**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 Pi has initiated the deadlock detection and the message is being sent by the home site of process Pj to the home site of process Pk.

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

**Algorithm:**

**Process of sending probe:**

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

2. Else for all Pj and Pk check following condition:

* **(a).** Process Pi is locally dependent on process Pj
* **(b).** Process Pj is waiting on process Pk
* **(c).** Process Pj and process Pk are on different sites.

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

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

1. Process Pk checks the following conditions:

* **(a).** Process Pk is blocked.
* **(b).** dependentk[i] is *false*.
* **(c).** Process Pk has not replied to all requests of process Pj

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

1. Set dependentk[i] to true.   
2. Now, If k == i then, declare the Pi is deadlocked.   
3. Else for all Pm and Pn check following conditions:

* **(a).** Process Pk is locally dependent on process Pm and
* **(b).** Process Pm is waiting upon process Pn and
* **(c).** Process Pm and process Pn are on different sites.

4. Send probe (i, m, n) to the home site of process Pn 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();

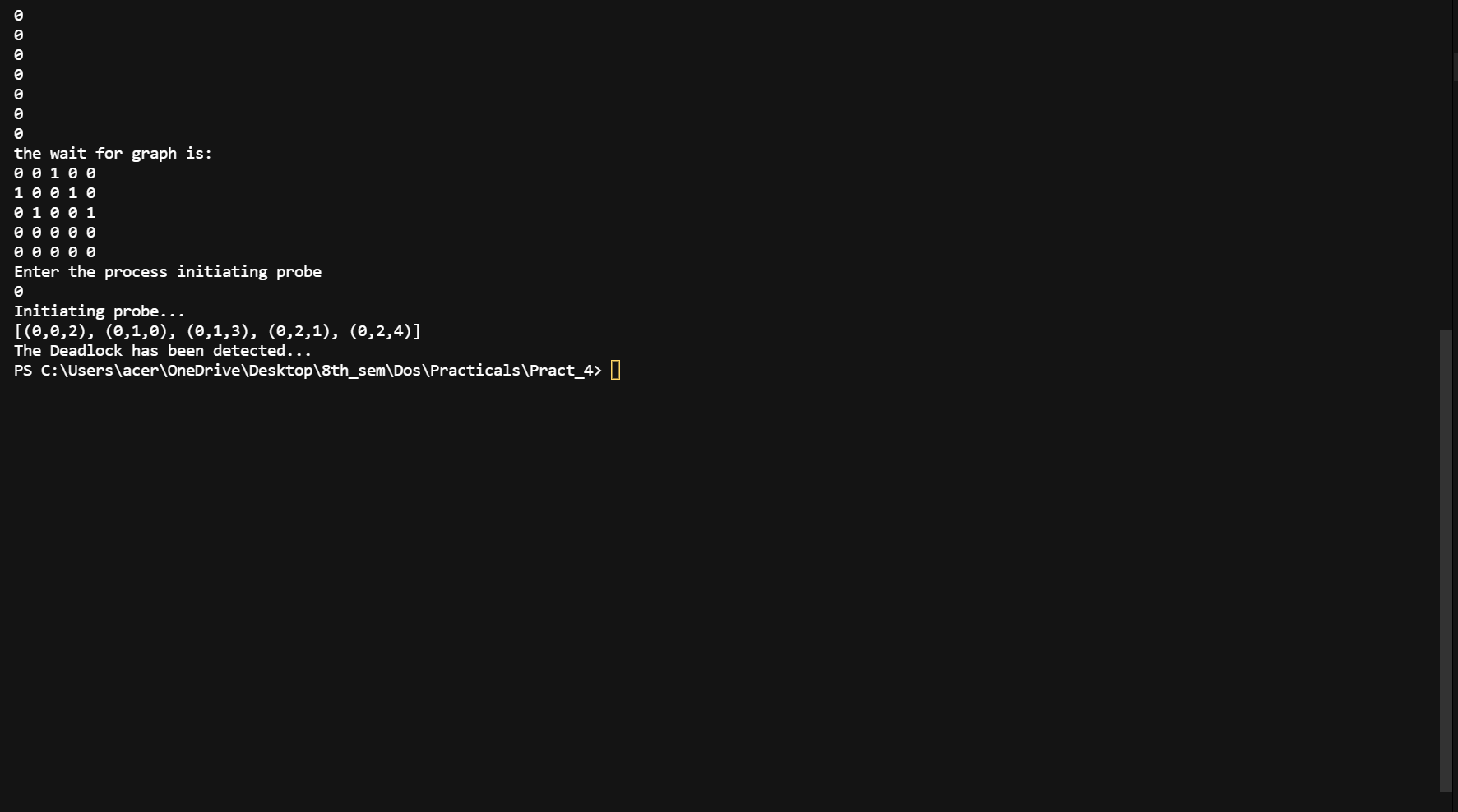
        }

    }

}

**OUTPUT:**





**CONCLUSION:**

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