DEADLOCK

DR.PADMAJA JOSHI

DEADLOCK

- *What is deadlock?*
  - *A set of processes are in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.*
- *A process must request for the resource before using it and must release after using it.*
- *Three operations are associated with resources*
  - *Request*
  - *Use*
  - *Release*

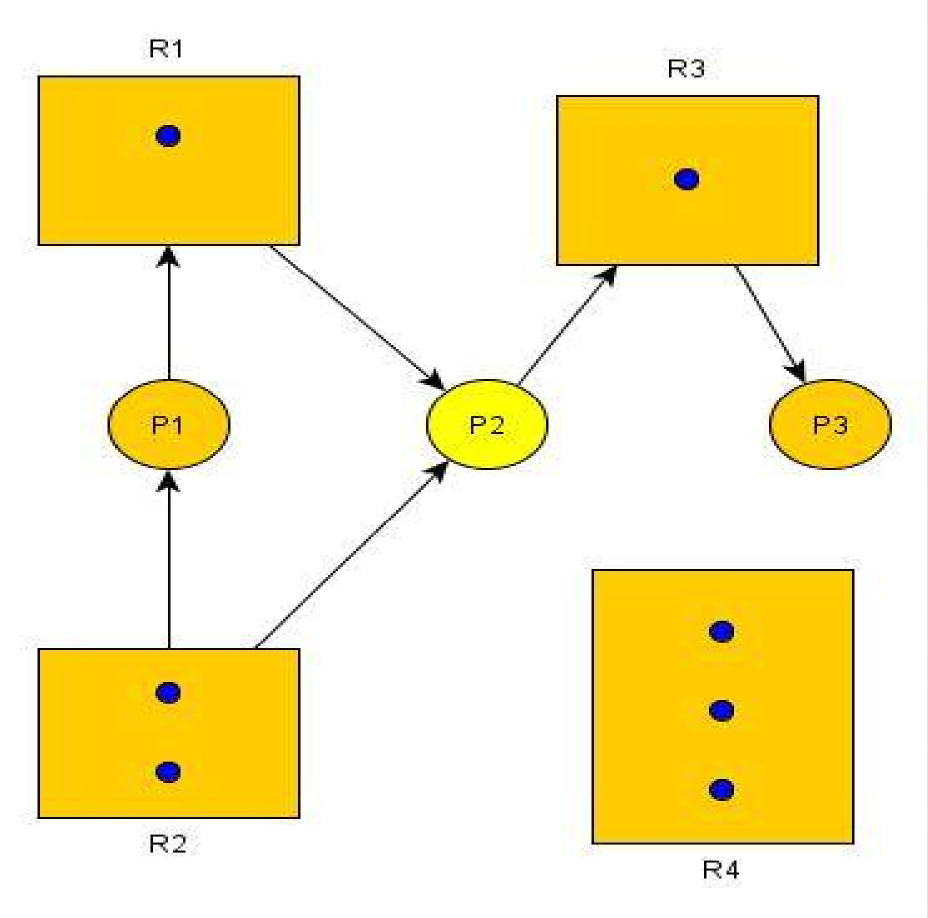
CONT…

- *Multithreaded programs are good candidates for deadlocks;*
- *In deadlock processes never finish executing and prevent other processes from starting leading into starvation of processes that are waiting;*

NECESSARY CONDITIONS FOR DEADLOCK

- *Mutual exclusion – at least one resource must be held in a non-sharable mode*
- *Hold and wait – A process must hold at least one resource and be waiting for additional resource*
- *No preemption – resources cannot be preempted*
- *Circular wait -*

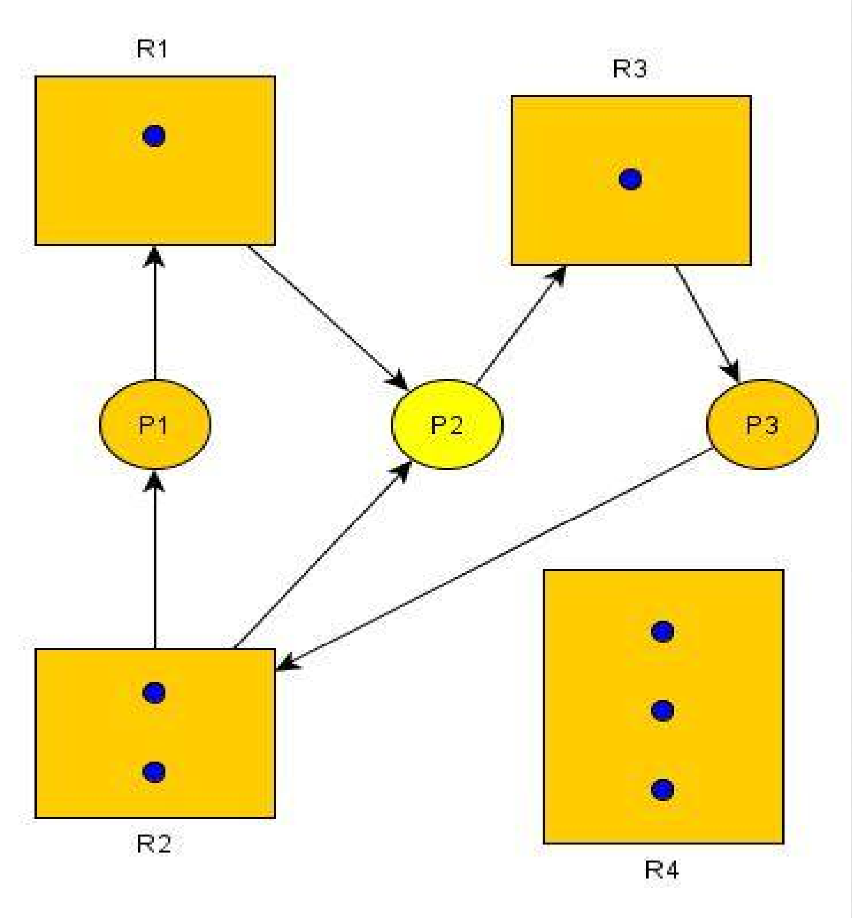
RESOURCE ALLOCATION GRAPH



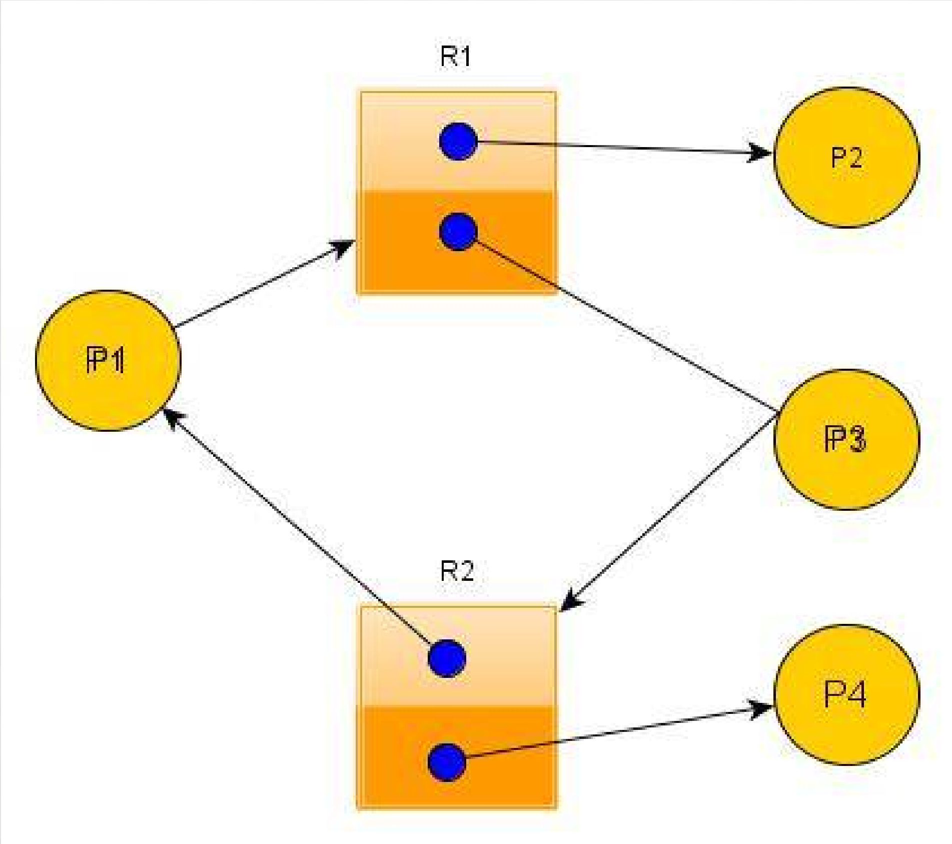
CONT…

- *Set of vertices and edges*
- *Vertices – processes and resources*
- *Edges – assignment and request*
- *Cycles in a graph represent deadlock situation*

RESOURCE ALLOCATION GRAPH WITH DEADLOCK



CYCLE BUT NO DEADLOCK



METHODS FOR HANDLING DEADLOCKS

- *Prevent or avoid deadlock*
- *Detect and recover deadlock*
- *Ignore and pretend no deadlock – unix and windows*

DEADLOCK PREVENTION

- *Ensure that at least one of the necessary conditions is not satisfied*
- *Sharable resources to avoid mutual exclusion– read only files*
- *Avoid hold and wait*
    - *All the resources be allocated before starting the execution*
    - *Request the resources only when it has none*
    - *Disadv – resource utilization low*
        - *Starvation possible*

CONT…

- *Prevent no preemption –*
  - *release the holding resource if process has to wait for another resource*
  - *Before starting the execution check if all the resources are available*
- *Prevent circular wait –*
  - *Each process will request the resources in increasing order only*

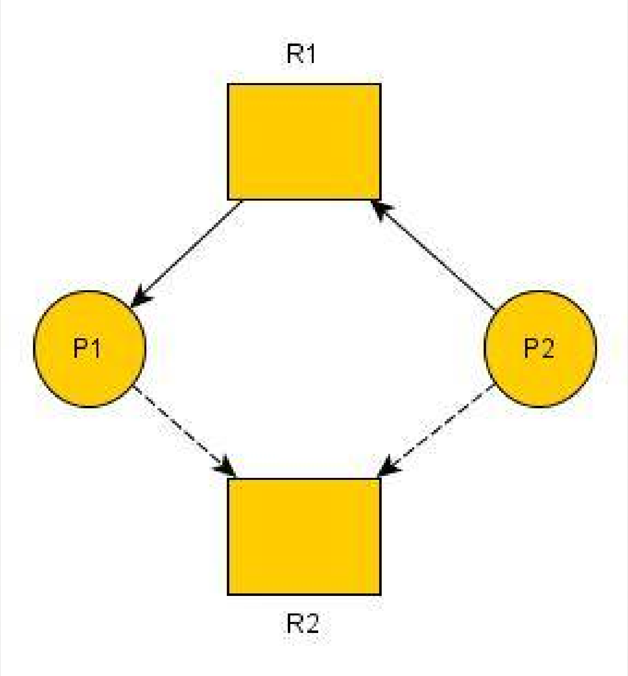
DEADLOCK AVOIDANCE

- *Provide information about the resource requirement in advance*
- *Safe state*
- *Safe sequence*

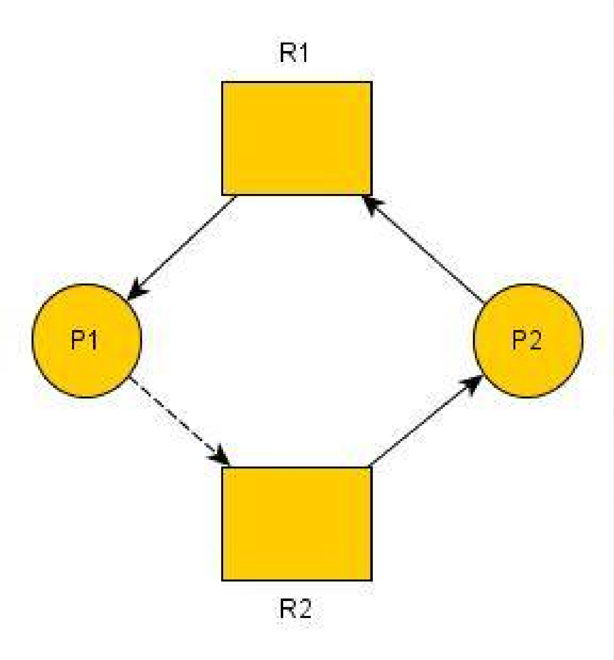
Process	Maximum Need	Currently holding	May need
P0	10	5	5
P1	4	2	2
P2	9	2	7

Number of resources = 12

RESOURCE ALLOCATION GRAPH ALGORITHM



Resource request



Resource allocation unsafe

BANKER'S ALGORITHM

- *m : number of resource types*
- *n : number of processes*
- *Available: vector[m] : resource availability*
- *Max: Max[i][j]: maximum requirement*
- *Allocation: Allocation[i][j]: current allocation*
- *Need: Need[i][j] – further need of resources*

SAFETY ALGORITHM

1. *Let Work[m] and Finish[n]*

   Work = Available

   Finish[i] = false for i=0,1,…n
2. *Find i*
   - *Finish[i] == false*
   - $Need_i \leq Work$ *if no such i then go to 4*
3. *Work = Work+ Allocation*

   Finish[i] = true

   Goto Step2

4. *If Finish[i] = true for all i, system is in safe state*

RESOURCE REQUEST ALGORITHM

$Request_i$ – request vector for process i

1. *If $Request_i \le Need_i$ goto 2 else generate error*
2. *If $Request_i \le Available_i$ goto 3 else wait*
3. *$Available = Available – Request_i$*

   $Allocation_i = Allocation_i + Request_i$

   $Need_i = Need_i - Request_i$

EXAMPLE

Process	Allocation	Max	Available	Need
			3 3 2	
P0	0 1 0	7 5 3	7 5 5	7 4 3
P1	2 0 0	3 2 2	5 3 2	1 2 2
P2	3 0 2	9 0 2	10 5 7	6 0 0
P3	2 1 1	2 2 2	7 4 3	0 1 1
P4	0 0 2	4 3 3	7 4 5	4 3 1

Safe Sequence: P1, P3, P4, P0, P2

DEADLOCK DETECTION

- *Identify first if deadlock has occurred*
- *Algorithm to recover from deadlock*

SINGLE INSTANCE OF EACH RESOURCE

- *cycle*
- *Algorithm to detect the cycle*

SEVERAL INSTANCES OF THE RESOURCES

- *Available: vector[m] : resource availability*
- *Allocation: Allocation[i][j]: current allocation*
- *Request: Request[i][]j]*

ALGORITHM

1. *Let Work[m] and Finish[n]*

   Work = Available

Finish[i] = false for i=0,1,…n-1 if $Allocation_i \neq 0$

2. *Find i*
   - *Finish[i] == false*
   - $Request_i \leq Work$ *if no such i then go to 4*
3. *Work = Work+ Allocation*

   Finish[i] = true

   Goto Step2

4. *If Finish[i] = true for all i, system is in safe state*

EXAMPLE

Process	Allocation	Request	Available	Request-2
P0	0 1 0	0 0 0	0 1 0	0 0 0
P1	2 0 0	2 0 2		2 0 2
P2	3 0 3	0 0 0		0 0 1
P3	2 1 1	1 0 0		1 0 0
P4	0 0 2	0 0 2		0 0 2

[redacted]

DEADLOCK RECOVERY

- *Preempt some or all resources from the deadlocked processes*
  - *Selecting a victim*
  - *Rollback*
  - *Starvation*
- *Terminate the processes involved in the deadlock*