Sanjana Asrani
D17/01/Batch B

DC 06
CHANDY MISRA HASS

AIM:
Program to demonstrate Chandy Mishra Hass algorithm for Deadlock Management in Distributed Systems.

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.

Terminologies:
- Dependent process:

A process Pi is said to be dependent on some other process Pj, if there exists a sequence of processes Pi, Pi1, Pi2, Pi3…, Pim, Pj such that in the sequence, each process except Pj is blocked and each process except Pi holds a resource for which previous process in the sequence is waiting.

- Locally dependent process:

A process Pi is said to be locally dependent on some other process Pj if the process Pi is dependent on process Pj and both are at same site.
Data structures:
A boolean array, dependenti. Initially, dependenti[j] is false for all value of i and j. dependenti[j] is true if process Pj is dependent on process Pi.

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.

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).

Advantages: There is no need for special data structure. A probe message, which is very small and involves only 3 integers and a two dimensional boolean array dependent is used in the deadlock detection process. At each site, only a little computation is required and overhead is also low. Unlike other deadlock detection algorithm, there is no need to construct any graph or pass nor to pass graph information to other sites in this algorithm. Algorithm does not report any false deadlock (also called phantom deadlock).

Disadvantages:
The main disadvantage of distributed detection algorithms is that all sites may not be aware of the processes involved in the deadlock which makes resolution difficult. Also, proof of correction of the algorithm is difficult. It may detect a false deadlock if there is a delay in message passing or if a message is lost. This can result in unnecessary process termination or resource preemption. It may not be able to detect all deadlocks in the system, especially if there are hidden deadlocks or if the system is highly dynamic. It is complex and difficult to implement correctly. It requires careful coordination between the processes, and any errors in the implementation can lead to incorrect results.
It may not be scalable to large distributed systems with a large number of processes and resources. As the size of the system grows, the overhead and complexity of the algorithm also increase.

CODE:
```
import java.io.*;
public class
ChandyMisraHaasAlgo {
public static int flag = 0;
public static void
main(String args[]) throws
Exception {
BufferedReader ob = new
BufferedReader(new
InputStreamReader(System.in));
int init, aa, bb, x = 0, end
= 5;
File input = new
File("Dependencymatrix.txt");
@SuppressWarnings("resource")
BufferedReader in = new
BufferedReader(new
InputStreamReader(new
FileInputStream(input)));
String line;
int[][] a = new int[end][end];
//This code reads the
dependency matrix from the
file line by line and stores
it into the 2D array a. It
skips the first two lines
because they contain headers.
line = in.readLine();
line = in.readLine();
while ((line =
in.readLine()) != null) {
```

```
aa = 3;
bb = 4;
for (int y = 0; y < end; y++)
{
a[x][y] =
Integer.parseInt(line.substri
ng(aa, bb));
aa += 2;
bb += 2;
}
x++;
}
System.out.println();
System.out.println("    Chandy
Mishra  Hass  algorithm  for
Deadlock Management");
System.out.println();
//This  code  prints  the
dependency matrix in tabular
format.
System.out.println("\tS1\tS2\
tS3\tS4\tS5");
for (int i = 0; i < end; i++)
{
System.out.print("S" + (i + 1)
+ "\t");
for (int j = 0; j < end; j++)
{
System.out.print(a[i][j] +
"\t");
}
System.out.println();
}
System.out.println();
System.out.print("Enter
Initiator Site No. : ");
init =
Integer.parseInt(ob.readLine()
);
int j = init - 1;
System.out.println();
System.out.println();
System.out.println("
DIRECTION\tPROBE");
System.out.println();
for (int k = 0; k < end; k++)
{
if (a[j][k] == 1) {
System.out.println(
" S" + (j + 1) + " --> S" +
(k + 1) + "       (" + init +
"," + (j + 1) + "," + (k + 1)
+ ")");
aman(a, j, k);
}
}
//If flag remains 0, it means
no deadlock was detected, so
it  prints  a  message
indicating no deadlock was
detected.
if (flag == 0) {
System.out.println();
System.out.println("       NO
DEADLOCK DETECTED");
}
ob.readLine();
}
public  static  void
aman(int[][] a, int init, int
k) {
int end = 5;
for (int x = 0; x < end; x++)
{
if (a[k][x] == 1) {
if (init == x) {
System.out.println(" S" + (k
+ 1) + " --> S" + (x + 1) +
"      (" + (init + 1) + "," +
(k + 1) + ","
+ (x + 1) + ")" + " -------->
DEADLOCK DETECTED");
flag = 1;
break;
}
System.out.println(" S" + (k
+ 1) + " --> S" + (x + 1) +
"      (" + (init + 1) + "," +
(k + 1) + ","
+ (x + 1) + ")");
aman(a, init, x);
}}}}
```

OUTPUT:

```
PS D:\Engineering_codes\Div-B_01_Sanjana Asrani\sem 8\DC\6-ChandyMisraHas> java ChandyMisraHaasAlgo

 Chandy Mishra Hass algorithm for Deadlock Management

        S1      S2      S3      S4      S5
S1      0       1       0       0       0
S2      0       0       1       0       0
S3      0       0       0       1       0
S4      0       0       0       0       1
S5      1       0       0       0       0

Enter Initiator Site No. : 2


 DIRECTION      PROBE

 S2 --> S3     (2,2,3)
 S3 --> S4     (2,3,4)
 S4 --> S5     (2,4,5)
 S5 --> S1     (2,5,1)
 S1 --> S2     (2,1,2) --------> DEADLOCK DETECTED
```

CONCLUSION:

Thus implementation of chandy misra hass algorith using edge chasing to identify deadlocks in distributed systems has been done with by reading a dependency matrix and traversing it according to the algorithm's logic, the program effectively detects deadlocks and provides insights into the paths leading to them. With minimal overhead and simple data structures, it effectively detects deadlocks while mitigating false positives. However, challenges persist in scalability, potential false deadlocks, and complex implementation.