EXPERIMENT NO : 08

AIM: Implement Deadlock management in Distributed systems.

Theory:

Chandy-Misra-Hass deadlock detection algorithm:

Chandy-Misra-Haas's distributed deadlock detection algorithm is an edge chasing algorithm to detect deadlock in distributed systems. It is also considered one of the best deadlock detection algorithms for distributed systems.

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 the 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:
l . 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:
l . Set dependentk[i] to true.
2. Now, If k == i then, declare the Pi is deadlocked.
3. Else for all Pm and Pn check the 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 the above conditions satisfy.
Thus, the probe message travels along the edges of the 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:

The algorithm requires at the most exchange of messages to detect deadlock. Here, m is the 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 a 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 a distributed detection algorithms is that all sites may not aware of the processes involved in the deadlock this makes resolution difficult. Also, proof of correction of the algorithm is difficult.

Code:

exp8.py

```python
def aman(a, i, k):
    global flag
    end = 5
    for x in range(end):
        if a[k][x] == 1:
            if i == x:
                print(f'S{k+1} DEADLOCK DETECTED')
                flag = 1
                return
            print(f'S{k+1} ==> aman(a, i, x) ==> S{x+1} (Process {i+1}, {k+1}, {x+1})')
            aman(a, i, x)

flag = 0
a = [[0, 1, 0, 0, 0],
     [0, 0, 1, 0, 0],
     [0, 0, 0, 1, 1],
     [1, 0, 0, 0, 0],
     [0, 0, 0, 0, 0]]

j = int(input("Enter Initiator Site No. (0-4): "))
i = j + 1

print(f"\nInitiating deadlock detection from site {i}\n")
for k in range(len(a[j])):
    if a[j][k] == 1:
        print(f'S{j+1} ==> S{k+1} (Process {i}, {j+1}, {k+1})')
        aman(a, j, k)

if flag == 0:
    print("\nNO DEADLOCK DETECTED")
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

/bin/python3.9 /home/computer/exp/exp8.py
computer@computer-ThinkCentre:~/exp$ /bin/python3.9 /home/computer/exp/exp8.py
Enter Initiator Site No. (0-4): 1

Initiating deadlock detection from site 2

S2 ==> S3 (Process 2, 2, 3)
S3 ==> aman(a, i, x) ==> S4 (Process 2, 3, 4)
S4 ==> aman(a, i, x) ==> S1 (Process 2, 4, 1)
S1 DEADLOCK DETECTED
S3 ==> aman(a, i, x) ==> S5 (Process 2, 3, 5)
computer@computer-ThinkCentre:~/exp$
```

Conclusion:

Thus, we had Successfully Implemented Mutual Exclusion Algorithm