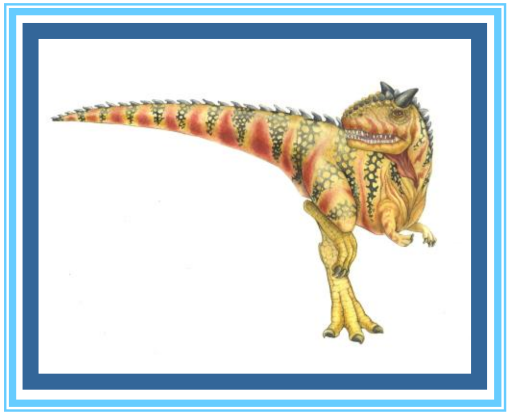
Chapter 7: Deadlocks



Chapter 7: Deadlocks

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

Chapter Objectives

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks.
- To present a number of different methods for preventing or avoiding deadlocks in a computer system.

System Model

- System consists of resources.
- 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**

Deadlock

- A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.
- Here is the simplest example:

```
Program 1 requests resource A and receives it.
Program 2 requests resource B and receives it.
Program 1 requests resource B and is queued up, pending the release of B.
Program 2 requests resource A and is queued up, pending the release of A.
```

Deadlock Characterization

Deadlock can arise 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.

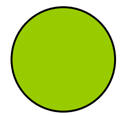
Resource-Allocation Graph

A set of vertices V and a set of edges E.

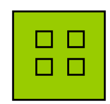
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** –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

Resource-Allocation Graph (Cont.)

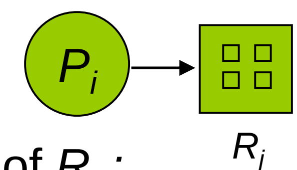
- Process:



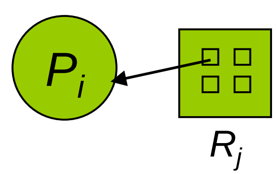
- Resource Type with 4 instances:



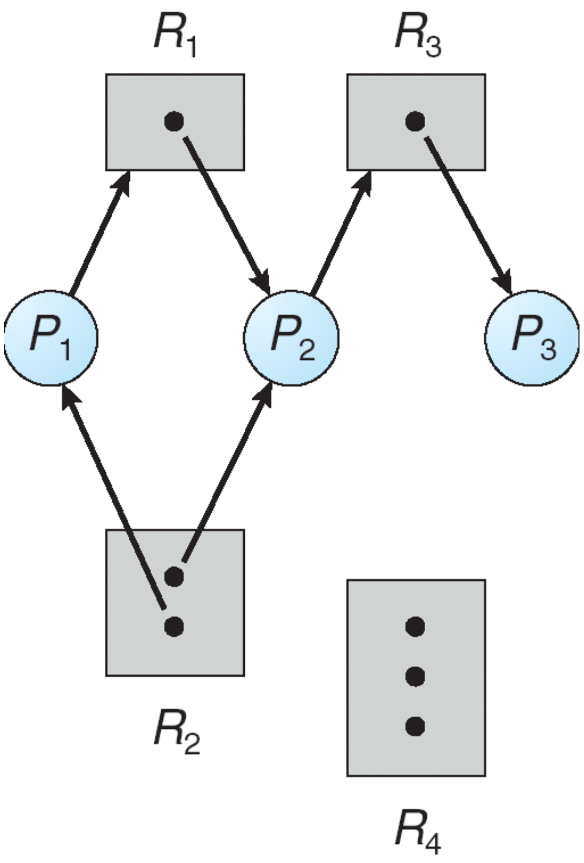
- P_i requests instance of R_j :



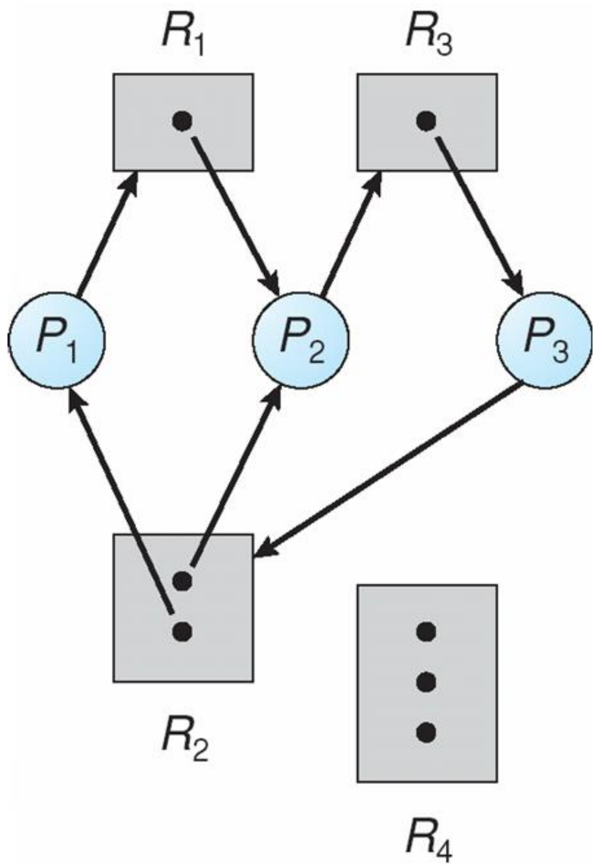
- P_i is holding an instance of R_j :



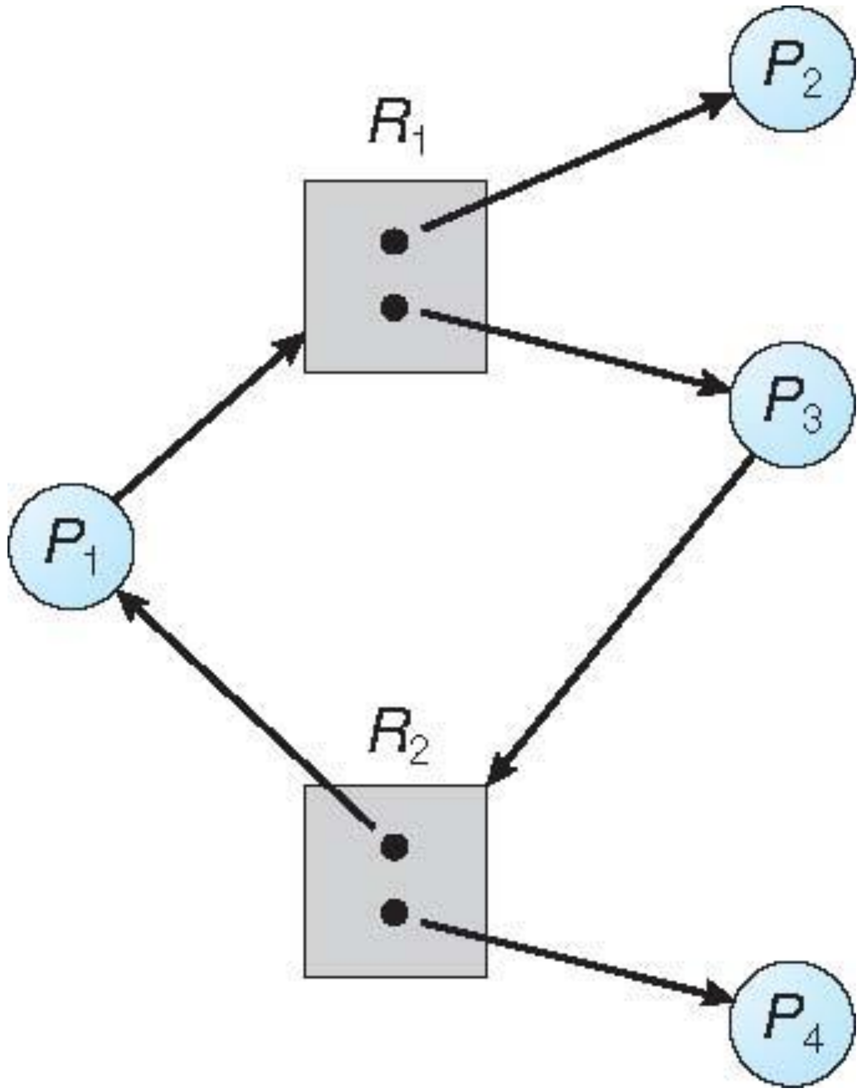
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles $\Rightarrow$ no deadlock.
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

Methods for Handling Deadlocks

- Ensure that the system will ***never*** enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by some operating systems, including UNIX.

Deadlock Prevention

Restrain the ways request can be made:

- **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.

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