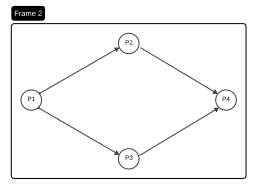
**19CSE312 – DISTRIBIUTED SYSTEMS**

**ALGORITHM IMPLEMENTATION**

**Chandy Misra Haas OR Model Algorithm**



**Team Details: GROUP -12**

|  |  |
| --- | --- |
| **Roll Number** | **Name** |
| CB.EN. U4CSE21031 | K.Deepak |
| CB.EN. U4CSE21042 | P.Sheshu |
| CB.EN. U4CSE21047 | Reshiha R G |
| CB.EN. U4CSE21063 | T.Vinay Kumar |

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

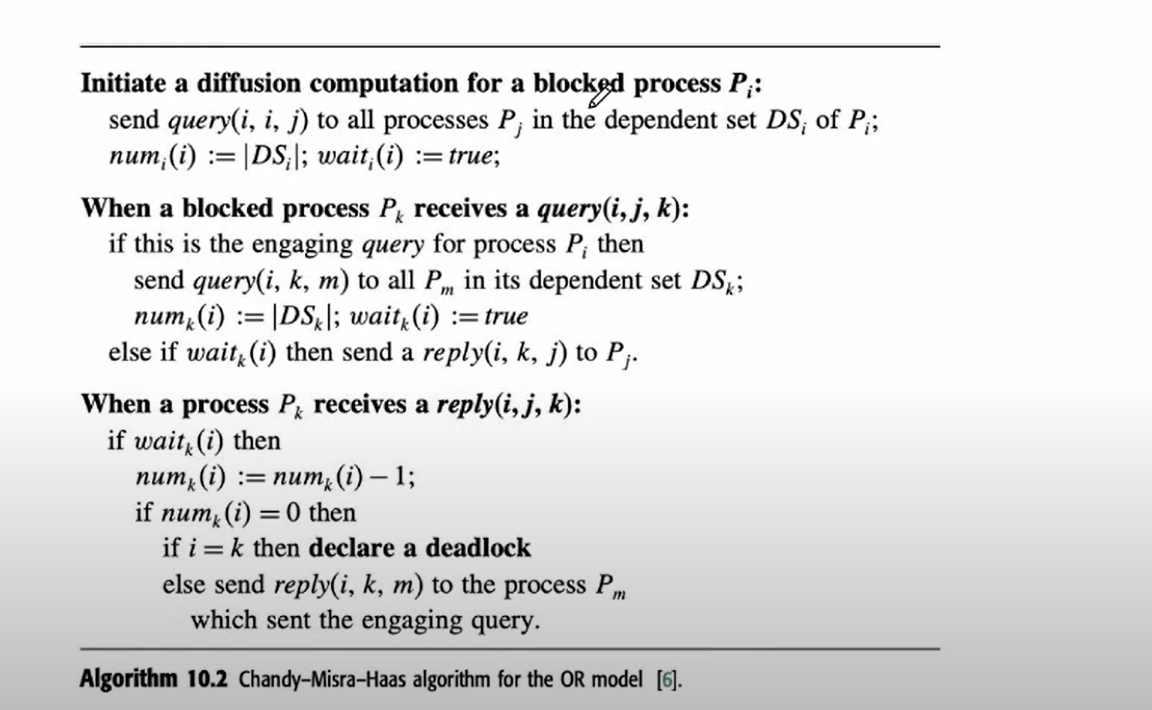
In edge chasing algorithm, a special message called probe is used in deadlock detection. A **probe** is a triplet **(i, j, k)** which denotes that process Pi has **initiated** the **deadlock detection** and the message is being sent by the home site of process Pj to the home site of process Pk.

**The OR model** of requests allows a computation making multiple different resource requests toun‐block as soon as any are granted

– **A cycle is a necessary condition**

– **A knot is a sufficient condition**

**ALGORITHM:**



**Performance:**

* Algorithm requires at most exchange of m(n-1)/2 messages to detect deadlock. Here, m is number of processes and n is the number of sites.
* The delay in detecting the deadlock is O(n).

**Algorithm Implementation**

The Source code is written in Java 8 and tested in Windows11 CMD. Source files Includes:

InitiateProcess.java – For Creating a **Global WFG**

Process.java – Working of the Algorithm **(Query and Reply)**

Initiator.java – Process Which **Initiates and Detects Deadlock**

and dependencies like Dependencies.txt

**Source Codes:**

**(Note: Sources Codes are Kept in a Separate Folder Named (Source)**

1. **InitiateProcess.java**



1. **Process.java**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashMap;

import java.util.Random;

*public* *class* Process {

*private* int[][] wfg;

*private* DatagramSocket dgSocket;

*private* Boolean[] waitFlag;

*private* int[] num;

*private* ArrayList<String> dependentSet = *new* ArrayList<String>();

*private* ArrayList<String> processNameList = *new* ArrayList<String>();

*private* ArrayList<String> engagingQuerySender = *new* ArrayList<String>();

    Random rnd = *new* Random();

*private* String pName;

    HashMap<String, String> processNames =*new* HashMap<String, String> ();

*private* int pId;

*public* Process(String *pName*, int *pId*,int *size*) throws IOException {

        this.pName = *pName*;

        int end =*size*;

        this.pId = *pId*;

        wfg=*new* int[end][end];

        int init, aa, bb, x = 0;

        ArrayList<Process> processSet = *new* ArrayList<Process>();

        ClassLoader loader = Thread.currentThread().getContextClassLoader();

        InputStream is = loader.getResourceAsStream("Dependencies.txt");

        BufferedReader in = *new* BufferedReader(*new* InputStreamReader(is));

        String line;

        line = in.readLine();

        line = in.readLine();

*while* ((line = in.readLine()) != null) {

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

            aa = 3;

            bb = 4;

*for* (int y = 0; y < end; y++) {

                wfg[x][y] = Integer.parseInt(line.substring(aa, bb));

                aa += 2;

                bb += 2;

            }

            processNameList.add(x, "P" + (x + 1));

            processNames.put(String.valueOf(x + 1), "P" + (x + 1));

            processNames.put("P" + (x + 1), String.valueOf(x + 1));

            x++;

        }

        System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

        System.out.println();

        System.out.println(" CHANDY-MISRA-HAAS DISTRIBUTED DEADLOCK DETECTION ALGORITHM OR MODEL ( DIFFUSION BASED) ");

        System.out.println();

*//      System.out.println(" Keys :");*

*//      for (String key : processNames.keySet()) {*

*//          System.out.print(key);*

*//          System.out.print(",");*

*//      }*

        System.out.println();

        System.out.println("-----------WFG---------");

*for* (int i = 0; i < processNameList.size(); i++) {

            System.out.print("\t"+processNameList.get(i));

        }

        System.out.println();

*for* (int i = 0; i < end; i++) {

            int state = i;

            state = state + 1;

            System.out.print("P" + state + "\t");

*for* (int j = 0; j < end; j++) {

                System.out.print(wfg[i][j] + "\t");

            }

            System.out.println();

        }

        waitFlag = *new* Boolean[wfg[0].length];

        num = *new* int[wfg[0].length];

        Arrays.fill(waitFlag, false);

*try* {

            dgSocket = *new* DatagramSocket(*pId* \* 1000);

        } *catch* (SocketException *ex*) {

            System.exit(1);

        }

*for* (int i = 0; i < wfg[0].length; ++i) {

*if* (wfg[*pId* - 1][i] == 1) {

                dependentSet.add(processNames.get(String.valueOf(i + 1)));

            }

        }

        System.out.println();

        System.out.print("Dependent Set for " + *pName* + " : ");

*for* (int i = 0; i < dependentSet.size(); ++i) {

            System.out.print(dependentSet.get(i));

        }

        System.out.println();

    }

*public* *synchronized* void deadLockDetectionInitiated() {

        String msg;

*try* {

            System.out.println("Deadlock detection Initiated");

*for* (int j = 0; j < dependentSet.size(); ++j) {

                msg = "QUERY-" + pName + "," + pName + "," + dependentSet.get(j);

                byte buff[] = msg.getBytes();

                DatagramPacket dgPacketSend = *new* DatagramPacket(buff, buff.length, InetAddress.getLocalHost(),

                        Integer.parseInt((processNames.get(dependentSet.get(j)))) \* 1000);

                dgSocket.send(dgPacketSend);

                System.out.println("QUERY(" + pName + "," + pName + "," + dependentSet.get(j)

                        + ") sent to Dependent Process " + dependentSet.get(j));

            }

            waitFlag[pId - 1] = true;

            num[pId - 1] = dependentSet.size();

        } *catch* (IOException *ex*) {

            System.err.println(ex);

        }

    }

*public* *synchronized* void processActivated() throws InterruptedException {

*while* (true) {

*try* {

                System.out.println(String.format("Process %s Active and Listening",pName));

                byte buff[] = *new* byte[128];

                DatagramPacket dgPacketReceive = *new* DatagramPacket(buff, buff.length);

                dgSocket.receive(dgPacketReceive);

                String strMsg = *new* String(dgPacketReceive.getData());

*if* (strMsg != null && !strMsg.isEmpty()) {

                    strMsg = strMsg.trim();

                    String msgType = strMsg.split("-")[0];

                    String triplet = strMsg.split("-")[1];

                    String initProcessName = triplet.split(",")[0];

                    String senderProcessName = triplet.split(",")[1];

                    String receiverProcessName = triplet.split(",")[2];

                    System.out.println(msgType + "(" + initProcessName + "," + senderProcessName + ","

                            + receiverProcessName + ") received from Process " + senderProcessName);

                    String msg;

*//System.out.println("dependentSet:" + dependentSet.size());*

*if* (msgType.equals("INITIATOR")) {

                        deadLockDetectionInitiated();

                    }

*if* (dependentSet.size() > 0) {

*if* (msgType.equals("QUERY")) {

*if* (waitFlag[Integer.parseInt(processNames.get(initProcessName)) - 1]) {

                                Thread.sleep(500 + rnd.nextInt(501));

                                msg = "REPLY-" + initProcessName + "," + pName + "," + senderProcessName;

                                byte buffreply[] = msg.getBytes();

                                DatagramPacket dgPacketSend = *new* DatagramPacket(buffreply, buffreply.length,

                                        InetAddress.getLocalHost(),

                                        (Integer.parseInt(processNames.get(senderProcessName))) \* 1000);

                                dgSocket.send(dgPacketSend);

                                System.out.println("REPLY(" + initProcessName + "," + pName + "," + senderProcessName

                                        + ") sent to Process " + senderProcessName);

                            } *else* {

                                engagingQuerySender.add(senderProcessName);

                                waitFlag[Integer.parseInt(processNames.get(initProcessName)) - 1] = true;

                                num[Integer.parseInt(processNames.get(initProcessName)) - 1] = dependentSet.size();

*for* (int k = 0; k < dependentSet.size(); ++k) {

                                    Thread.sleep(500 + rnd.nextInt(501));

                                    msg = "QUERY-" + initProcessName + "," + pName + "," + dependentSet.get(k);

                                    byte buffsend[] = msg.getBytes();

                                    DatagramPacket dgPacketSend = *new* DatagramPacket(buffsend, buffsend.length,

                                            InetAddress.getLocalHost(),

                                            (Integer.parseInt(processNames.get(dependentSet.get(k)))) \* 1000);

                                    dgSocket.send(dgPacketSend);

                                    System.out.println(

                                            "QUERY(" + initProcessName + "," + pName + "," + dependentSet.get(k)

                                                    + ") sent to Dependent Process " + dependentSet.get(k));

                                }

                            }

                        }

*if* (msgType.equals("REPLY")) {

*if* (waitFlag[Integer.parseInt(processNames.get(initProcessName)) - 1]) {

                                num[Integer.parseInt(processNames.get(initProcessName)) - 1] -= 1;

*if* (num[Integer.parseInt(processNames.get(initProcessName)) - 1] == 0) {

*if* (initProcessName.equals(receiverProcessName)) {

                                        Thread.sleep(500 + rnd.nextInt(501));

                                        System.out.println("\n!!!!!   Deadlock Detected   !!!!!");

                                    } *else* {

*for* (int m = 0; m < engagingQuerySender.size(); ++m) {

                                            Thread.sleep(500 + rnd.nextInt(501));

                                            msg = "REPLY-" + initProcessName + "," + pName + ","

                                                    + engagingQuerySender.get(m);

                                            byte buffreply[] = msg.getBytes();

                                            DatagramPacket dgPacketSend = *new* DatagramPacket(buffreply,

                                                    buffreply.length, InetAddress.getLocalHost(),

                                                    (Integer.parseInt(processNames.get(engagingQuerySender.get(m))))

                                                            \* 1000);

                                            dgSocket.send(dgPacketSend);

                                            System.out.println("REPLY(" + initProcessName + "," + pName + ","

                                                    + engagingQuerySender.get(m) + ") sent to Process "

                                                    + engagingQuerySender.get(m) + " which sent the engaging query");

                                        }

                                    }

                                }

                            }

                        }

*//System.out.println("Round");*

                    }

                } *else*

                {

                    System.out.println("No msg");

                }

            } *catch* (IOException *ex*) {

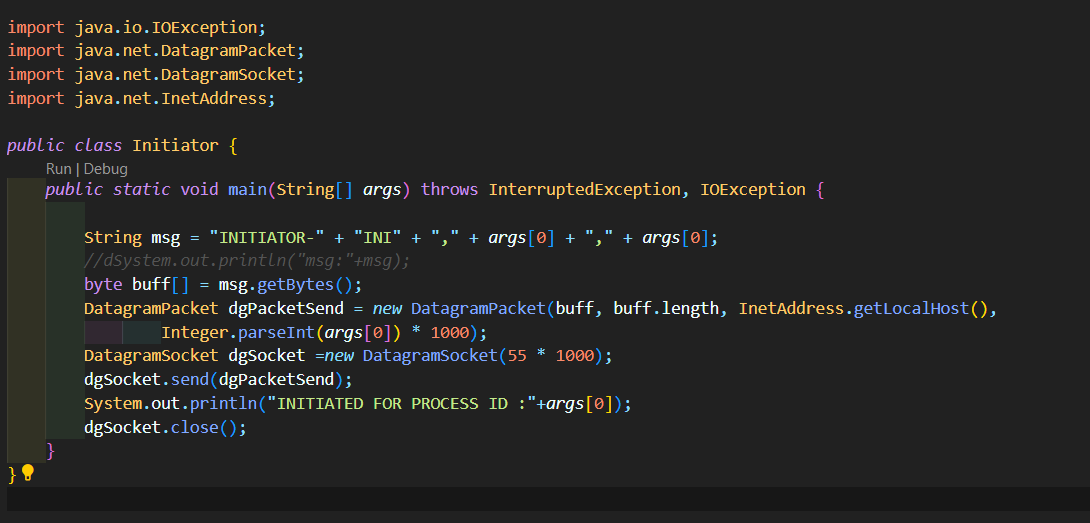
                System.err.println(ex);

            }

        }

    }

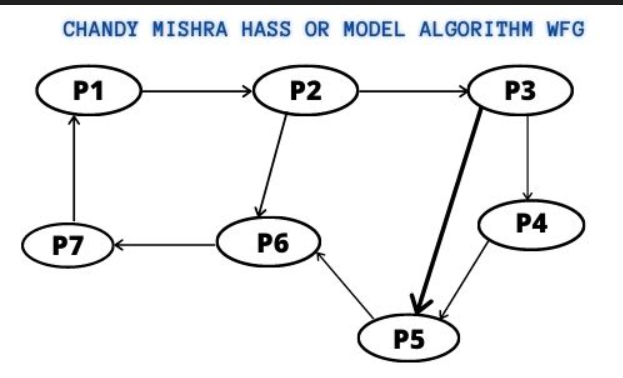
1. **Initiator.java**

****

**Working / Outputs:**

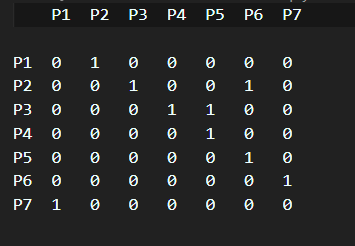
**CASE – 1: (DEADLOCK)**

**Input WFG:**

****

**From the WFG We can find there is Deadlock, now we will try to show it using the CODE (Algorithm)**

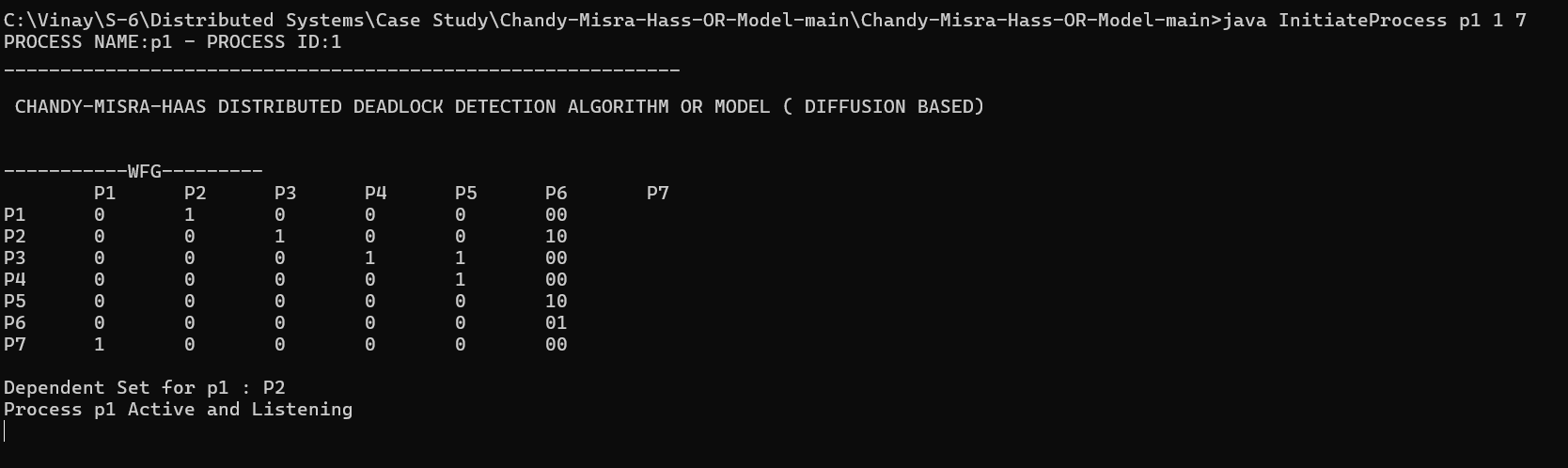
**Dependencies:**

****

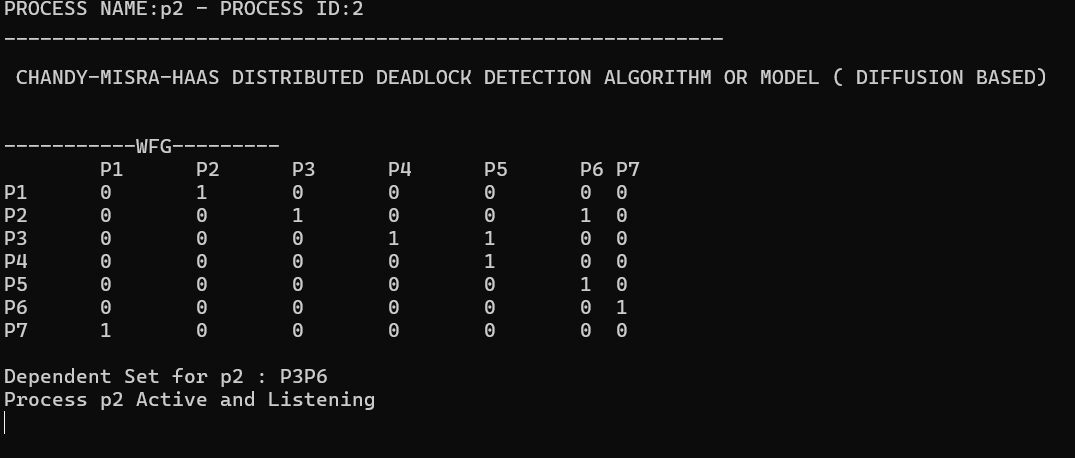
The code consists of a process class named Process.java and the Process is initiated by class InitiateProcess.java. The class InitiateProcess.java takes 3 command line input for process name, process ID and total No of process. **The use should use the same Process name as defined in Dependencies.txt i.e Process 1 is named P1 and ID is 1.**

**Java InitiateProcess P1 1 7 (Like this Create for every Process)**

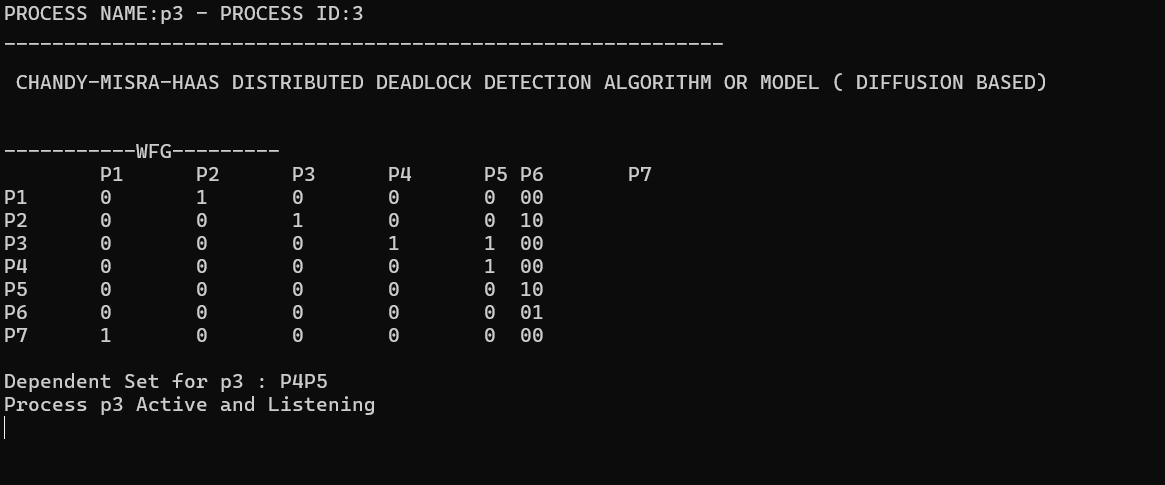
For P1



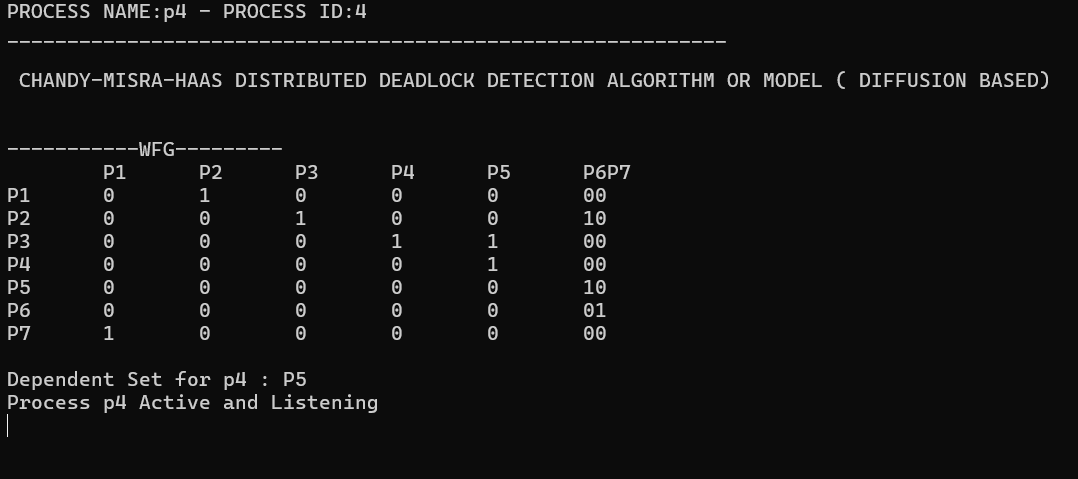
For P2



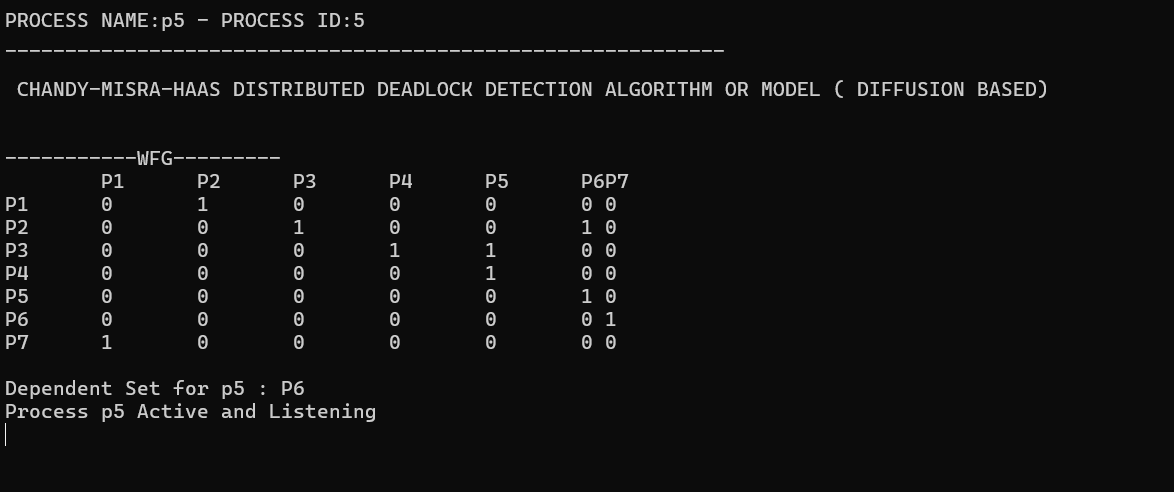
For P3



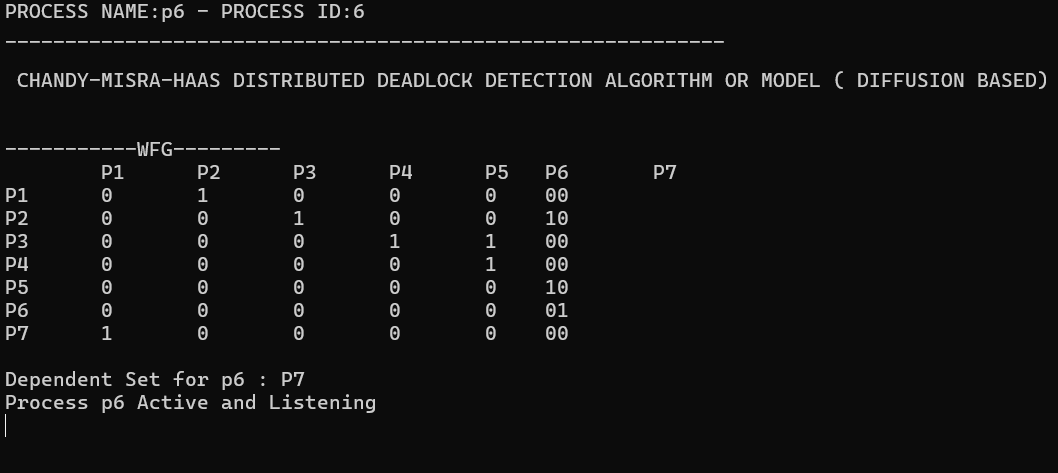
For P4



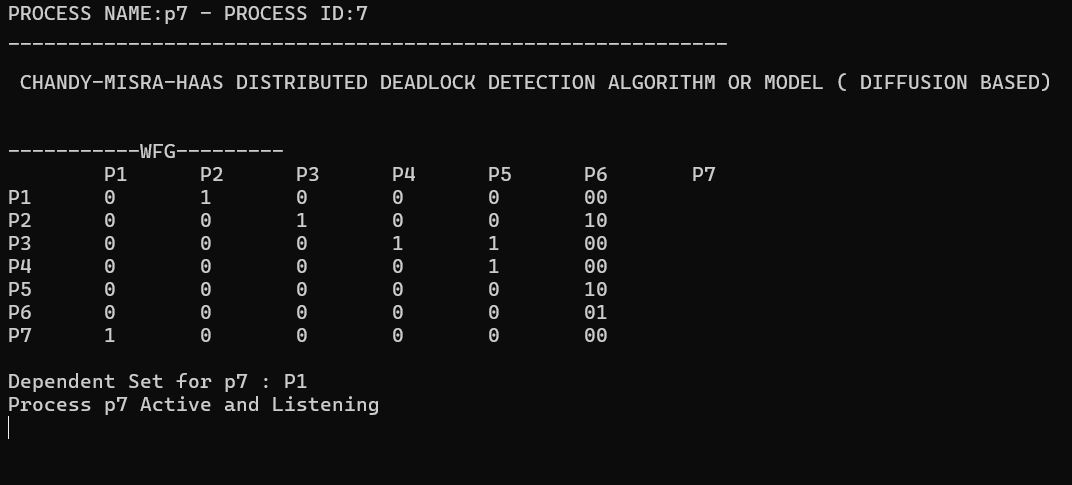
For P5



For P6



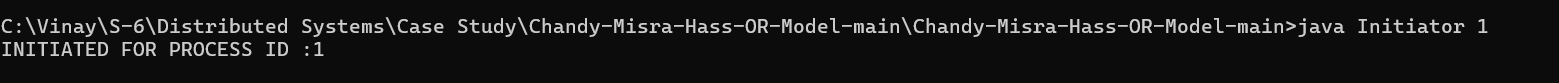
For P7



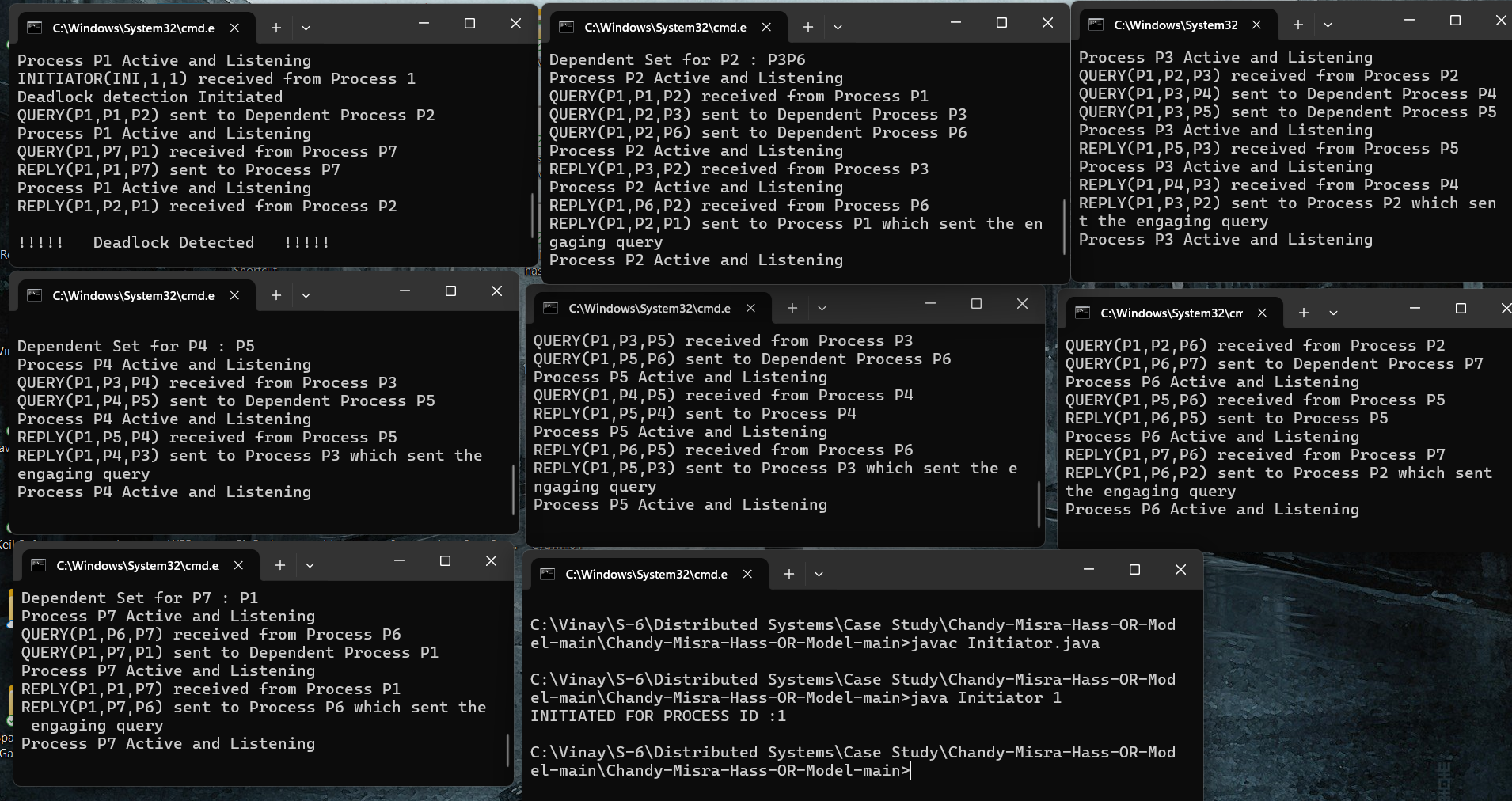
**Now We Have Initialised and Created WFG**

**We can Start Initiating a Process and try to DETECT DEADLOCK!!!**

**We can Start Initiating Process ID : 1**

****

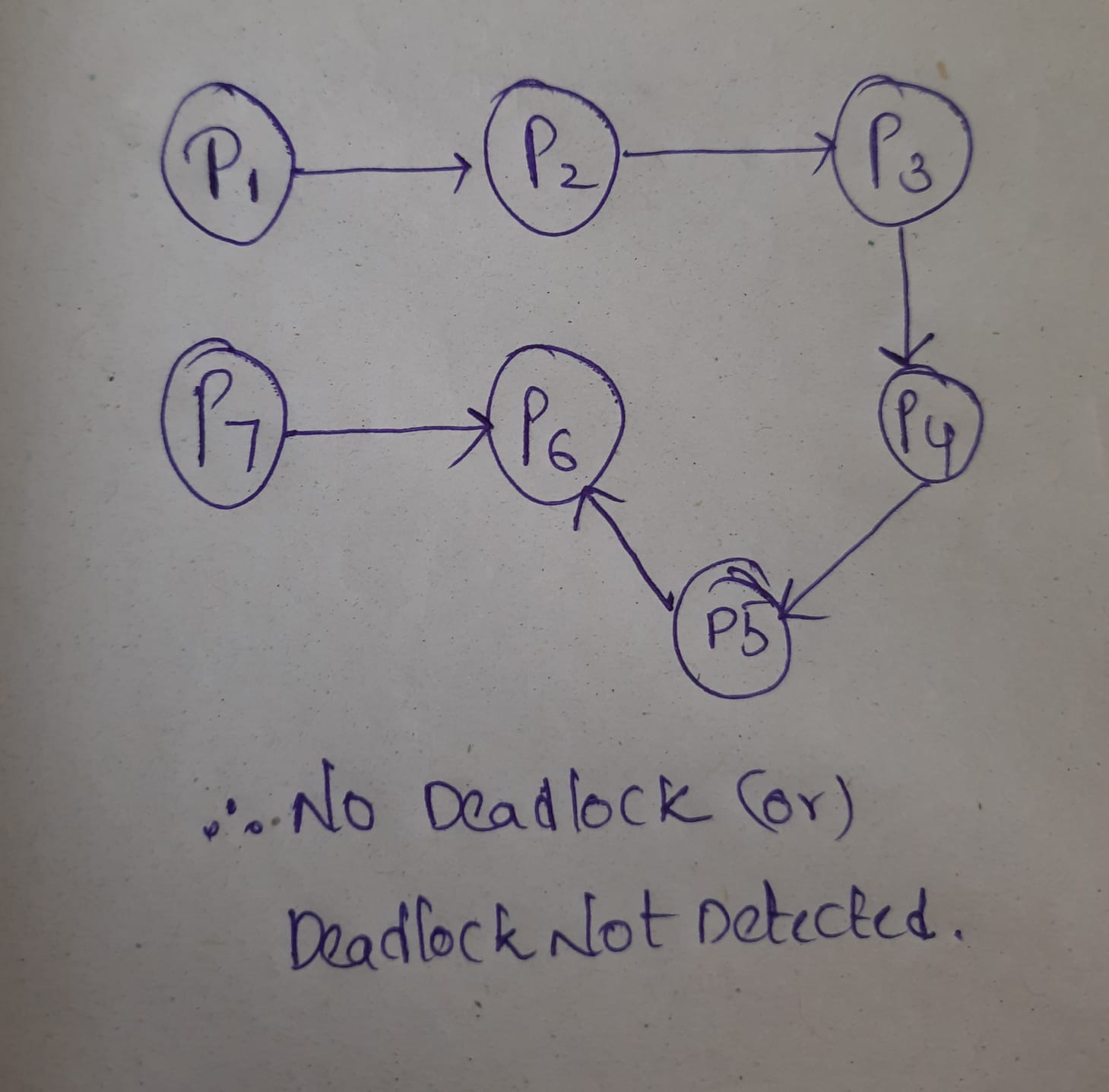
**The OUTPUT:**

****

**WE CAN SEE P1 DETECTED DEADLOCK! AND ALGORITHM WORKS…**

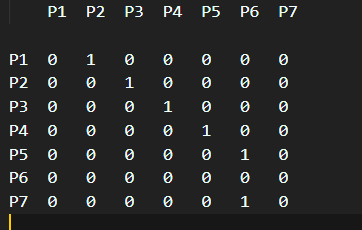
**CASE – 2: (NO DEADLOCK)**

**Input WFG:**

****

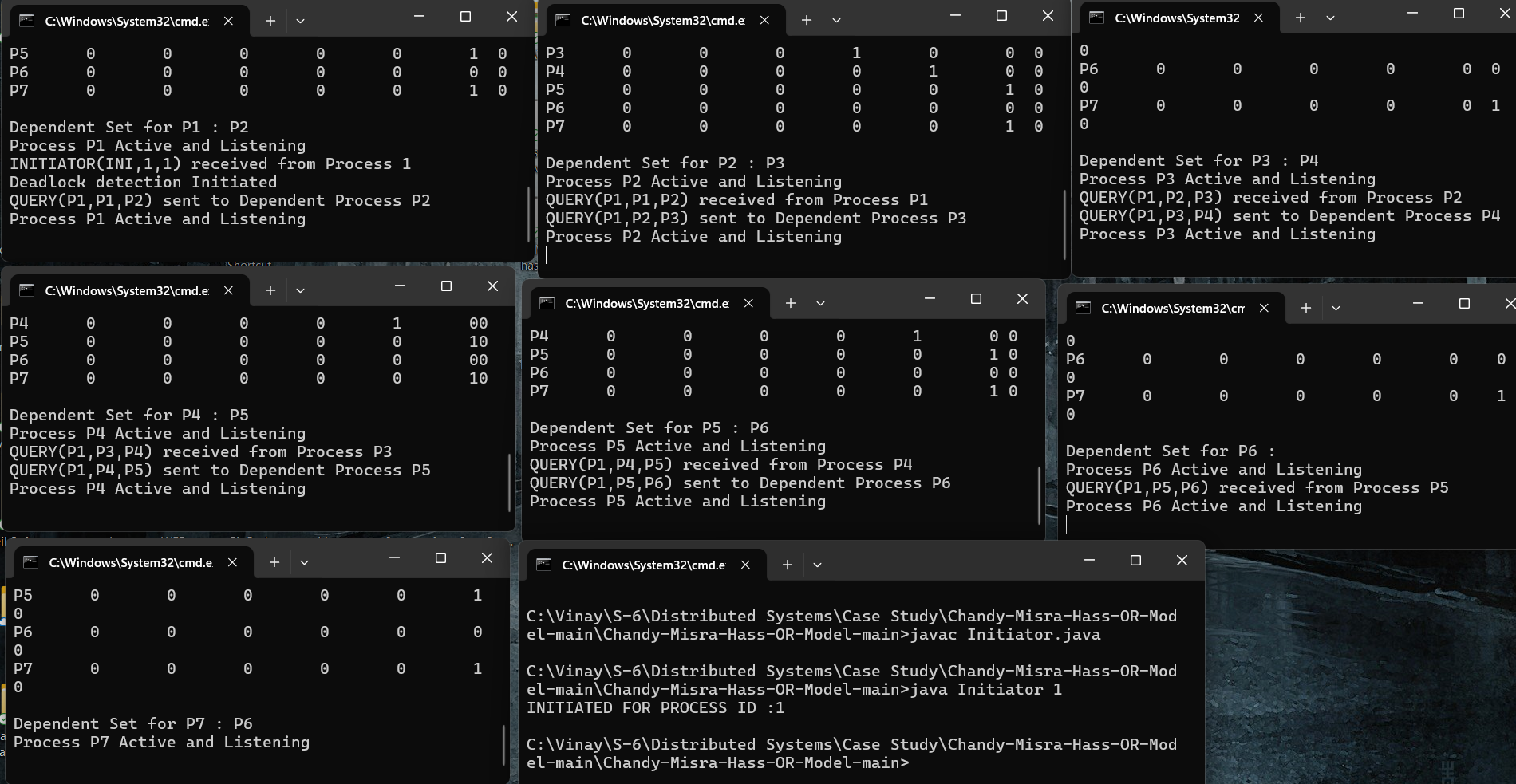
**From the WFG We can find there is Deadlock, now we will try to show it using the CODE (Algorithm)**

**Dependencies:**

****

**RUN THE CODE SAME AS BEFORE …...AND Initiate Process P1 to Detect the DEADLOCK**

**THE OUTPUT:**

****

**Since Every Process Did Not Receive Reply from all the Sent Queries It will be Kept Waiting … and from this We can Say Num Value will be not Equal to ZERO ... SO Deadlock is Not Detected ...**

**HENCE the Initiator Did Not Detect the DEADLOCK …!**

**REFERENCES**

**<https://www.geeksforgeeks.org/chandy-misra-haass-distributed-deadlock-detection-algorithm/>**

[**https://www.cs.fsu.edu/~xyuan/cop5611/lecture10.html**](https://www.cs.fsu.edu/~xyuan/cop5611/lecture10.html)