Deadlocks

Mehdi Kargahi
School of ECE
University of Tehran
Summer 2016

What is a Deadlock

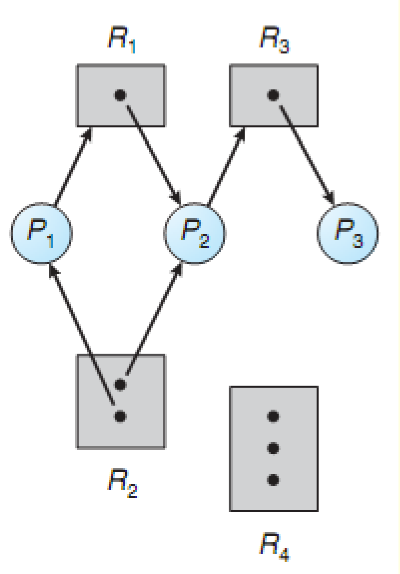
- Processes use resources in the following sequence:
  - Request → Use → Release
- A number of processes may participate in a deadlock
- Example
  - Three processes on three CD/RW drives: *cyclic wait*
  - Two processes requesting printer and DVD drive

Necessary Condition for Deadlock

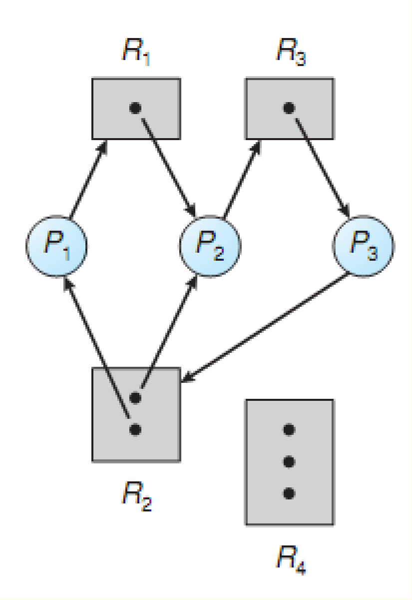
- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait

Resource-Allocation Graph

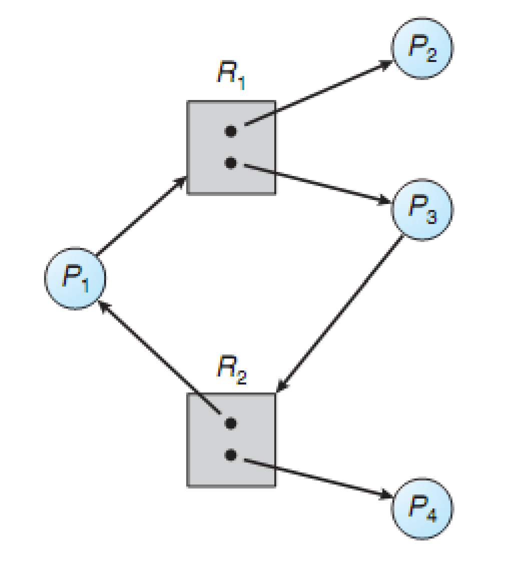
- $P = \{P_1, \ldots, P_n\}$
- $R = \{R_1, \ldots, R_m\}$
- Request edge: $P_i \rightarrow R_j$
- Assignment edge: $R_j \rightarrow P_i$



Resource-Allocation Graph with a Deadlock



Resource-Allocation Graph with a Cycle but no Deadlock



- Deadlock ➔ Cycle
- Cycle ➔ Deadlock is probable

Deadlock Handling

- Deadlock prevention or deadlock avoidance
  - *Prevention*: To ensure deadlock will never occur
  - *Avoidance*: Knowing the resource requirements, deadlock will be avoided on each allocation
- Deadlock detection and recovery
- Ignoring deadlock (assuming it never occurs)

Deadlock Prevention

- At least one of the necessary conditions for deadlock should not be held

1. Mutual exclusion
   - This condition cannot be denied, unless for sharable resources, e.g., a real-only file

Deadlock Prevention

2. Hold and wait
    - *Method*1: Each process request is to be granted (all desired resources be allocated) before execution (no waiting)
    - *Method*2: A process requests resources only when it has none (no holding)

- Disadvantages of
    - *Method*1: low resource utilization
    - *Method*2: starvation

Deadlock Prevention

3. No preemption
   - A protocol
     - A waiting process will implicitly release its current resources
     - If a process requests some resources
       - Resources are free ➔ allocate them
       - Resources are held by a waiting process ➔ preempt the resources and allocate them to this process
       - Otherwise, the requesting process must wait
   - This protocol can be applied only to resources such as CPU registers or memory space whose states can be easily saved and restored later
   - Disadvantages: low resource utilization and starvation

Deadlock Prevention

4. Cyclic wait
   - One way to ensure that this condition may not hold
     - Imposing a total ordering of all resource types
       - R={R_1, …, R_m}, F: $R \rightarrow N$
     - Requiring that each process requests resources in an increasing order of enumeration, or
     - Releasing the high-order resource before requesting a low-order resource
   - *Proof?*
   - This protocol has been implemented in Mainframes

Deadlock Avoidance

- Deadlock is potentially possible, but on every resource request, the safety of the allocation is examined
- More information is required
  - Available resources as well as the allocated ones
  - Maximum demands of the resources by each process
- A state is ***safe*** if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock
- A system is in a safe state only if there exists a ***safe sequence***

Deadlock Avoidance

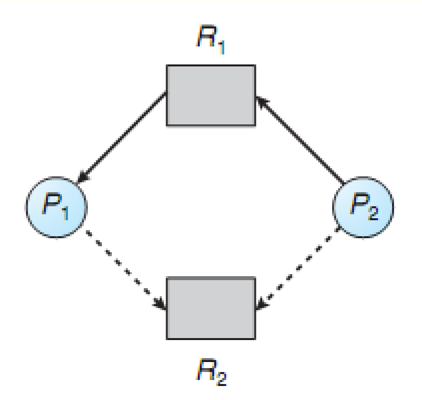
- Example:
  - A system consists of 12 magnetic tapes and three processes

	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2

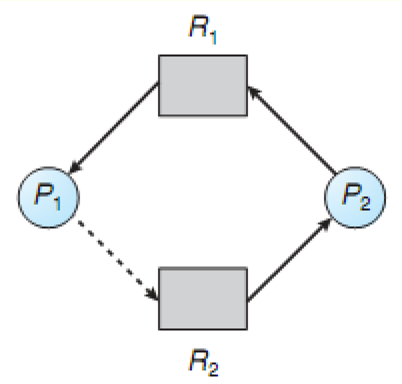
  - Is the requests safe?
  - Safe sequence: <p_1, p_0, p_2>
  - What if P_2 request a tape?

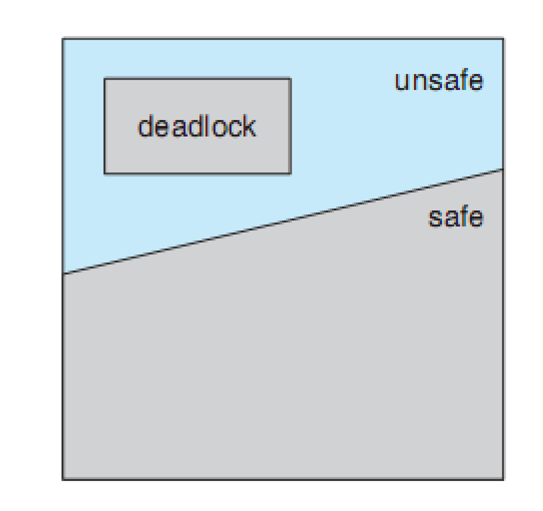
Resource-Allocation-Graph Algorithm Only applicable to one-instance resources

- Claim edge



- An unsafe state





Banker's Algorithm
Applicable to multiple-instance resources

- *n* processes
- *m* resources
- $Available_{1\times m}$: $Available\,[j] = k$
- $Max_{n\times m}$: $Max\,[i][j] = k$
- $Allocation_{n\times m}$: $Allocation\,[i][j] = k$
- $Need_{n\times m}$: $Need\,[i][j] = k$
  - $Need\,[i][j] = Max\,[i][j] - Allocation\,[i][j]$
- These data structures vary over time in both size and value
- $Allocation_i$, $Need_i$, and Max_i show the row related to P_i
- $X \le Y$ *iff* $X[i] \le Y[i]$ for all $i=1, 2, \ldots$
- $X < Y$ *iff* $X \le Y$ *and* $X \ne Y$

Banker's Algorithm
Safety Algorithm

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

2. Find an i such that both

   a. $Finish[i] == false$

   b. $Need_i \le Work$

   If no such i exists, go to step 4.

3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   Go to step 2.

4. If $Finish[i] == true$ for all i, then the system is in a safe state.

Banker's Algorithm Resource-Request Algorithm

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:

$$
\begin{aligned}
&Available = Available - Request_i;\\
&Allocation_i = Allocation_i + Request_i;\\
&Need_i = Need_i - Request_i;
\end{aligned}
$$

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

	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	

➔

	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

➔ $< P_1, P_3, P_4, P_2, P_0 >$

- The state is safe
- What if $Request_1$=(1, 0, 2)?

	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 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

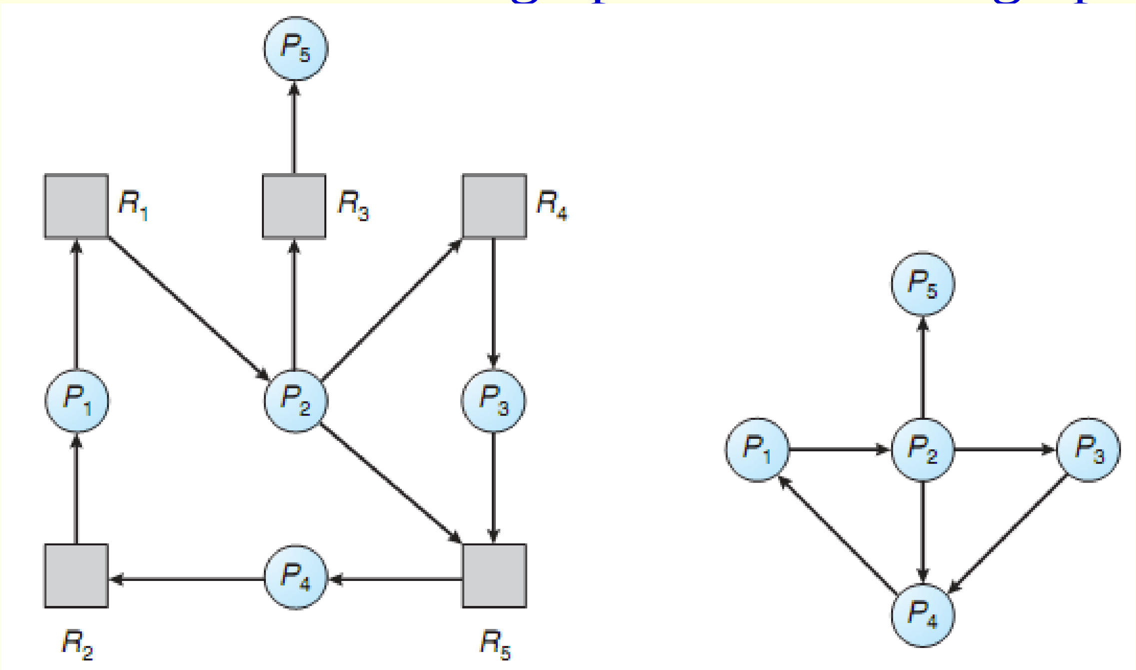
➔ $< P_1, P_3, P_4, P_0, P_2 >$

- $Request_4$=(3, 3, 0)?
- $Request_0$=(0, 2, 0)?

Deadlock Detection

- Single instance of each resource type

Resource allocation graph   Wait-for graph



- Deadlock detection requires O(n^2) operations

Deadlock Detection

- Several instances of a resource type

1. Let *Work* and *Finish* be vectors of length m and n, respectively. Initialize *Work* = *Available*. For $i = 0, 1, ..., n-1$, if $Allocation_i \neq 0$, then $Finish[i] = false$. Otherwise, $Finish[i] = true$.
2. Find an index i such that both
   a. $Finish[i] == false$
   b. $Request_i \leq Work$

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

- Deadlock detection requires $m \times n^2$ operations

Example

	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	

- This state is safe ➔ Safe sequence: $<P_0, P_2, P_3, P_1, P_4>$
- P_2 makes one additional request of C

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

- Which processes are in deadlock?

How often the detection algorithm should be invoked?

- On each allocation
  - High overhead
  - The process creating the deadline can be detected
- Low frequency of the detection algorithm
  - Low overhead
  - More than one cycles may be detected
- Higher frequency but when the system isn't overloaded!!
- Even if the system is overloaded, large number of cycles due to less frequent detection results in performance loss

Deadlock Recovery

- Process termination
  - Abort all deadlocked processes
  - Abort one process at a time until the deadlock cycle is eliminated
    - Cost factors to select the next victim

1. What the priority of the process is
2. How long the process has computed and how much longer the process will compute before completing its designated task
3. How many and what type of resources the process has used (for example, whether the resources are simple to preempt)
4. How many more resources the process needs in order to complete
5. How many processes will need to be terminated
6. Whether the process is interactive or batch

Deadlock Recovery

- Resource preemption
  - Selecting a victim
    - Which resources and which processes are to be preempted? (Minimizing the cost)
  - Rollback
    - Total rollback vs. checkpointing
  - Starvation
    - If the same process is always selected as the victim!!
    - Cost factor: number of roll-backs

Priority Inversion

- PIP
- Deadlock
- Deadline