OPERATING SYSTEMS

Module3_Part7

Textbook : Operating Systems Concepts by Silberschatz

Resource-Allocation Graph

- Deadlocks can be described more 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

- A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_{j;}$

  it signifies that process P_i has requested an instance of resource type Rj and

  is currently waiting for that resource.

A directed edge from resource type Rj to process P_i is denoted by $Rj \rightarrow P_i$; it signifies that

an instance of resource type Rj has been allocated to process P_i ;

A directed edge P_i -$\rightarrow R_j$ is called a request edge;

a directed edge $Rj \rightarrow P_i$; is called an assignment edge

Resource-Allocation Graph

- Pictorially we represent each process P_i as a circle and each resource type *Rj* as a rectangle.
- Since resource type *Rj* may have more than one instance, we represent each such instance as a dot within the rectangle.
- Note that a request edge points to only the rectangle *Rj,* whereas an assignment edge must also designate one of the dots in the rectangle.
- When process P_i requests an instance of resource type *Rj,* a request edge is inserted in the resource-allocation graph. When this request can be fulfilled, the request edge is *instantaneously* 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

Consider the current situation in the system

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

P == {P1, P2, P3}

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

E == {Pl✉ Rl, p2✉ *R3*, Rl✉ *p2*, R2✉ *p2*,R2✉ Pl, R3✉ P3}

Resource instances:

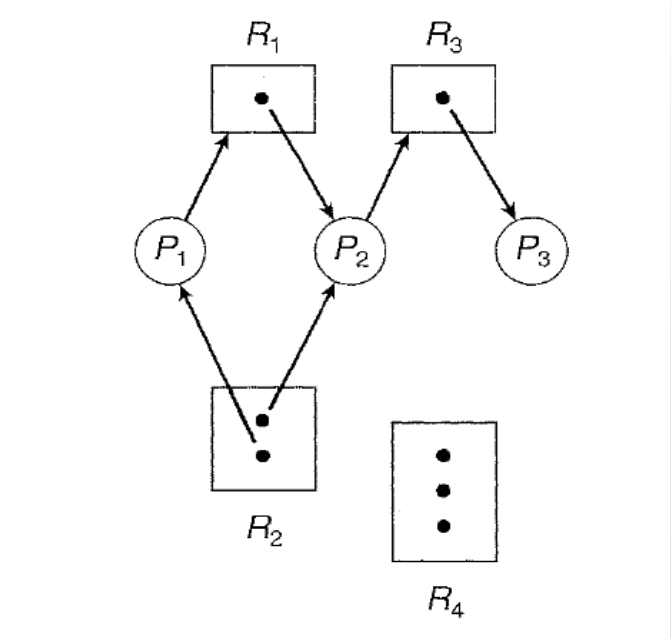
One instance of resource type R1

Two instances of resource type R2

One instance of resource type R3

Three instances of resource type R4

Resource allocation graph



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

Given the definition of a resource-allocation graph, it can be shown that, 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.

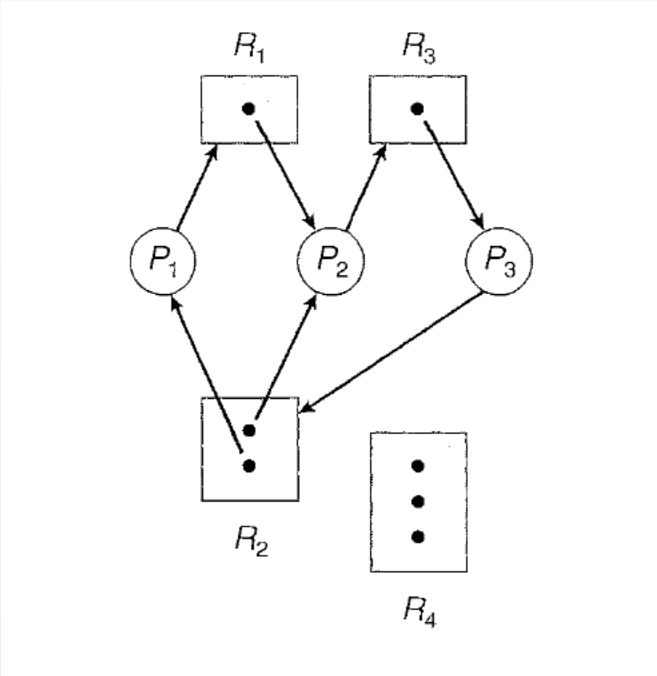
Resource allocation graph

- 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.

To illustrate this concept, we return to the resource-allocation graph depicted in Figure. Suppose that process *P3* requests an instance of resource type R2. Since no resource instance is currently available, a request edge *P3* ✉ R2 is added to the previous graph. At this point, two minimal cycles exist in the system:

Resource allocation graph

- P1 ✉ *R* 1 ✉ P2 ✉ R3 ✉ *P3* ✉ R2 ✉ P1
- P2 ✉ R3 ✉ *P3* ✉ R2 ✉ P2



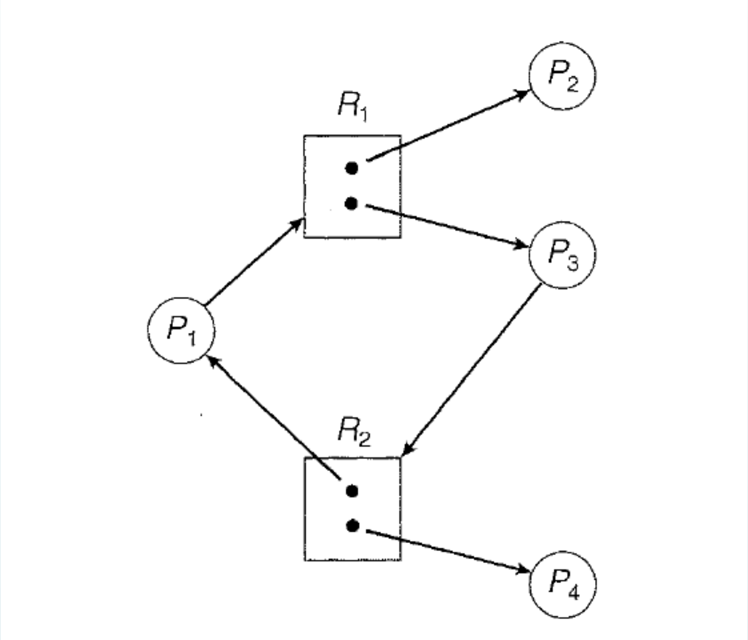
Resource-allocation graph with a deadlock

Resource allocation graph

- Processes *P1*, *P2*, and *P3* are deadlocked. Process *P2* is waiting for the resource
- *R3*, which is held by process *P3*. Process *P3* is waiting for either process *P1* or
- process *P2* to release resource R2. In addition, process *P1* is waiting for process
- *P2* to release resource R1.

Resource allocation graph

- Now consider the resource-allocation graph below. in this example,
- we also have a cycle:
- P1 → R1 → P3 → R2 → P1



Resource allocation graph with cycle and no deadlock

However, there is no deadlock. Observe that process *P4* may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle. In summary, 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. This observation is important when we deal with the deadlock problem.