Practical 4

Distributed Operating System

Name - Ritesh Parkhi

Roll No. – 46

Batch – 3

AIM:
Construct a java program to demonstrate the Distributed Deadlock Detection using Chandy Haas Misra.

THEORY:

Chandy-Misra-Haas's distributed deadlock detection algorithm is an edge chasing algorithm to detect deadlock in distributed systems.

In edge chasing algorithm, a special message called *probe* is used in deadlock detection. A *probe* is a triplet *(i, j, k)* which denotes that process P_i has initiated the deadlock detection and the message is being sent by the home site of process P_j to the home site of process P_k.

The probe message circulates along the edges of WFG to detect a cycle. When a blocked process receives the probe message, it forwards the probe message along its outgoing edges in WFG. A process P_i declares the deadlock if probe messages initiated by process P_i returns to itself.

Algorithm:

Process of sending probe:

1. If process P_i is locally dependent on itself then declare a deadlock.

2. Else for all P_j and P_k check following condition:

- **(a).** Process P_i is locally dependent on process P_j
- **(b).** Process P_j is waiting on process P_k
- **(c).** Process P_j and process P_k are on different sites.

If all of the above conditions are true, send probe (i, j, k) to the home site of process P_k.

On the receipt of probe (i, j, k) at home site of process P_k:

1. Process P_k checks the following conditions:

- **(a).** Process P_k is blocked.
- **(b).** $dependent_k[i]$ is *false*.
- **(c).** Process P_k has not replied to all requests of process P_j

If all of the above conditions are found to be true then:

1. Set $dependent_k[i]$ to true.
2. Now, If k == i then, declare the P_i is deadlocked.
3. Else for all P_m and P_n check following conditions:

- **(a).** Process P_k is locally dependent on process P_m and
- **(b).** Process P_m is waiting upon process P_n and
- **(c).** Process P_m and process P_n are on different sites.

4. Send probe (i, m, n) to the home site of process P_n if above conditions satisfy.

Thus, the *probe* message travels along the edges of transaction wait-for (TWF) graph and when the *probe* message returns to its initiating process then it is said that deadlock has been detected.

PROGRAM:

```
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 ChandyHaasMisra {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int graph[][];
        boolean isDeadlock = false;
```

```
        System.out.println("Enter the number of processes");
        int n = sc.nextInt();
        graph = new int[n][n];
        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 ChandyHaasMisra().Display(graph);
        System.out.println("Enter the process initiating probe");
        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++) {
                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...");
        sc.close();
    }

    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();
        }
```

```
    }
}
```

OUTPUT:

```
PS C:\Users\acer\OneDrive\Desktop\8th_sem\Dos\Practicals\Pract_4> cd "c:\Users\acer\OneDrive\Deskto
p\8th_sem\Dos\Practicals\Pract_4\" ; if ($?) { javac ChandyHaasMisra.java } ; if ($?) { java Chandy
HaasMisra }
Enter the number of processes
5
Enter the wait for graph:
0
0
1
0
0
1
0
0
1
0
0
1
0
0
1
0
0
0
0
0
0
0
0
0
0
the wait for graph is:
0 0 1 0 0
1 0 0 1 0
0 1 0 0 1
0 0 0 0 0
0 0 0 0 0
Enter the process initiating probe
0
Initiating probe...
```

```
0
0
0
0
0
0
0
the wait for graph is:
0 0 1 0 0
1 0 0 1 0
0 1 0 0 1
0 0 0 0 0
0 0 0 0 0
Enter the process initiating probe
0
Initiating probe...
[(0,0,2), (0,1,0), (0,1,3), (0,2,1), (0,2,4)]
The Deadlock has been detected...
PS C:\Users\acer\OneDrive\Desktop\8th_sem\Dos\Practicals\Pract_4>
```

CONCLUSION:

Hence we have successfully built a program to implemented Chandy-Haas-Misra.