TOPIC : DEADLOCK AVOIDANCE

PRESENTATION BY

Mrs.D.Beulah
Assistant Professor
Aditya Degree College, Kkd.

Deadlock Avoidance

- To avoid deadlocks, we need to know more information about how resources are to be requested.
- A deadlock avoidance algorithm dynamically examines the resource allocation state.
- A state is safe if the system allocates resources to each process in some order without deadlock.
- A system is in safe state if there exists a safe sequence.

Banker's Algorithm

Procedure :

- When a new process enters the system, it must declare the maximum
- number of instances of each resource type that it may need. This number must not exceed the total number of resources in the system.
- When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated else, the process must wait until some other process releases enough resources.

Data structures: (n is the number of processes in the system and m is the number of resource types)

Available : A vector of length m indicates the number of available resources of each type.

If Available[j] equals k, then k instances of resource type Rj are available.

Max : An n x m matrix defines the maximum demand of each process.

If Max[i] [j] equals k, then process Pi may request at most k instances of

resource type Rj.

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

Need : An n x m matrix indicates the remaining resource need of each process.

If Need[i][j] equals k, then process Pi may need k more instances of resource type Rj to complete its task.

Need[i][j] equals Max[i][j]- Allocation [i][j].

These data structures vary over time in both size and value.

- 5 processes - P0, P1, P2, P3 & P4.
- 3 resource types – A, B & C.
  - A – 10 instances
  - B – 5 instances
  - C – 7 instances
- Initially,

	Allocation			Max		
	A	B	C	A	B	C
P_1	0	1	0	7	5	3
P_2	2	0	0	3	2	2
P_3	3	0	2	9	0	2
P_5	2	1	1	2	2	2
P_4	0	0	2	4	3	3

3	3	2
A	B	C

From this, we can find need matrix.

Need[i][j] = Max[i][j] – Allocation[i][j].

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