In a multiprogramming environment, several processes may compete for a finite number of resources. 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.

**System Model**

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources may be partitioned into several types (or classes), each consisting of some number of identical instances. CPU cycles, files, and I/O devices (such as printers and DVD drives) are examples of resource types. If a system has two CPUs, then the resource type CPU has two instances. Similarly, the resource type printer may have five instances.

If a process requests an instance of a resource type, the allocation of any instance of the type should satisfy the request. If it does not, then the instances are not identical, and the resource type classes have not been defined properly.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

**1. Request:** The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.

2. **Use:** The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).

3**. Release:** The process releases the resource.

**Necessary Conditions**

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

**1. Mutual exclusion:** At least one resource must be held in a non-sharable 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.

**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.

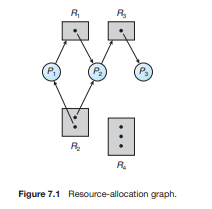
**4. Circular wait:** A set { P0 , P1 , ..., Pn } of waiting processes must exist such that P0 is waiting for a resource held by P1 , P1 is waiting for a resource held by P2 , ..., Pn −1 is waiting for a resource held by Pn ,and Pn is waiting for a resource held by P0.

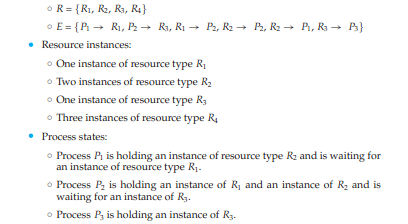
We emphasize that all four conditions must hold for a deadlock to occur. The circular-wait condition implies the hold-and-wait condition, so the four conditions are not completely independent.

**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, and R = { R1 , R2 , ..., Rm }, the set consisting of all resource types in the system.

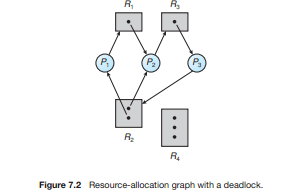
A directed edge from process Pi to resource type Rj is denoted by Pi →Rj ; it signifies that process Pi has requested an instance of resource type Rj and is currently waiting for that resource. A directed edge from resource type Rj to process Pi is denoted by Rj →Pi; it signifies that an instance of resource type Rj has been allocated to process Pi. A directed edge Pi →Rj is called a request edge; a directed edge Rj →Pi is called an assignment edge.





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.

Suppose that process 3 requests an instance of resource type R2.



Since no resource instance is currently available, we add a request edge P3 →R2 to the graph (Figure 7.2). At this point, two minimal cycles exist in the system:

P1 →R1 →P2 →R3 →P3 →R2 →P1

P2 →R3 →P3 →R2 →P2

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.

Video Links: <https://www.youtube.com/watch?v=UVo9mGARkhQ>

REFERENCES:

1. Galvin, Peter B., Silberchatz, A., “*Operating System Concepts*”, Addison Wesley, 8th Edition.
2. Flynn, *“Operating Systems”*, Cengage Learning.
3. Dhamdhere, D.M., "*Operating System:A Concept Based Approach*",  
   Tata Mc-Graw- Hill.
4. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-828-operating-system-engineering-fall-2012/lecture-notes-and-readings/
5. **https://www.youtube.com/watch?v=8ZS7xzWPo5A&list=PL3-wYxbt4yCjpcfUDz-TgD\_ainZ2K3MUZ&index=33**
6. <https://computing.llnl.gov/tutorials/>
7. https://nptel.ac.in/courses/106/105/106105214
8. <https://www.guru99.com/operating-system-tutorial.html>
9. <https://www.geeksforgeeks.org/operating-systems/>