Paper Name : Operating System

Prepared by : Debanjali Jana

Deadlock Detection –

In the absence of deadlock prevention or avoidance algorithm, deadlock may occur in the system. So system must provide deadlock detection algorithm to detect deadlock from time to time.

- An algorithm that examines the state of the system to determine whether a deadlock has occurred
- An algorithm to recover from the deadlock

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-for graph. We obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges. More precisely, an edge from Pi to Pj in a wait-for graph implies that process Pi is waiting for process Pj to release a resource that Pi 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 some resource Rq. To detect deadlocks, the system needs to maintain the wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph.

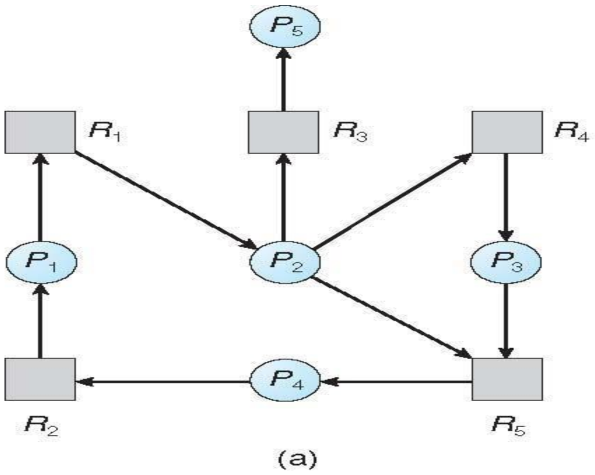

Fig: Resource-Allocation Graph

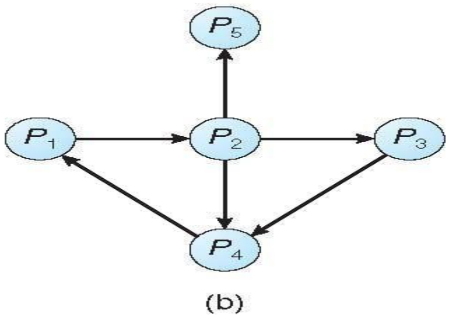

Fig: Corresponding wait-for graph

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system may provide:

- An algorithm that examines the state of the system to determine whether a deadlock has occurred
- An algorithm to recover from the deadlock

In the following discussion, we elaborate on these two requirements as they pertain to systems with only a single instance of each resource type, as well as to systems with several instances of each resource type. At this point, however, we note that a detection-and-recovery scheme requires overhead that includes not only the run-time costs of maintaining the necessary information and executing the detection algorithm but also the potential losses inherent in recovering from a deadlock.

Single Instance of a 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-for graph. We obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.

More precisely, an edge from Pi to Pj in a wait-for graph implies that process Pi is waiting for process Pj to release a resource that Pi 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 some resource Rq. As before, a deadlock exists in the system if and only if the wait-for graph contains a cycle. To detect deadlocks, the system needs to maintain the wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph. An algorithm to detect a cycle in a graph requires an order of n2 operations, where n is the number of vertices in the graph.

Several Instances of a Resource Type

The algorithm employs several time-varying data structures that are similar to those used in the banker's algorithm .

- **Available** - A vector of length m indicates the number of available resources of each type.
- **Allocation** - An n x m matrix defines the number of resources of each type currently allocated to each process.
- **Request -** An n x m matrix indicates the current request of each process. If Request[i][j] equals k, then process Pi is requesting k more instances of resource type Rj.

Consider a system with five processes P_0, P_1, P_2, P_3 and P_4 and three resources of type A, B, C. Resource type A has 7 instances, B has 2 instances and type C has 6 instances.

Process	**Allocation**	**Request**	**Available**
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

The safe sequence <P_0, P_2, P_3, P_1, P_4> will not result in deadlock.

Now, P_2 requests an additional instance of type C

Process	**Allocation**	**Request**	**Available**
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

The request of P_0 can be satisfied so P_0 can execute. As P_0 executes it will release it's resources. So it's allocated resources will be added to the available resources. So now available resources become 0 1 0. Now the system is in a state where the request of the processes P_1, P_2, P_3, and P_4 cannot be fulfilled with the available resources. Thus the processes P_1, P_2, P_3, and P_4 are in deadlock.

Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

Detection-Algorithm Usage-

When should we invoke the detection algorithm? The answer depends on two factors:

1. How often is a deadlock likely to occur?
2. How many processes will be affected by deadlock when it happens?

If deadlocks occur frequently, then the detection algorithm should be invoked frequently. Resources allocated to deadlocked processes will be idle until the deadlock can be broken. In addition, the number of processes involved in the deadlock cycle may grow.

Invoking the deadlock-detection algorithm for every resource request will incur considerable overhead in computation time. A less expensive alternative is simply to invoke the algorithm at defined intervals-for example, once per hour or whenever CPU utilization drops below 40 percent. (A deadlock eventually cripples system throughput and causes CPU utilization to drop.) If the detection algorithm is invoked at arbitrary points in time, the resource graph may contain many cycles. In this case, we generally cannot tell which of the many deadlocked processes "caused" the deadlock

Recovery from Deadlock –

- **Process Termination:**

To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.

I. Abort all deadlocked processes - This method clearly will break the deadlock cycle, but at great expense; the deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.
II. 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 rnust be invoked to determine whether any processes are still deadlocked.

    In which order should we choose to abort?

    a. Priority of the process

    b. How long process has computed, and how much longer to completion

c. Resources the process has used

d. Resources process needs to complete

e. How many processes will need to be terminated

f. Is process interactive or batch?

- **Resource Preemption:**
- Selecting a victim – Which resources and which processes are to be pre-empted? minimize cost
- Rollback – If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution as it is missing some required resource. We must rollback the process to some safe state, restart process for that state
- Starvation – same process may always be picked as victim, include number of rollback in cost factor