DEADLOCKS

DEADLOCKS

- Two processes each want to record a scanned document on a CD.
- Process *A requests permission to use the scanner and is granted it.*
- *Process B* is programmed in such a way that it requests the CD recorder first and is also granted it.
- Now *A asks for the CD recorder, but the request is denied until B releases it.*
- Instead of releasing the CD recorder *B asks for the scanner*.
- *At this* point both processes are blocked and will remain so forever. This situation is called a **deadlock.**

DEADLOCKS

- Deadlocks can occur on hardware resources or on software resources.
- Eg:In a database system, a program may have to lock several records it is using, to avoid race conditions. If process *A locks record R1 and process B locks record R2, and then each process tries to lock the other one's record, then there is a* a deadlock.

DEADLOCKS

RESOURCES

- A resource is anything that can be used by only a single process at any instant of time.
- A resource can be a hardware device (e.g., a tape drive) or a piece of information (e.g., a locked record in a database).
- Some resources, several identical instances may be available, such as three tape drives.
- When several copies of a resource are available, any one of them can be used to satisfy any request for the resource.

DEADLOCKS

Preemptable and Nonpreemptable Resources

- A preemptable resource is one that can be taken away from the process owning it with no ill effects.
- Eg: Memory
- A non preemptable resource, is one that cannot be taken away from its current owner without causing the computation to fail.
- Eg: CD Recorder
- If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a garbled CD, CD recorders are not preemptable at an arbitrary moment.

DEADLOCKS

- Deadlocks involve nonpreemptable resources.
- Potential deadlocks that involve preemptable resources can usually be resolved by reallocating resources from one process to another.
- The sequence of events required to use a resource
  1. Request the resource.
  2. Use the resource.
  3. Release the resource.

DEADLOCKS

DEADLOCKS

- *A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.*
- A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a **deadlock.**

DEADLOCKS

Conditions for Deadlock

- A deadlock situation can arise if the following four conditions hold simultaneously in a system

1. Mutual exclusion condition.

2. Hold and wait condition.

3.No preemption condition..

4.Circular wait condition.

DEADLOCKS

1. **Mutual exclusion:** At least one resource must be held in a nonsharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. **Hold and wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

DEADLOCKS

3. **No preemption:** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. **Circular wait.:** A set *{P0, P1, ..., Pn} of waiting processes must exist such* that *P0 is waiting for a resource held by P1, P1 is waiting for a resource* held by *P2, ..., Pn−1 is waiting for a resource held by Pn, and Pn is waiting* for a resource held by *P0.*

- **All four of these conditions must be present for a deadlock to occur. If one of them is absent, no deadlock is possible.**

Resource-Allocation Graph

- Deadlocks can be described precisely in terms of a directed graph called a **system resource-allocation graph.**

- This graph consists of a set of vertices *V* and a set of edges *E*.
- *The set of vertices V is partitioned into two different types* of nodes:
  - *P = {P1, P2, ..., Pn}, the set consisting of all the active processes in the* system.
  - *R = {R1, R2, ..., Rm}, the set consisting of all resource types in the* system.

Resource-Allocation Graph

Request edge

- *A directed edge Pi → Rj is called a* **request edge.**
- It signifies that process *Pi has requested an instance of resource type Rj and* is currently waiting for that resource.

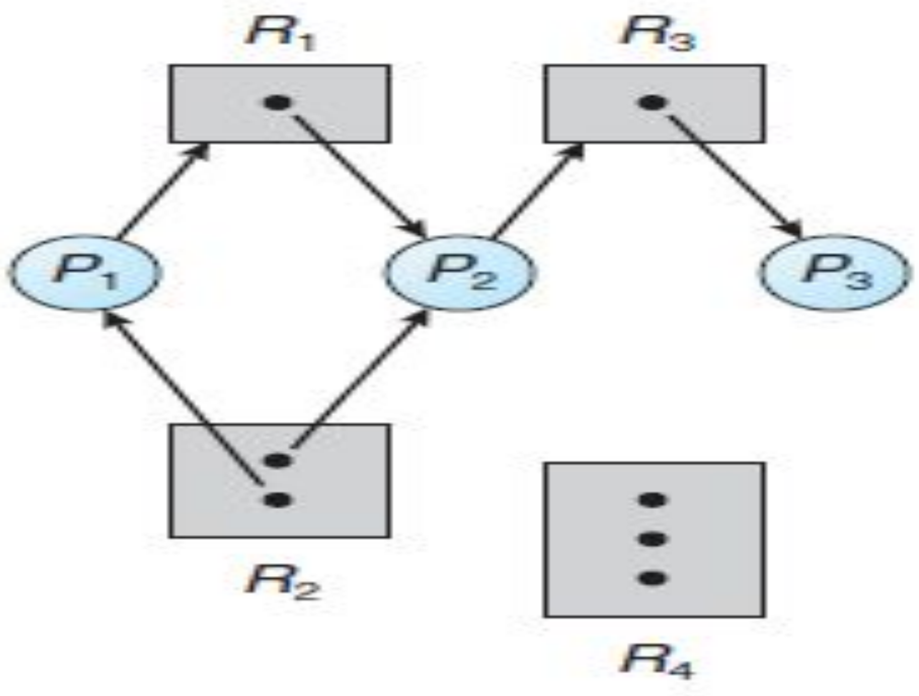
Assignment edge

- A directed edge *Rj → Pi is called an* ***assignment edge*** .
- *It signifies that an instance of resource* type *Rj has been allocated to process Pi.*

Resource-Allocation Graph

- Pictorially, we can represent each process P_i *as a circle and each resource type* R_j *as a rectangle.*
- *Since resource type* R_j *may have more than one instance, we* represent each instance as a dot within the rectangle.
- When process P_i *requests an instance of resource type* R_j*, a request edge* is inserted in the resource-allocation graph.
- When this request can be fulfilled, the request edge is *transformed to an assignment edge.*
- *When* the process no longer needs access to the resource, it releases the resource. As a result, the assignment edge is deleted.

Resource-Allocation Graph-Example



Resource-allocation graph.

Resource-Allocation Graph-Example

- **The sets *P, R, and E:***

  $P = \{P1, P2, P3\}$

  $R = \{R1, R2, R3, R4\}$

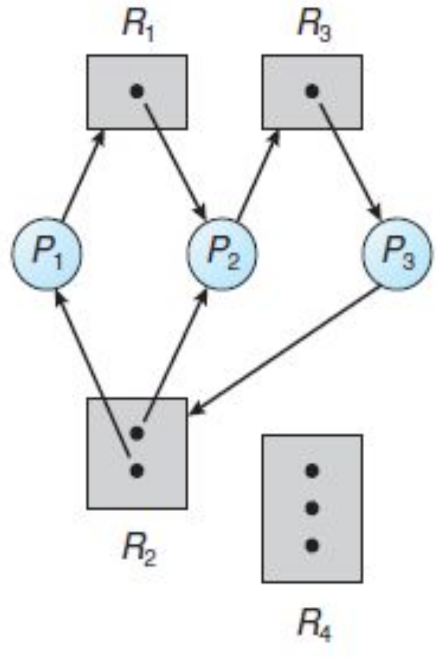
  $E = \{P1 \rightarrow R1, P2 \rightarrow R3, R1 \rightarrow P2, R2 \rightarrow P2, R2 \rightarrow P1, R3 \rightarrow P3\}$
- **Resource instances:**
  - One instance of resource type *R1*
  - Two instances of resource type *R2*
  - One instance of resource type *R3*
  - Three instances of resource type *R4*
- **Process states:**
  - Process *P1 is holding an instance of resource type R2 and is waiting for* an instance of resource type *R1*.
  - Process *P2 is holding an instance of R1 and an instance of R2 and is* waiting for an instance of *R3*.
  - Process *P3 is holding an instance of R3*.

Resource-Allocation Graph

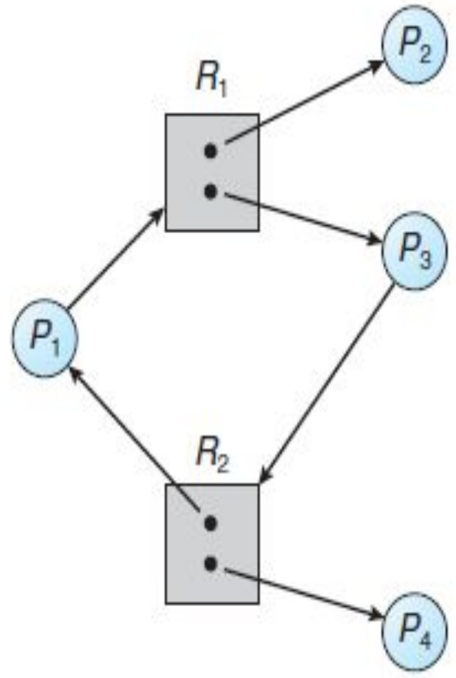
- If the graph contains no cycles, then no process in the system is deadlocked.
- If the graph does contain a cycle, then a deadlock may exist.
- If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred.
- If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred. Each process involved in the cycle is deadlocked. In this case, a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock.
- If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.

Resource-Allocation Graph

- If a resource-allocation graph does not have a cycle, then the system is ***not in a deadlocked state. If there is a cycle, then the system may or*** may not be in a deadlocked state.



Resource-allocation graph with a deadlock.



Resource-allocation graph with a cycle but no deadlock.