For Mca 2nd semester
CS-23 operating system and shell programming
Unit 2

Deadlock Detection

Single instance of each resource type

If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource allocation graph, called a wait graph.

```
1)  Let work and finish be vectors of length m and n respectively. Initialise work = Available .
    For i = 1,2,....,n if Allocation i != 0 then finish[i] := false ;
    Otherwise, finish [i] := true;


2) Find an index i such that both
    a) finish[i]= false
    b) request i <= work
If no such i exists, go to step 4.


3)work:= work+Allocation;
   finish[i] =true
   Go to step 2

4) If finish[i] =false, for some I, 1<=i<n , then the system is in deadlock state.
```

Moreover, if finish [i]= false, then process P is deadlock.

Recovery from Detection

Process Termination:

* Abort all deadlocked processes : This. Ethos clearly will break the deadlock cycle bat at a frat expense. These processes may have computed for a long time and the results of these partial computations must be discarded and probably recomputed later.
* Abort one process at a time until the deadlock cycle is eliminated. This method incurs considerable overhead, since after each process is aborted a deadlock detection algorithm must be invoked to determine whether any processes are still deadlocked .
* Resource Preemtion : If preemption is required to deal with deadlocks, then three issues need to be addressed-

SELECTING A VICTIM : Which resources and which are to be preempted ? As in process termination, we must determine the order of preemption to minimise cost. Cost factors may include such parameters as the number of resources a deadlock process is holding and the amount of time a deadlock process has thus far consumed during its execution.
ROLLBACK: It is more effective to rollback the process only as fast as necessary to break the deadlock. This method required the system to keep more information about the state of all the running processes.

STARVATION: How it can be guaranteed that resources will not always be preempted from the same process? In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked up as victim. As a result this process never completes its task. So starvation situation needs to be dealt within any practical system.

Deadlock Characterisation

In a deadlock , processes neve finish executing and system resources are tied up, preventing other jobs from starting.

Necessary Conditions

A deadlock situation can arise if the following four conditions hold simultaneously in a system :

1) Mutual exclusion - At least one resource must be held in a non shareable mode . That is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
2) Hold and wait- A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other process.
3) No preemption- Resources can't br preempted. That is, a resource can be released only voluntarily by the process holding it after that process has completed its task.
4) Circular wait- A set { P0, P1,...Pn} of waiting processes must exist such that P0 is waiting for resource that is held by P1. P1 is waiting for a resource that is held by P2...Pn-1 is waiting for a resource that's held by Pn and Pn is waiting for a resource held by P0.

So all four conditions must hold for a deadlock to occur. The circular wait condition, imp,use the hold and wait condition. So the four conditions aren't completely independent.

Deadlock Prevention

By ensuring that at least one of the four necessary conditions can't hold , we can prevent the occurrence of deadlock.

1) Mutual exclusions- This condition use hole for non shareable resources. For example, a printer can't be simultaneously shared by several processes. Shareable resources on the other hand don't require mutually exclusive access and thus can't involve in a deadlock.
2) Hold and wait- To ensure that this condition never occurs in the system, we must guarantee that, whenever a process requests a resources , it does not

hold any other resources . One protocol that can be used requires each process to request and be allocated all its resources before it begins execution.

3) No Preemption- To ensure that this condition does not hold , we an use the following protocol. If a process is holding same resources and requests another resources that cannot be immediately allocated to it, then all resources currently being held are preempted. In other words, these resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
4) Circular Wait- One way to ensure that this condition never holds is to impose a total ordering of all resources and to require that each process requests resources in an increasing order of enumeration.

Let R= { R1, R2,...Rn} be the set of resources types. We assign each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our in our ordering.
If set of resources of types R includes tape drive, disk drives and printers , then the function F might be defined as follows :

F(tape drive)=1
F(disk drive)=5
F(printer)=12

Let the set of processes involved that a circular wait condition cannot hold. We can demonstrate this fact by assuming that a circular exist. Let the set involved in the circular wait be { P0, P1,...Pn} where Pi is waiting for a resource Ri which is held by process Pi+1. Since process Pi+1 is holding resource R, while requesting resource Ri+1, we must have F(R1) < F(Rn). By transitioning, F(R0)<F(R0) which is impossible. No circular wait!

By Mamta
Mobile no. 9430964676