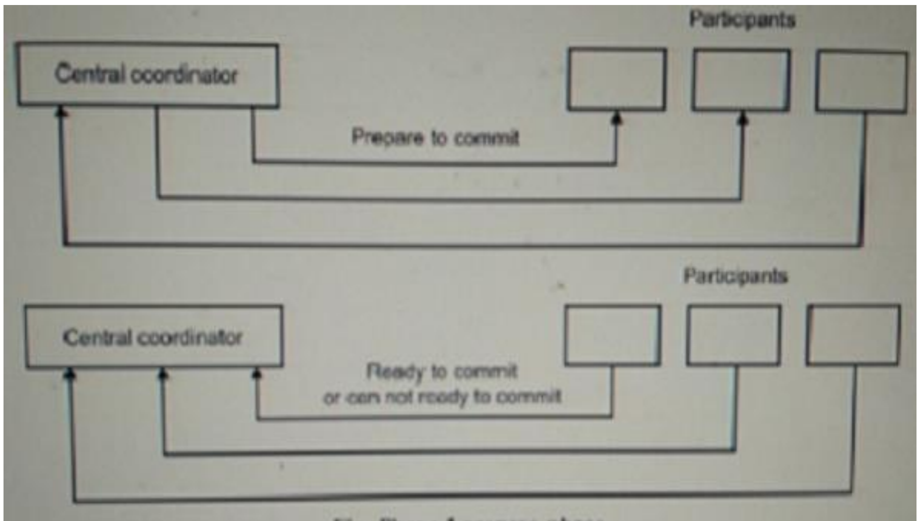


Fig. Phase I prepare phase

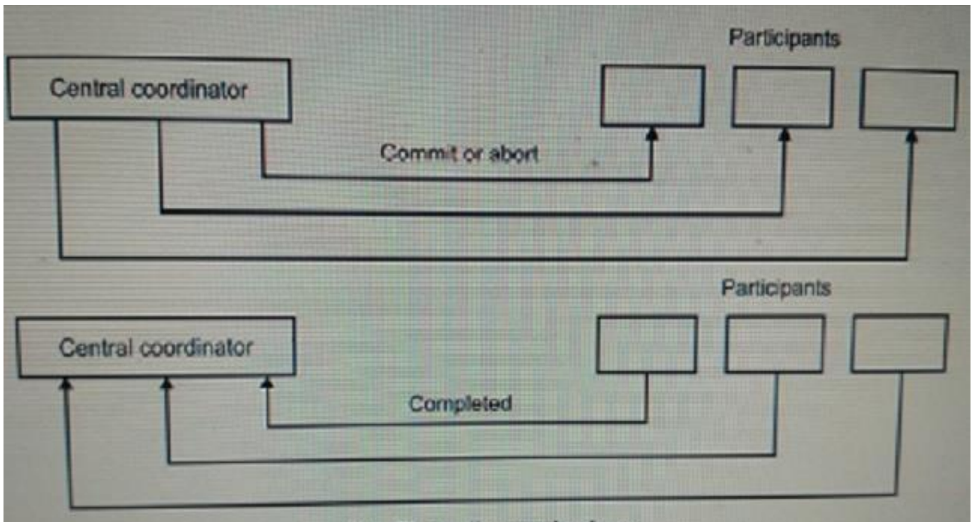


Fig. Phase II commit phase.

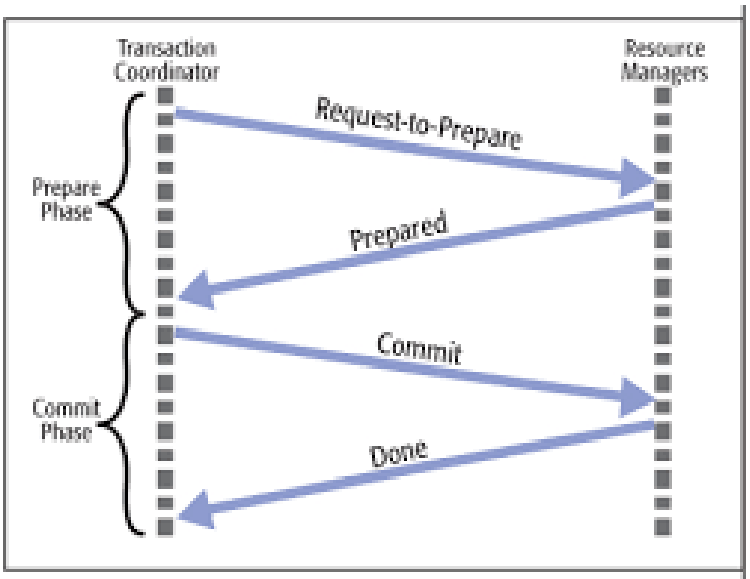


Figure 1 • The two-phase commit protocol

CODE:

Server :-

```
import java.io.*;
import java.net.*;
class TPCServer
{
public static void main(String a[])throws Exception
 {
 BufferedReader br;
 InetAddress lclhost;
 lclhost=InetAddress.getLocalHost();
 Server ser=new Server(lclhost);
System.out.println("Server in sending mode....."); // Sending data to client 1
 ser.setSendPort(9000):  //recport=8000
ser.setRecPort(8001); //sendport=9001
 System.out.println("Send request data to client1..");
br=new BufferedReader(new InputStreamReader(System.in));
String s=br.readLine();
System.out.println("Data is "+s);
 ser.sendData();
 System.out.println("Waiting for response from client1....");
ser.recData(); // Sending data to client 2
ser.setSendPort(9002); //recport=8002
 ser.setRecPort(8003); //senport=9003
System.out.println("Send request data to client2..");
 br=new BufferedReader(new InputStreamReader(System.in));
 String s1=br.readLine();
 System.out.println("Data is "+s1);
 ser.sendData();
System.out.println("Waiting for response from client2....");
 ser.recData(); //Sending the final result to client 1
ser.setSendPort(9000);
 ser.sendData(); //Sending the final result to client 2
ser.setSendPort(9002);
 ser.sendData();
}
}
class Server
{
InetAddress lclhost;
 int sendPort,recPort;
 int ssend =0;
int scounter=0;
 Server(InetAddress lclhost)
{
 this.lclhost=lclhost;
 }
 public void setSendPort(int sendPort)
 {
this.sendPort=sendPort;
 }
 public void setRecPort(int recPort)
 {
```

```
 this.recPort=recPort;
 }

 public void sendData()throws Exception
 {
 DatagramSocket ds;
DatagramPacket dp;
 String data="";
 if(scounter<2 && ssend<2)
 {
 data="VOTE_REQUEST";
 }
 if(scounter<2 && ssend>1)
 {
 data="GLOBAL_ABORT";
data= data + " TRANSACTION ABORTED";
 }
if(scounter==2 && ssend>1)
 {
 data="GLOBAL_COMMIT";
data= data + " TRANSACTION COMMITED";
}
ds=new DatagramSocket(sendPort);
 dp=new DatagramPacket(data.getBytes(),data.length(),lclhost,sendPort-1000);
ds.send(dp);
 ds.close();
 ssend++;
}
public void recData()throws Exception
{
 byte buf[]=new byte[256];
 DatagramPacket dp=null;
 DatagramSocket ds=null;
 String msgStr="";
 try{
 ds=new DatagramSocket(recPort);
 dp=new DatagramPacket(buf,buf.length);
 ds.receive(dp);
 ds.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
}
 msgStr=new String(dp.getData(),0,dp.getLength());
System.out.println("String = "+msgStr);
if(msgStr.equalsIgnoreCase("VOTE_COMMIT"))
 {
 scounter++;
}
System.out.println("Counter value = "+scounter + "\n Send value = "+ssend);
}
}
```

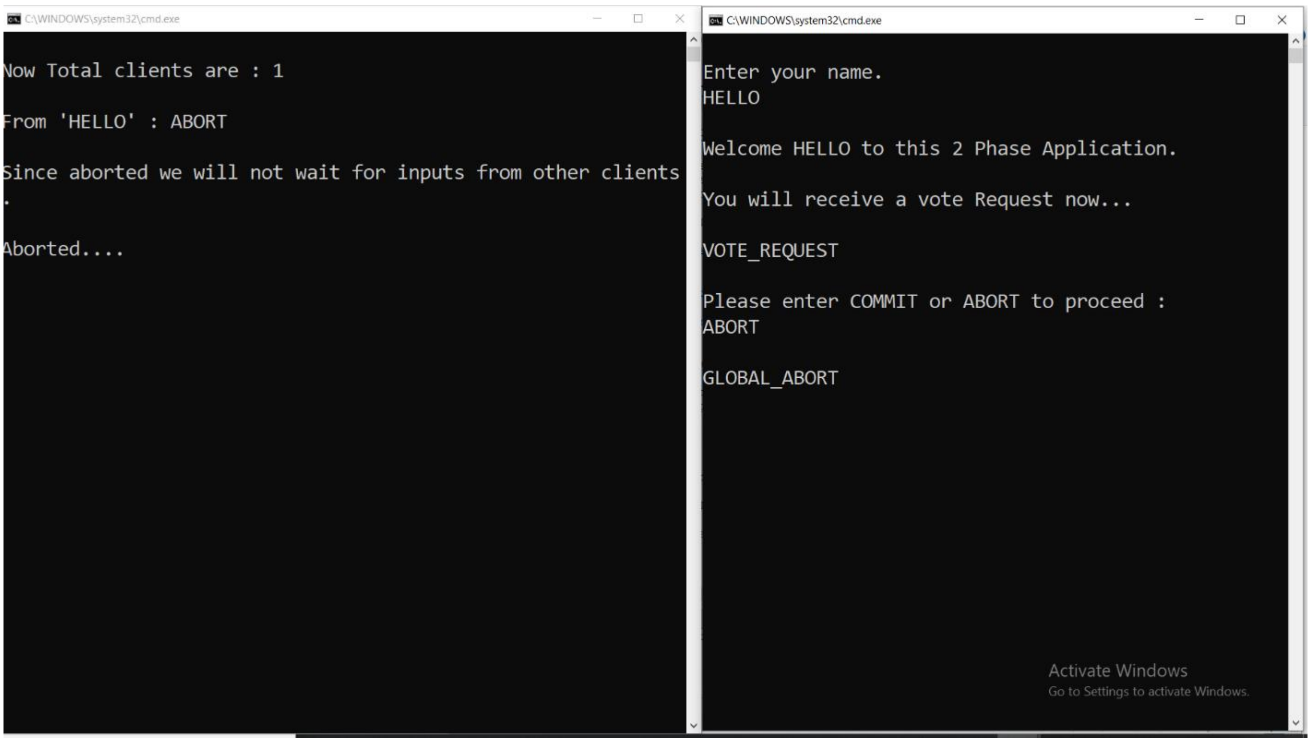
<u>Client-1:-</u>

```
import java.io.*;
import java.net.*;
class TPCClient1
{
public static void main(String a[])throws Exception
 {
InetAddress lclhost;
lclhost=InetAddress.getLocalHost();
 Client clnt=new Client(lclhost);
 clnt.setSendPort(9001); //recport=8000
clnt.setRecPort(8000); //sendport=9001
clnt.recData();
 clnt.sendData();
 clnt.recData();
 }
}
class Client
{
InetAddress lclhost;
 int sendPort,recPort;
Client(InetAddress lclhost)
 {
 this.lclhost=lclhost;
}
 public void setSendPort(int sendPort)
 {
 this.sendPort=sendPort;
}
public void setRecPort(int recPort)
{
this.recPort=recPort;

 }
 public void sendData()throws Exception
 {
 BufferedReader br;
 DatagramSocket ds;
DatagramPacket dp;
 String data="";
 System.out.println("Enter the Response 'VOTE_COMMIT' || 'VOTE_ABORT' ");
 br=new BufferedReader(new InputStreamReader(System.in));
 data = br.readLine();
System.out.println("Data is "+data);
 ds=new DatagramSocket(sendPort);
 dp=new DatagramPacket(data.getBytes(),data.length(),lclhost,sendPort-1000);
ds.send(dp);
 ds.close();
}

public void recData()throws Exception
{
 byte buf[]=new byte[256];
DatagramPacket dp;
 DatagramSocket ds;
ds=new DatagramSocket(recPort);
 dp=new DatagramPacket(buf,buf.length);
 ds.receive(dp);
 ds.close();
 String msgStr=new String(dp.getData(),0,dp.getLength());
 System.out.println("Client1 data " +msgStr);
 }
}
```

<u>Client-2</u>

```
import java.io.*;
import java.net.*;
class TPCClient1
{
public static void main(String a[])throws Exception
 {
InetAddress lclhost;
 lclhost=InetAddress.getLocalHost();
Client clnt=new Client(lclhost);
 clnt.setSendPort(9001); //recport=8000
 clnt.setRecPort(8000); //sendport=9001
 clnt.recData();
 clnt.sendData();
 clnt.recData();
 }
}
class Client
{
InetAddress lclhost;
 int sendPort,recPort;
 Client(InetAddress lclhost)
 {
 this.lclhost=lclhost;
}
 public void setSendPort(int sendPort)
 {
 this.sendPort=sendPort;
 }

 public void setRecPort(int recPort)
 {
this.recPort=recPort;
 }
 public void sendData()throws Exception
 {
 BufferedReader br;
 DatagramSocket ds;
 DatagramPacket dp;
 String data="";
 System.out.println("Enter the Response 'VOTE_COMMIT' || 'VOTE_ABORT' ");
 br=new BufferedReader(new InputStreamReader(System.in));
 data = br.readLine();
 System.out.println("Data is "+data);
 ds=new DatagramSocket(sendPort);
 dp=new DatagramPacket(data.getBytes(),data.length(),lclhost,sendPort-1000);
 ds.send(dp);
 ds.close();
 }
public void recData()throws Exception
 {
 byte buf[]=new byte[256];
 DatagramPacket dp;
 DatagramSocket ds;
ds=new DatagramSocket(recPort);
 dp=new DatagramPacket(buf,buf.length);
  ds.receive(dp);
 ds.close();
 String msgStr=new String(dp.getData(),0,dp.getLength());
 System.out.println("Client1 data " +msgStr);

}
}
```

OUTPUT:



```
Now Total clients are : 1

From 'HELLO' : ABORT

Since aborted we will not wait for inputs from other clients
.

Aborted....
```

```
Enter your name.
HELLO

Welcome HELLO to this 2 Phase Application.

You will receive a vote Request now...

VOTE_REQUEST

Please enter COMMIT or ABORT to proceed :
ABORT

GLOBAL_ABORT
```

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.

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

OBJECTIVES:

- To know the concept of Supply Chain Management.

INLAB

AIM: Write a case study on Supply Chain Management.

OBJECTIVES:

- To know the concept of Supply Chain Management.

THEORY/ALGORITHM:

Supply Chain Management “The supply chain encompasses all activities associated with the flow and trans-formation of goods from raw materials stage (extraction), through to the end user, as well as the associated information flows. Material and information flow both up and down the supply chain. Supply chain management (SCM) is the integration of these activities through improved supply chain relationships, to achieve a sustain-able competitive advantage” (Handfield &amp; Nichols, 1999: 2). This definition is taken as an exemplary one, while several others have been proposed. A number of reviews and systemizations have been provided (e.g. Bechtel &amp; Jayaram, 1997; Cooper et al., 1997; Ganeshan et al., 1998; Croom et al., 2000; 2000; Mentzer et al., 2001; Seuring, 2001a; Otto &amp; Kotzab, 2001; Müller et al., 2003). While these contributions point towards different definitions and conceptualizations, at least two recurring themes can be observed:

(1) Supply chains deal with material and information flows, which

(2) have to be managed in a cooperative way by all partners involved in the supply chain. Several authors have pointed toward the problems in even establishing a central content of supply chain management (Mouritsen et al., 2003; Chen &amp; Paulraj, 2004) as well as observed problems practitioners face in aiming at implementing supply chain management (Fawcett &amp; Magnan, 2002). One central issue is identi-fying which entities are constitutive for a supply chain. It is not trivial to decide which companies form certain supply chains and how far

integration has to reach. As Frohlich &amp; Westbrook (2001) argue in their survey- based research, companies to this point have predominantly looked a stage up or down the supply chain. Furthermore, few examples exist where information from different stages, espe-cially more Than two stages of the supply chain, have been collected (e.g. Cooper &amp; Slagmulder, 2004; Seuring, 2001; 2002). Related to this, Stuart et al. (2002: 431) emphasize the need for a "customer fo-cused approach" in management research, where practitioners' perceptions of research are taken into account. In this regard, they highlight that case studies can be a "powerful, influential, and useful contribution to both management practice and theory development" (Stuart et al., 2002) and have a high validity with practi-tioners (Voss et al., 2002: 195). Some problems faced in supply chain manage-ment can be perceived as complex, unstructured situations, where a mapping of major variables (Stuart et al., 2002) or exploration to uncover areas for research and theory development (Voss et al., 2002) are suitable research strategies. These are typical situations where a case study approach seems appropriate (Yin, 2003; Saunders et al., 2003). Hence, two central questions in related research are:

1.How can a suitable supply chain which can serve as a case be identified?

2.How can access be gained to the different stages of the supply chain to allow data collection at some or all relevant stages?

Conclusion : This paper discusses why case study research proves to be an interesting option for empirical research in supply chain management. It is not intended to rewrite or reinvent case study research, as numerous, comprehensive accounts of such re-search already exist. In contrast, the three cases briefly outlined here show exam-ples of how the research process was carried out in such projects.

In section 2, two questions were raised, which will not be discussed against the background of the cases presented.

1.How can a suitable supply chain which can serve as a case be identified?

As discussed in literature (e.g. Yin, 2003: 21), case selection often has to be opportunistic. As the example of Ecolog shows, it might be difficult to find suitable examples at all. In this case only a second suitable case study could be identified, which is now research to provide insight in a cross- case analysis. Still, the active search for appropriate cases that allow insight into how supply chain management works across several stages of the supply chain will be very useful. In general, case studies often emerge from existing contacts a researcher has to industry. This was the case for the Otto and Steilmann examples, as presented in this paper. While this is justifiable, the researcher still needs to assess why these cases are Case study on Supply Chain Managementuseful and what the main purpose for researching them would be. This way, one central critique of case study research (that it lacks the rigor of other approaches) could be avoided or at least mitigated.

2.How can access be gained to the different stages of the supply chain to allow data collection at some or all relevant stages?

A key approach therefore might be starting at a focal company. From this point onwards, suppliers could be identified. In the Steilmann case, focus was provided by the particular product studies, so there were only two suppliers involved and no further selection possible. A different example is provided in the paper of Chivaka (2005, in this volume). Initially identifying focal companies (as we did), he asked them to identify suitable first-tier suppliers. By repeating this at the supplier, he was able to reach a second-tier supplier, which finally allowed him to research three different three-stage supply chains.

Seuring As the reach is beyond a single organization, more flexible and opportunistic approaches of getting access to and collecting data from various stages of the supply chain have to be used. As a final comment, it has to be admitted that the written findings of such research always idealize the research process, but one strength of the case study method is its flexibility (Yin, 2003; Stuart et al., 2002). Rigor, as expressed in valid and reliable research, stems from process documentation. Triangulation of findings by using multiple sources of evidence is a second important measure. Case study research in supply chain management can help to further explore the field, but is also valid for theory building, testing and extension.

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 web services.

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, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top 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 the following 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 as HTML, 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:

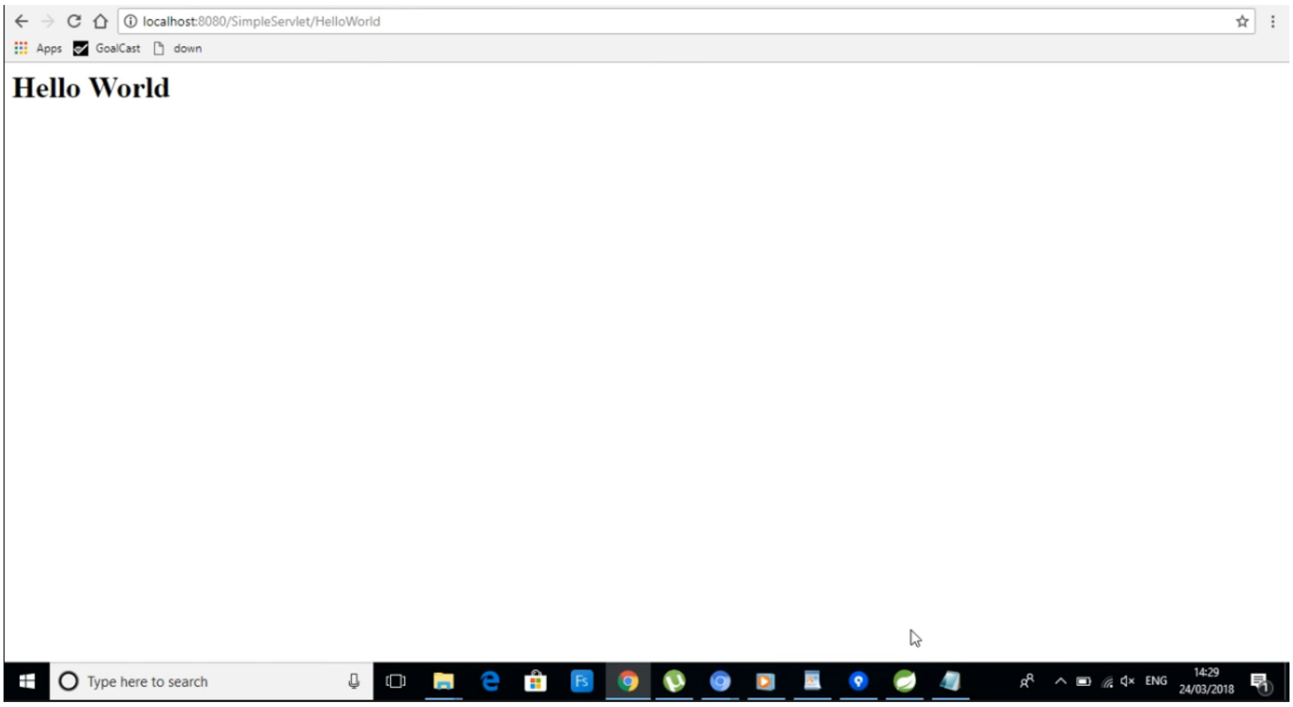
Hello Serverlet:

```
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/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee 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 :



Hello World

CONCLUSION:

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