Name : Ajay Shitkar
Roll No : 58
Experiment No. 07
Implement Deadlock Management in Distributed System
Date of Performance: 18/03/24
Date of Submission: 01/04/24

Aim: To implement deadlock management in Distributed System

Objective: Develop a program to implement deadlock management in distributed system

Theory:

The algorithm allows a process to request for multiple resources at a time. Special messages or probes circulated along edges of WFG. Deadlock exists if the probe is received back by the initiator.

Working:

1. When a process that requests for a resource, fails to get it and times out,
2. It generates a special probe message and sends it to the process (or processes) holding the requested resource.

- The probe message (acts as a unique id) contains the following fields:
- The id of the process just blocked
- The id of the process sending this message
- The id of the process to whom this message is being sent

3. On receipt of the probe message, the receiver checks to see if it itself is waiting for any resource.
4. If not, this means that the recipient is using the resource requested by the process that sent the probe message to it.
5. In this case, the recipient simply ignores the probe message.
6. On the other hand, if the recipient is itself waiting for any resource, it passes the probe message to the resource holding process (or processes)
7. But, before the probe is forwarded, the recipient does following modifications
8. The first field is left unchanged
9. The recipient changes the second field to its own process id.
10. The third field is changed to the id of the process that will be the new recipient of the probe.
11. Every new recipient of the probe message repeats this procedure.
12. If the probe message returns back to the original sender,(the process whose id is the first field of the message), a cycle exists and, the system is deadlocked.

Code:

```
import java.util.*;
class Process {
int id;
boolean hasRequested;
boolean hasGranted;
boolean hasResponded;
List<Integer> requests;
List<Integer> replies;
Process(int id) {
this.id = id;
this.hasRequested = false;
this.hasGranted = false;
this.hasResponded = false;
this.requests = new ArrayList<>();
this.replies = new ArrayList<>();
}
}
class DeadlockDetection {
int numProcesses;
List<Process> processes;
```

```
public DeadlockDetection(int numProcesses) {
this.numProcesses = numProcesses;
this.processes = new ArrayList<>();
for (int i = 0; i < numProcesses; i++) {
processes.add(new Process(i));
}
}
public void request(int fromProcess, int toProcess) {
processes.get(toProcess).requests.add(fromProcess);
}
public void grant(int fromProcess, int toProcess) {
processes.get(fromProcess).replies.add(toProcess);
}
public void detectDeadlock() {
boolean[] marked = new boolean[numProcesses];
Arrays.fill(marked, false);
for (int i = 0; i < numProcesses; i++) {
for (int j : processes.get(i).requests) {
if (!processes.get(j).hasGranted) {
System.out.println("Deadlock detected involving processes
" + i + " and " + j);
marked[i] = true;

marked[j] = true;

}

}

}

for (int i = 0; i < numProcesses; i++) {

if (marked[i]) {

System.out.println("Process " + i + " is in deadlock.");

}

}

}

public static void main(String[] args) {

DeadlockDetection detector = new DeadlockDetection(4);

// Simulating requests and grants

detector.request(0, 1);

detector.request(1, 2);

detector.request(2, 3);

detector.request(3, 0);

detector.grant(1, 0);

detector.grant(2, 1);

detector.grant(3, 2);
detector.grant(0, 3);
// Detecting deadlock
detector.detectDeadlock();
}
}
```

Output :

```
java -cp /tmp/jrBid9a9X6 DeadlockDetection
Deadlock detected involving processes 0 and 3
Deadlock detected involving processes 1 and 0
Deadlock detected involving processes 2 and 1
Deadlock detected involving processes 3 and 2
Process 0 is in deadlock.
Process 1 is in deadlock.
Process 2 is in deadlock.
Process 3 is in deadlock.
```

Conclusion: Effective deadlock management in distributed systems is paramount for maintaining system reliability and performance. By employing techniques such as deadlock detection, prevention, and avoidance, distributed systems can mitigate the risk of deadlock occurrences. However, it's crucial to strike a balance between overhead and effectiveness when implementing deadlock management strategies. Continuous monitoring and adaptation are essential to address evolving system dynamics and ensure the smooth operation of distributed environments. Overall, proactive deadlock management enhances system resilience and contributes to overall system stability and efficiency.

.