SAVEETHA ENGINEERING COLLEGE

AUTONOMOUS

NAAC A GRADE

Affiliated to Anna University | Approved by AICTE

19CS405 OPERATING SYSTEMS

Topic – Deadlock Prevention, Deadlock Avoidance

Dr. K. Suresh Kumar,
Associate Professor,
Department of Information Technology.

UNIT II - PROCESS MANAGEMNET

Processes - Process Concept, Process Scheduling, Operations on Processes, Inter-process Communication; CPU Scheduling - Scheduling criteria, Scheduling algorithms, Multiple processor scheduling, Real time scheduling; Threads- Overview, Multithreading models, Threading issues; Process Synchronization - The critical-section problem, Synchronization hardware, Mutex locks, Semaphores, Classic problems of synchronization, Critical regions, Monitors; Deadlock – System model, Deadlock characterization, Methods for handling deadlocks, **Deadlock prevention, Deadlock avoidance**, Deadlock detection, Recovery from deadlock.

Deadlock Prevention

- ✓ ***Deadlock prevention*** means to block at least one of the four conditions required for **deadlock** to occur. If we are able to block any one of them then ***deadlock can be prevented***.

- **Prevent mutual exclusion**
  - use only sharable resources (e.g., a read-only file)
  - impossible for practical systems
- **Prevent hold and wait**
  - ✓ methods
    - Pre allocate
      - do not pick up one chopstick if you cannot pick up the other
      - for a process that copies data from DVD drive to a file on disk and then prints it from there:
        - request DVD drive
        - request disk file
        - request printer
      - all system calls requesting resources must proceed all other system calls

Deadlock Prevention

- a process can request resources only when it has none
  - request DVD drive and disk file
  - release DVD drive and disk file
  - request disk file and printer (no guarantee data will still be there)
  - release disk file and printer

✓ inefficient
✓ starvation possible

- **Prevent no pre-emption** (i.e., allow pre-emption, and permit the OS to take away resources from a process)
  - when a process must wait, it must release its resources
  - some resources cannot be feasibly pre-empted (e.g., printers, tape drives)

- **Prevent circular wait**
  - impose a total ordering on resources
  - only allow requests in an increasing order

Usually a deadlock prevention approach is simply unreasonable.

Deadlock Avoidance

- ✓ This requires that the system has some information available up front. Each process declares the maximum number of resources of each type which it may need. Dynamically examine the resource allocation state to ensure that there can never be a circular-wait condition.
- ✓ The system's resource-allocation state is defined by the number of available and allocated resources, and the maximum possible demands of the processes.
- ✓ When a process requests an available resource, the system must decide if immediate allocation leaves the system in a ***safe state***.
- ✓ The system is in a safe state if there exists a safe sequence of all processes:

  Sequence $< P_1, P_2, \ldots P_n >$ *is safe for the current allocation state if, for each* P_i, *the resources which* P_i *can still request can be satisfied by the currently available resources plus the resources held by all of the* P_j*'s, where* $j < i$.
- ✓ If the system is in a ***safe state***, there can be ***no deadlock***. If the system is in an ***unsafe state***, there is the ***possibility*** **of deadlock**.

Deadlock Avoidance Example

Example: consider a system with 12 magnetic tapes and 3 processes (P_0, P_1, and P_2):

available = 3			
Process	**Maximum Needs**	**Holding**	**Needs**
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

- ✓ Is the system in a safe state? If so, which sequence satisfies the safety criteria?

available = 2			
Process	**Maximum Needs**	**Holding**	**Needs**
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

- ✓ Is the system in a safe state? If so, which sequence satisfies the safety criteria?
- ✓ In this scheme, a process which requests a resource that is currently available, may still have to wait. Thus, resource utilization may be lower than it would otherwise be.

Deadlock Avoidance Algorithms

Two deadlock avoidance algorithms

- ✓ **Resource-allocation graph algorithm**
- ✓ **Banker's algorithm**

Resource-allocation graph algorithm

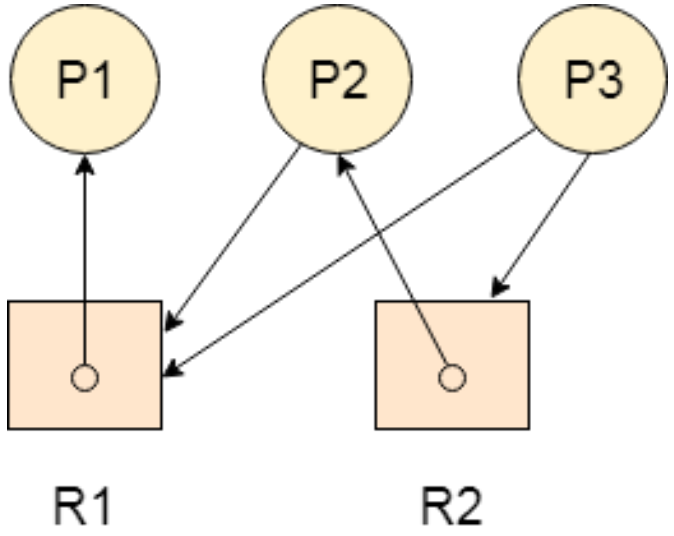
- ✓ only applicable when we only have 1 instance of each resource type
- ✓ claim edge (dotted edge), like a *future* request edge
- ✓ when a process requests a resource, the claim edge is converted to a request edge
- ✓ when a process releases a resource, the assignment edge is converted to a claim edge

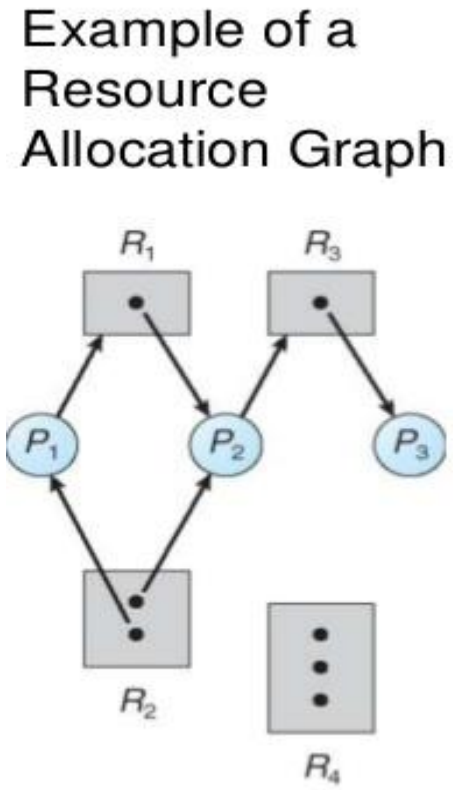
Banker's Algorithm

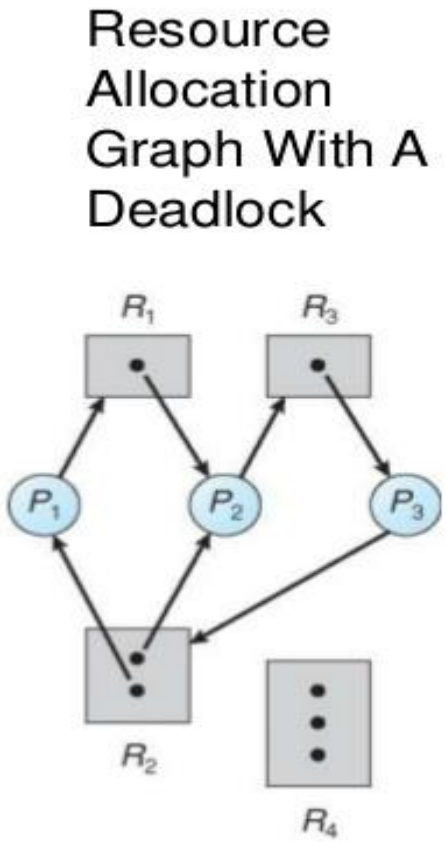
- ✓ A classic deadlock avoidance algorithm more general than resource-allocation graph algorithm (handles multiple instances of each resource type), but is less efficient.

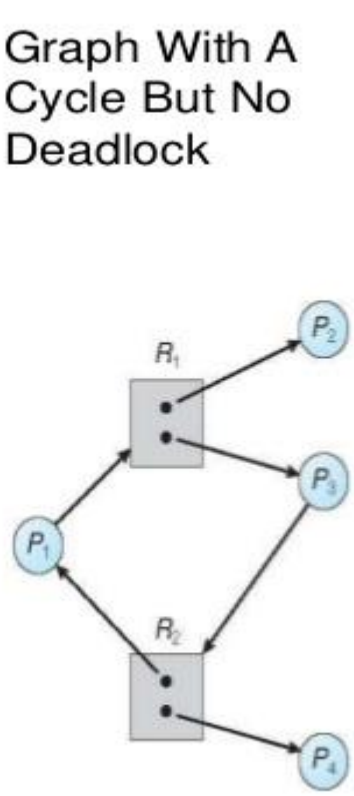
Deadlock Avoidance Algorithms

- ✓ Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.
- ✓ According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.
- ✓ The graph is deadlock free since no cycle is being formed in the graph.









Banker's Algorithms

We call Banker's algorithm when a request for R is made. Let n be the number of processes in the system, and m be the number of resource types.

Define:

- ✓ available[m]: the number of units of R currently unallocated (e.g., available[3] = 2)
- ✓ max[n][m]: describes the maximum demands of each process (e.g., max[3][1] = 2)
- ✓ allocation[n][m]: describes the current allocation status (e.g., allocation[5][1] = 3)
- ✓ need[n][m]: describes the *remaining* possible need (i.e., need[i][j] = max[i][j] - allocation[i][j])

Resource-request algorithm:

Define:

request[n][m]: describes the current outstanding requests of all processes (e.g., request[2][1] = 3)

1. If request[i][j] <= need[i][j], to to step 2; otherwise, raise an error condition.
2. If request[i][j] > available[j], then the process must wait.
3. Otherwise, pretend to allocate the requested resources to Pi :

    available[j] = available[j] - request[i][j]
    allocation[i][j] = allocation[i][j] + request[i][j]
    need[i][j] = need[i][j] - request[i][j]

Once the resources are allocated, check to see if the system state is safe. If unsafe, the process must wait and the old resource-allocated state is restored.

Deadlock Avoidance Algorithms

Safety algorithm (to check for a safe state):

1. Let **work** be an integer array of length m, initialized to **available**.
   Let **finish** be a Boolean array of length n, initialized to **false**.

2. Find an i such that both:
   1. finish[i] == false
   2. need[i] <= 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, otherwise unsafe.

Deadlock Avoidance Algorithms

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:

Process	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	

What will be the content of the Need matrix?

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

So, the content of Need Matrix is:

Process	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

Deadlock Avoidance Algorithms

Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm

Process	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	

Process	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

m=3, n=5 — Step 1 of Safety Algo

Work = Available

Work = | 3 | 3 | 2 |

	0	1	2	3	4
Finish =	false	false	false	false	false

For i = 0 — Step 2

$Need_0$ = 7, 4, 3 — 7,4,3 > 3,3,2

Finish [0] is false and $Need_0$ > Work

So P_0 must wait — But Need ≤ Work

For i = 1 — Step 2

$Need_1$ = 1, 2, 2 — 1,2,2 < 3,3,2

Finish [1] is false and $Need_1$ < Work

So P_1 must be kept in safe sequence

Step 3

Work = Work + $Allocation_1$ — 3, 3, 2 + 2, 0, 0

Work = | A: 5 | B: 3 | C: 2 |

	0	1	2	3	4
Finish =	false	true	false	false	false

For i = 2 — Step 2

$Need_2$ = 6 , 0, 0 — 6, 0, 0 > 5,3, 2

Finish [2] is false and $Need_2$ > Work

So P_2 must wait

Deadlock Avoidance Algorithms

Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,

Process	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	

Process	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

For i=3 — Step 2

$Need_3 = 0, 1, 1$

Finish [3] = false and $Need_3$ (0, 1, 1) < Work (5, 3, 2)

So P_3 must be kept in safe sequence

Step 3

$Work = Work + Allocation_3$ (5, 3, 2 + 2, 1, 1)

Work =	A	B	C
	7	4	3

Finish =	0	1	2	3	4
	false	true	false	true	false

For i = 4 — Step 2

$Need_4 = 4, 3, 1$

Finish [4] = false and $Need_4$ (4, 3, 1) < Work (7, 4, 3)

So P_4 must be kept in safe sequence

Step 3

$Work = Work + Allocation_4$ (7, 4, 3 + 0, 0, 2)

Work =	A	B	C
	7	4	5

Finish =	0	1	2	3	4
	false	true	false	true	true

For i = 0 — Step 2

$Need_0 = 7, 4, 3$

Finish [0] is false and Need (7, 4, 3) < Work (7, 4, 5)

So P_0 must be kept in safe sequence

Deadlock Avoidance Algorithms

Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,

Process	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	

Process	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

Step 3

Work = Work + Allocation$_0$ (7, 4, 5 + 0, 1, 0)

	A	B	C
Work =	7	5	5

	0	1	2	3	4
Finish =	true	true	false	true	true

Step 2

For i = 2

$Need_2 = 6, 0, 0$

Finish [2] is false and $Need_2 < Work$ (6, 0, 0 < 7, 5, 5)

So P_2 must be kept in safe sequence

Step 3

Work = Work + Allocation$_2$ (7, 5, 5 + 3, 0, 2)

	A	B	C
Work =	10	5	7

	0	1	2	3	4
Finish =	true	true	true	true	true

Step 4

Finish [i] = true for $0 \leq i \leq n$

Hence the system is in Safe state

The safe sequence is P_1, P_3, P_4, P_0, P_2