**Name :** Shivam Tiwari

**Roll No:** 5117060

**Aim :** Deadlock Management in Distributed System

**Theory:**

If the total amount of request made by multiple concurrent processes for resources of a certain type exceeds the amount available, some strategy is needed to order the assignment of a resource in time. Care must be taken that the strategy applied cannot cause a deadlock that is , a situation in which competing processes prevent their mutual progress even though no single one requests more resources than are available.

A **deadlock** is a condition in a system where a set of processes (or threads) have requests for resources that can never be satisfied. Essentially, a process cannot proceed because it needs to obtain a resource held by another process but it itself is holding a resource that the other process needs. More formally, [Coffman](http://people.cs.umass.edu/~mcorner/courses/691J/papers/TS/coffman_deadlocks/coffman_deadlocks.pdf) defined four conditions have to be met for a deadlock to occur in a system:

**Necessary conditions for Deadlock:**

1. Mutual exclusion: If a resource is held by a process, any other process requesting for that resources must wait until that resources has been released.

1. Hold and wait: Processes are allowed to request for new resources without releasing the resources that they are currently holding.

1. No-preemption:. A resource that has been allocated to a process becomes available for the allocation to another process only after it has been voluntarily released by the process holding it

1. Circular wait - 2 or more processes must form a chain in which each process is waiting for a resource that is held by the next member of the chain.

Handling Deadlock in Distributed System:

In principle, handling of Deadlock in Ds is more complex than in centralised system because the resources, the processes & other relevant info are scattered on different nodes of the the System.

The commonly used strategy to handle deadlock are as follows:

1. Avoidance.
2. Prevention.
3. Detection & recovery.

**Output:**

import java.util.\*; class Message

{

public int initiator=0; public int from=0; public int to=0;

public Message(int i,int j,int k)

{

initiator=i; from=j; to=k;

}

public String toString()

{

return "("+initiator+","+from+","+to+")";

}

}

public class Deadlock {

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int graph[][];

boolean isDeadlock=false; //initially there is no deadlock i n the system

System.out.println("Enter the number of processes");

//Read the number of processes

int n=sc.nextInt(); graph=new int[n][n];

//Wait for graph(WFG) is a directed graph used for deadlock detection in operating systems

System.out.println("Enter the wait for graph:");

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

{

for(int j=0;j<n;j++)

{

graph[i][j]=sc.nextInt();

}

}

System.out.println("the wait for graph is:");

new Deadlock().Display(graph);

System.out.println("Enter the process initiating probe");

//The probe message circulates along the edges of WFG to detect a cycle. int init=sc.nextInt();

System.out.println("Initiating probe...");

List<Message> mess\_list=new ArrayList<Message>(); int count=0;

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

{

for(int j=0;j<n;j++)

{

if(graph[i][j]==1)

{

Message m=new Message(init,i,j);

mess\_list.add(m); count+=1;

}

}

}

System.out.println(mess\_list);

for(int i=0;i<count;i++)

{

for(int j=0;j<count;j++)

{

//A process Pi declares the deadlock if probe messages initiated by process Pi returns to itself.

if(mess\_list.get(i).initiator==mess\_list.get(j).to) isDeadlock=true;

}

}

if(isDeadlock)

System.out.println("The Deadlock has been detected..."); else

System.out.println("No Deadlock has been detected...");

}

void Display(int[][] mat)

{

int n=mat[0].length; int m=mat.length;

for(int i=0;i<m;i++)

{

for(int j=0;j<n;j++)

{

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

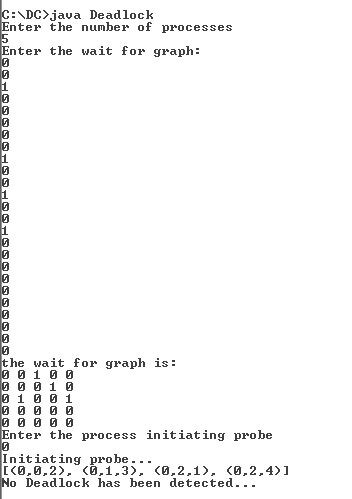
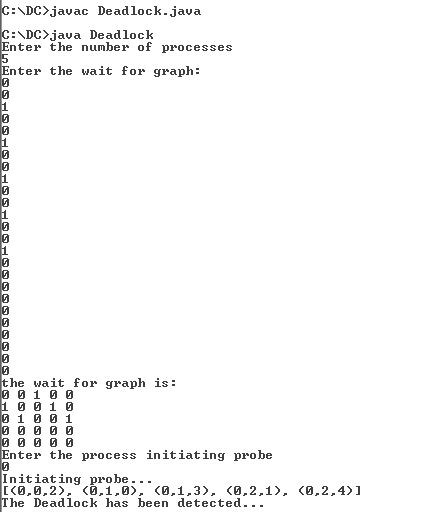
}

System.out.println();

}

}

}



Conclusion:

Thus , we have successfully implemented deadlock management in distributed system.