Methods Of Deadlock Handling

Banker's Algorithm

This algorithm can be used in a bank to ensure that the bank never allocates the available money in a way that it could no longer satisfy the needs of all its clients. A new task must declare the maximum number of instances of each resource type that it may need. This number should not exceed the total number of instances of that resource type in the system.
When a process requests a set of resources, the system must determine whether allocating these resources will leave the system in a safe state. If yes, then the resources may be allocated to the process. If not, then the process must wait till other processes release enough resources.

Data Structures for the Banker's Algorithm

Available: Vector of length m. If available [j] = k, there are k instances of resource type Rj available
Max: n x m matrix. If Max [i,j] = k, then process Pi may request at most k instances of resource type Rj
Allocation: n x m matrix. If Allocation[i,j] = k then Pi is currently allocated k instances of Rj
Need: n x m matrix. If Need[i,j] = k, then Pi may need k more instances of Rj to complete its task Need [i,j] = Max[i,j] – Allocation [i,j]
Finish: Boolean value, either true or false. If finish[i]=true for all i return safe else unsafe

Safety Algorithm
1.Let Available and Finish be vectors of length m and n, respectively.
 Initialize: Finish [i] = false for i = 0, 1, ..., n- 1

2.Find an i such that both: (a) Finish [i] = false (b) Needi Available If no such i exists, go to step 4
3. Available = Available + Allocation Finish[i] = true go to step 2
4.If Finish [i] == true for all i, then the system is in a safe state

Resource-request algorithm checks if a request can be safely granted. Pi is requesting for more resources and Request[m] be the request.

```
1. If Request > Need[i], then error
2. If Request > Available, then wait
 3. Pretend to allocate the request
        Available = Available - Request
        Allocation[i] = Allocation[i] + Request
        Need[i] = Need[i] - Request
```

If the resultant state is safe then the resources are actually allocated, else values of Available, Allocation[i] and Need[i] are restored to their previous values.
Time complexity = O(m).

Ostrich Algorithm

- The simplest approach is the ostrich algorithm: stick your head in the sand and pretend there is no problem at all.
- Different people react to this strategy in different ways. Mathematicians find it totally unacceptable and say that deadlocks must be prevented at all costs.
- Engineers ask how often the problem is expected, how often the system crashes for other reasons, and how serious a deadlock is.
- If deadlocks occur on the average once every five years, but system crashes due to hardware failures, compiler errors, and operating system bugs occur once a week, most engineers would not be willing to pay a large penalty in performance or convenience to eliminate deadlocks.
- Most operating systems, including UNIX and Windows, just ignore the problem on the assumption that most users would prefer an occasional deadlock to a rule restricting all users to one process, one open file, and one of everything.
- If deadlocks could be eliminated for free, there would not be much discussion. The problem is that the price is high, mostly in terms of putting inconvenient restrictions on processes, as we will see shortly. Thus we are faced with an unpleasant trade-off between convenience and correctness, and a great deal of discussion about which is more important, and to whom. Under these conditions, general solutions are hard to find.

Resource preEmption

To eliminate deadlocks using resource preemption, preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.
There are 3 methods to eliminate the deadlocks using resource preemption. These are :
a) SELECTING A VICTIM : Select a victim resource from the deadlock state, and preempt that one.
b) ROLLBACK : If a resource from a process is preempted, what should be done with that process. The process must be roll backed to some safe state and restart it from that state.
c) STARVATION : It must be guaranteed that resources will not always be preempted from the same process to avoid starvation problem.