Department of Computer Science and Engineering

FACULTY OF ENGINEERING AND TECHNOLOGY
UNIVERSITY OF LUCKNOW
LUCKNOW

CS-501

Dr. Zeeshan Ali Siddiqui

Assistant Professor

Deptt. of C.S.E.

METHODS FOR HANDLING DEADLOCKS (Part-4)

Banker’s Algorithm

Methods for Handling Deadlocks

- *Deadlock Prevention*
- ***Deadlock Avoidance***
- *Deadlock Detection*
- *Ignore the problem*

DEADLOCK AVOIDANCE Continue…

AVOIDANCE ALGORITHMS

Avoidance algorithms

- Single instance of a resource type:
  - *Use a resource-allocation graph*

- Multiple instances of a resource type:
  - *Use the banker's algorithm*

BANKER’S ALGORITHM

Banker's Algorithm

- *Multiple instances* of resources of the same type.

- Each process must *a priori* claim maximum use.

- The algorithm allocates *resources* to a requesting thread if the allocation leaves the system in a safe state. Otherwise, the thread must wait.

- When a process gets all its resources it must return them in a *finite* amount of time.

Data Structure

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

  - **Available**: Vector of length m.
    - If available[j] = k, there are k instances of resource type Rj are 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	*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	

  - **Allocation**: n x m matrix.
    - If Allocation[i,j] = k then Pi is currently allocated k instances of Rj

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

  - **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]

Safety Algorithm

- *Safety algorithm* finds out whether or not a system is in a safe state.

1. Let *Work* and *Finish* be vectors of length m and n, respectively. Initialize $Work = Available$ and $Finish[i] = false$ for $i = 0, 1, ..., n - 1$.

2. Find an index 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

Let $Request_i$ be the request vector for process P_i. If $Request_i[j] == k$, then process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, 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, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

$$Available = Available - Request_i;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i;$$

- If *safe* -> the resources are allocated to Pi
- If *unsafe* -> Pi must wait, and the old resource-allocation state is restored

References

1. Silberschatz, Galvin and Gagne, “Operating Systems Concepts”, Wiley.

2. William Stallings, “Operating Systems: Internals and Design Principles”, 6th Edition, Pearson Education.

3. D M Dhamdhere, “Operating Systems: A Concept based Approach”, 2nd Edition, TMH.

Thank You.