Operating System

Deadlocks

UNIT-IV

Prepared By
Alok Haldar
Assistant professor
Department of Computer Science & BCA
Kharagpur College

Banker’s Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

Let *n* = number of processes, and *m* = number of resources types.

Available : A vector of length m indicates the number of available resources of each type. If Available[j] = k, then k instances of resource type Rj are available.

• **Max** : An n × m matrix defines the maximum demand of each process. If Max[i][j] equals k, then process P i may request at most k instances of resource type Rj .

• **Allocation :** An n × m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process Pi is currently allocated k instances of resource type Rj .

• **Need :** An n × m matrix indicates the remaining resource need of each process. If Need[i][j] = k, then process Pi may need k more instances of resource type Rj to complete its task.

Note that Need[i][j] = Max[i][j] − Allocation[i][j].

Safety Algorithm

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

   Work = *Available*

   Finish [*i*] = *false* for *i* = 1,2,3, …, *n.*

2. Find and *i* such that both:

   (a) *Finish* [*i*] = *false*

   (b) $Need_i \leq Work$

   If no such *i* exists, go to step 4.

3. $Work = Work + Allocation_i$

   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 for Process P_i

Let $Request_i$ be the request vector for process P i . If $Request_i$ [j] == k, then process Pi wants k instances of resource type Rj . When a request for resources is made by process Pi ,

then the following actions are taken:

1. *If $Request_i \leq Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.*
2. *If $Request_i \leq$ Available, go to step 3. Otherwise, Pi must wait, since the resources are not available.*
3. *Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:*

$$Available = Available - Request_i ;$$

$$Allocation_i = Allocation_i + Request_i ;$$

$$Need_i = Need_i - Request_i ;$$

If the resulting resource-allocation state is safe, the transaction is completed, and process P i is allocated its resources. However, if the new state is unsafe, then P i must wait for Request i , and the old resource-allocation state is restored.

Example of Banker's Algorithm

consider a system with five processes P0 through P4 and three resource types A, B, and C. Resource type A has ten instances, resource type B has five instances, and resource type C has seven instances.
Suppose that, at time T0 , the following snapshot of the system has been taken:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Example (Cont.)

The content of the matrix Need is defined to be Max − Allocation and is as follows:

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

Example P_1 Request (1,0,2) (Cont.)

Suppose now that process P1 requests one additional instance of resource type A and two instances of resource type C, so $Request_1 = (1,0,2)$.

To decide whether this request can be immediately granted, we first check that $Request_1 \le Available$—that is, $(1,0,2) \le (3,3,2)$, which is true.

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Executing safety algorithm shows that sequence $<P_1, P_3, P_4, P_0, P_2>$ satisfies safety requirement.

Can request for (3,3,0) by P_4 be granted?

Can request for (0,2,0) by P_0 be granted?

Deadlock Detection

Allow system to enter deadlock state

Detection algorithm

Recovery scheme

Several Instances of a Resource Type

- Available : A vector of length m indicates the number of available resources of each type.
- Allocation : An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- Request : An $n \times m$ matrix indicates the current request of each process. If Request[i][j] = k, then process P_i is requesting k more instances of resource type R_j .

Detection Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available. For i = 0, 1, ..., n–1, 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. $Request_i \le Work$

   If no such i exists, go to step 4.
3. Work = Work + Allocation i
   Finish[i] = true
   Go to step 2.
4. If Finish[i] == false for some i, $0 \le i < n$, then the system is in a deadlocked state. Moreover, if Finish[i] == false, then process P i is deadlocked.

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm

- Five processes P_0 through P_4; three resource types A (7 instances), *B* (2 instances), and *C* (6 instances).
- Snapshot at time T_0:

	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	

- Sequence <P_0, P_2, P_3, P_1, P_4> will result in *Finish*[*i*] = true for all *i*.

Example (Cont.)

- P_2 requests an additional instance of type C.

	Request
	A B C
P_0	0 0 0
P_1	2 0 1
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests.
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.

- Abort one process at a time until the deadlock cycle is eliminated.

- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return 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.

Combined Approach to Deadlock Handling

- Combine the three basic approaches
  - prevention
  - avoidance
  - detection

  allowing the use of the optimal approach for each of resources in the system.

- Partition resources into hierarchically ordered classes.

- Use most appropriate technique for handling deadlocks within each class.