DEADLOCK DETECTION AND RECOVERY

- SRADHA KEDIA

INTRODUCTION TO DEADLOCK

A deadlock can occur in almost any situation where processes share resources. It can happen in any computing environment, but it is widespread in distributed systems, where multiple processes operate on different resources. In this situation, one process may be waiting for a resource already held by another process.

Dining philosopher problem is one of the famous problems of Deadlock.

DEADLOCK DETECTION AND RECOVERY

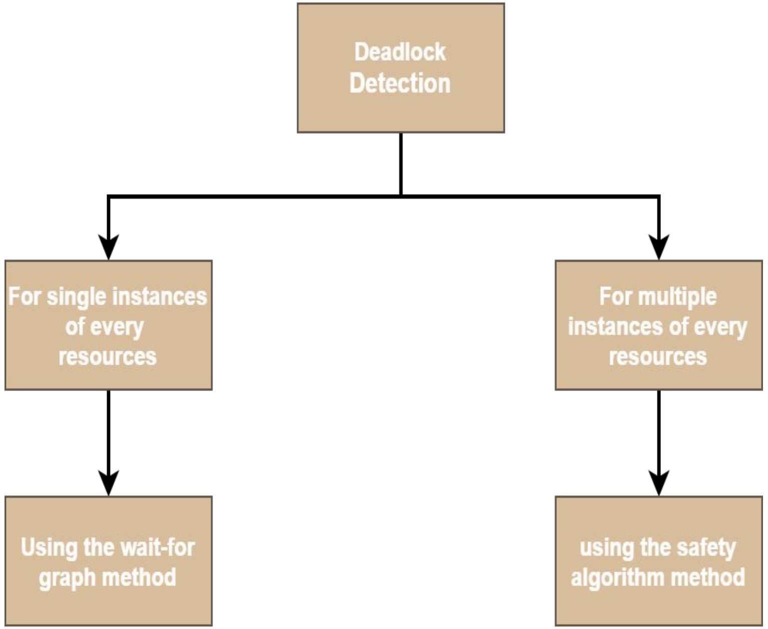
If a system does not employ either a deadlock-prevention or deadlock-avoidance algorithm, then there are chances of occurrence of a deadlock.

In this case, the system may provide two things:

- An algorithm is used to examines the state of the system in order to determine whether a deadlock has occurred.
- An algorithm that is used to recover from the deadlock.

Thus order to get rid of deadlocks the operating system periodically checks the system for any deadlock. After Finding the deadlock the operating system will recover from it using recovery techniques.

Now, the main task of the operating system is to detect the deadlocks and this is done with the help of Resource Allocation Graph.


Deadlock Detection
For single instances of every resources
For multiple instances of every resources
Using the wait-for graph method
using the safety algorithm method

SINGLE INSTANCE OF EACH RESOURCE TYPE

If all the resources have only a single instance, then a deadlock-detection algorithm can be defined that mainly uses the variant of the resource-allocation graph and is known as a wait-for graph. This wait-for graph is obtained from the resource-allocation graph by removing its resource nodes and collapsing its appropriate edges.

An edge from Pi to Pj in a wait-for graph simply implies that process Pi is basically waiting for process Pj in order to release a resource that it needs. An edge Pi, Pj exists in a wait-for graph if and only if the corresponding resource allocation graph contains two edges Pi,Rq and Rq,Pj for a resource Rq.

A deadlock exists in the system if and only if there is a cycle in the wait-for graph. In order to detect the deadlock, the system needs to maintain the wait-for graph and periodically system invokes an algorithm that searches for the cycle in the wait-for graph.

The algorithm that is used to detect the cycle in the graph mainly requires n^2 operations; where n indicates the number of vertices in the graph.

MULTIPLE INSTANCE OF EACH RESOURCE TYPE

The above scheme that is a wait-for graph is not applicable to the resource-allocation system having multiple instances of each resource type. Now we will move towards a deadlock detection algorithm that is is applicable for such systems.

This algorithm mainly uses several time-varying data structures that are similar to those used in Banker's Algorithm and these are as follows:

1. Available: It is an array of length m. It represents the number of available resources of each type.

2. Allocation: It is an n x m matrix which represents the number of resources of each type currently allocated to each process.

3. Request: It is an n*m matrix that is used to indicate the request of each process; if Request[i][j] equals to k, then process Pi is requesting k more instances of resource type Ri.

Allocation and Request are taken as vectors and referred to as Allocation and Request. The Given detection algorithm is simply used to investigate every possible allocation sequence for the processes that remain to be completed.

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:

    Work = Available

    Finish[i] =false for i = 0, 1, ... , n - 1.

If Allocationi is not equal to 0,then Finish[i] = false; else Finish[i] = true.

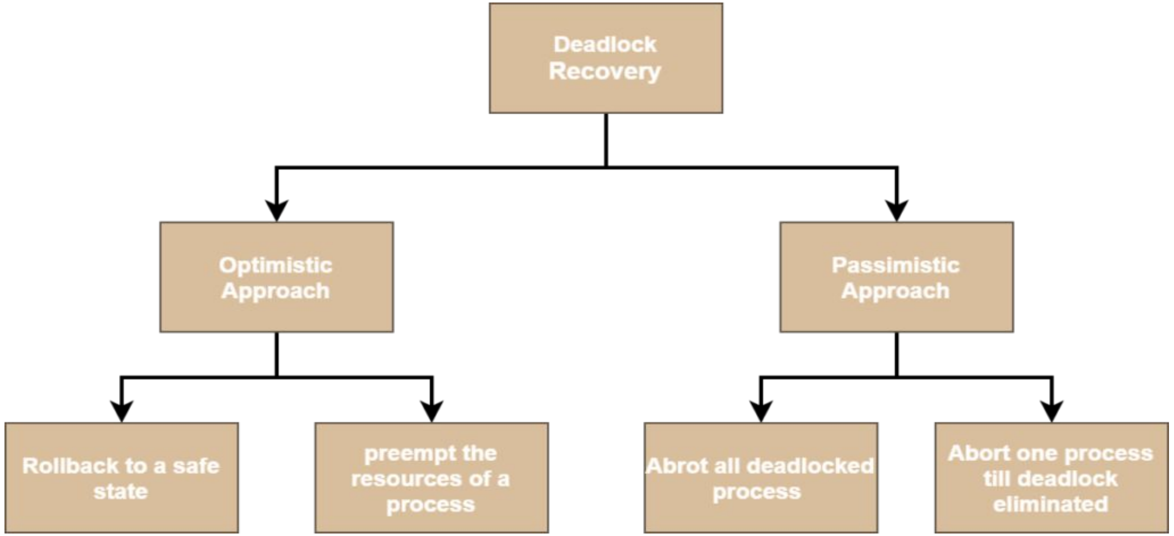
```
2. Find an index i such that both
        Finish[i] ==false
        Requesti <= Work
If no such i exist then go to step 4.

3. Perform the following:
        Work = Work + Allocation i
        Finish[i] = true
Go to step 2.
```

4. If If Finish[i] == false for some i, 0<=i<n, then it means the system is in a deadlocked state.Moreover,if Finish[i]==false,then process Pi is deadlocked.

This algorithm may require an order of mxn^2 operations in order to determine whether the system is in a deadlocked state.

RECOVERY FROM DEADLOCK

Now, as soon as the OS detects a deadlock, it'll start the recovery method. There are two main approaches to recover from a deadlock:


Deadlock Recovery
Optimistic Approach
Passimistic Approach
Rollback to a safe state
preempt the resources of a process
Abrot all deadlocked process
Abort one process till deadlock eliminated

OPTIMISTIC APPROACH

An example of the optimist approach is the resource and process preemption. In this approach, the OS will select some processes and preempt their resources, thereafter allocate them to other processes. Thus, the OS will try to break the circle.
One disadvantage of this approach is there is a possibility that the same process may become the victim of preemption. In this condition, the process will be stuck in starvation.

Another approach in which the OS will roll back to a certain safe state where deadlock hasn’t occurred. But for this, OS has to maintain some logs up to which it was in a safe state.

PESSIMIST APPROACH

One pessimist approach is to abort all the deadlocked processes. This is the simplest way of breaking the cycle to recover from the deadlock, but it’s also the most costly way of handling deadlock. In this method, we kill all the processes, and the OS will either discard them or restart a portion of the processes later as per requirement.
Also, we can abort the one process at a time till we eliminate deadlock from the system.

In this method, the OS kills one process at a time, and it selects the process which has done the least work. After that, it runs the deadlock detection algorithm to verify if the deadlock is recovered. If it is not recovered, then it will keep killing the processes till the deadlock is eliminated.