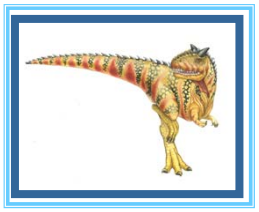
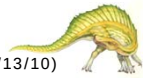
Chapter 7: Deadlocks



(slides selected/modified by R. Doemer, 01/12/11)

Chapter 7: Deadlocks

- The Deadlock Problem
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

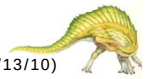


(slide modified by R. Doemer, 05/13/10)

The Deadlock Problem

- A set of **blocked processes**, each holding a resource and waiting to acquire a resource held by another process in the set
- Application Example
  - System has 2 disk drives
  - P_1 and P_2 each hold one disk drive and each needs another one
- Example with semaphores
  - Binary semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	wait(B)
wait (B);	wait(A)

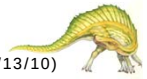


(slide modified by R. Doemer, 05/13/10)

Dead Locks, System Model

- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**



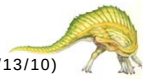
(slide modified by R. Doemer, 05/13/10)

Deadlock Characterization

Deadlock can arise *if and only if* four conditions hold simultaneously:

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

Note that these four are *necessary conditions*!



(slide modified by R. Doemer, 05/13/10)

Resource-Allocation Graph

Resource-Allocation Graph:
A set of vertices V and a set of edges E.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- E is partitioned into two types:
  - request edge – directed edge $P_i \rightarrow R_j$
  - assignment edge – directed edge $R_j \rightarrow P_i$

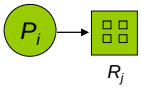
(slide modified by R. Doemer, 05/13/10)

Resource-Allocation Graph

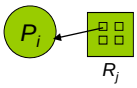
- Process

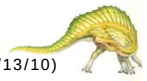
- Resource type with 4 instances

- P_i requests instance of R_j



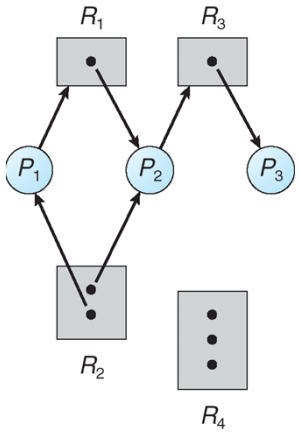
- P_i is holding an instance of R_j

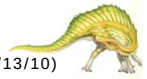




(slide modified by R. Doemer, 05/13/10)

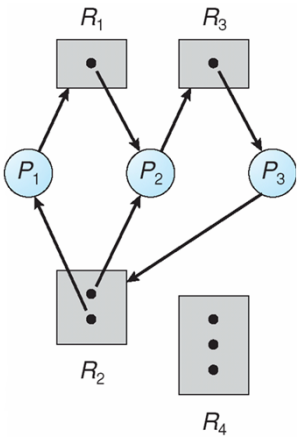
Example of a Resource-Allocation Graph



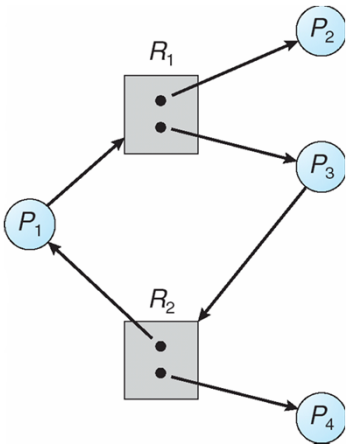


(slide modified by R. Doemer, 05/13/10)

Resource-Allocation Graph With A Deadlock

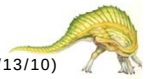


Graph With A Cycle But No Deadlock



Basic Facts

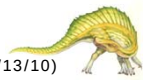
- If resource-allocation graph contains no cycles
  ⇒ no deadlock!
- If resource-allocation graph contains a cycle
  ⇒
  - if only one instance exists per resource type, then it is a deadlock
  - if several instances exist per resource type, then there is a *possibility* of a deadlock



(slide modified by R. Doemer, 05/13/10)

Methods for Dealing with Deadlocks

- Ensure that the system will *never* enter a deadlock state
  - Deadlock prevention
  - Deadlock avoidance
- Allow the system to enter a deadlock state and then recover
  - Recovery from deadlock
- Ignore the problem and pretend that deadlocks never occur in the system
  - used by most operating systems, including UNIX and Windows



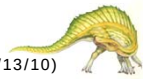
(slide modified by R. Doemer, 05/13/10)

Deadlock Prevention

To **prevent** deadlocks from occurring,
we can restrain the ways request can be made.

Prevent one of the four necessary conditions!

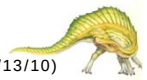
- **Mutual Exclusion** – not required for sharable resources; must hold for non-sharable resources
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none
  - Low resource utilization; starvation possible



(slide modified by R. Doemer, 05/13/10)

Deadlock Prevention (Cont.)

- **No Preemption** –
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration
  - This is the most realistic way of deadlock prevention!



(slide modified by R. Doemer, 05/13/10)

End of Chapter 7