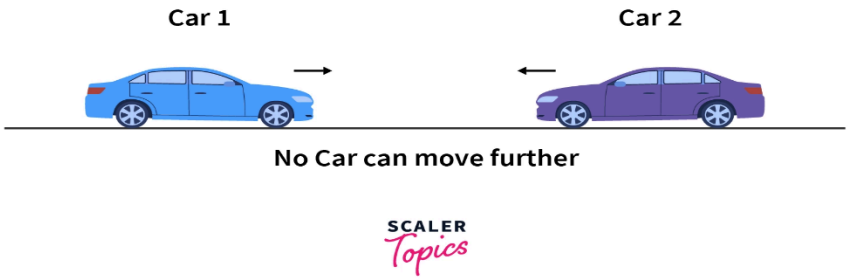
Introduction of Deadlock in Operating System

A process in operating system uses resources in the following way.

1. Requests a resource
2. Use the resource
3. Releases the resource

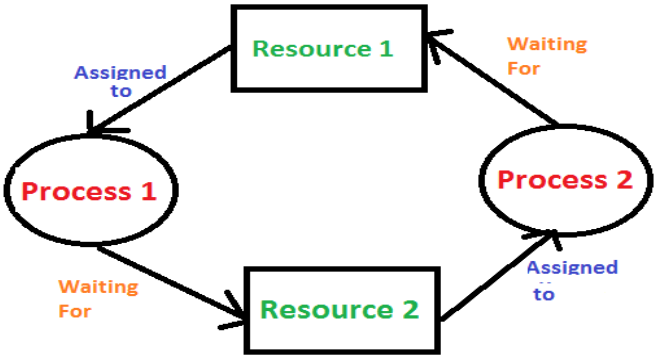
*A **deadlock*** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider a one-way road with two cars approaching from opposite directions, blocking each other. The road is the resource, and crossing it represents a process. Since it's a one-way road, both cars can't move simultaneously, leading to a deadlock.



A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s).

For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Examples Of Deadlock

1. The system has 2 tape drives. P0 and P1 each hold one tape drive and each needs another one.
2. Semaphores A and B, initialized to 1, P0, and P1 are in deadlock as follows:

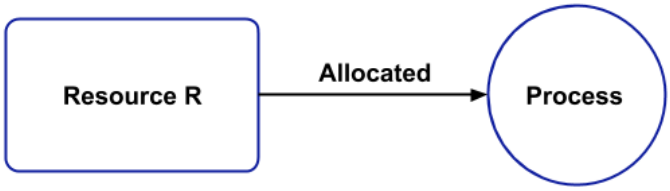
- P0 executes wait(A) and preempts.
- P1 executes wait(B).
- Now P0 and P1 enter in deadlock.

- Deadlock occurs if both processes progress to their second request.
- **Condition to leading to deadlock**
- **Deadlock can arise if** the **following four conditions hold simultaneously (Necessary Conditions)**
- ***Mutual Exclusion:*** Two or more resources are non-shareable (Only one process can use at a time)
  Hold and Wait: A process is holding at least one resource and waiting for resources.
  No Preemption: A resource cannot be taken from a process unless the process releases the resource.
  Circular Wait: A set of processes waiting for each other in circular form.

Necessary Conditions of Deadlock

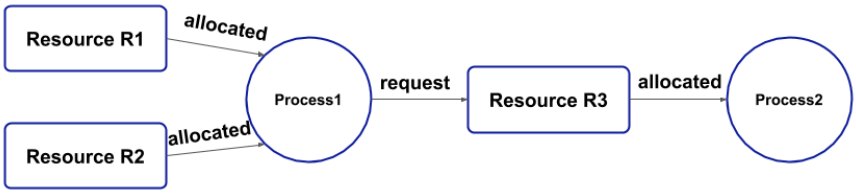
There are four different conditions that result in Deadlock. These four conditions are also known as Coffman conditions and these conditions are not mutually exclusive. Let's look at them one by one.

- **Mutual Exclusion:** A resource can be held by only one process at a time. In other words, if a process P1 is using some resource R at a particular instant of time, then some other process P2 can't hold or use the same resource R at that particular instant of time. The process P2 can make a request for that resource R but it can't use that resource simultaneously with process P1.



AfterAcademy

- **Hold and Wait:** A process can hold a number of resources at a time and at the same time, it can request for other resources that are being held by some other process. For example, a process P1 can hold two resources R1 and R2 and at the same time, it can request some resource R3 that is currently held by process P2.

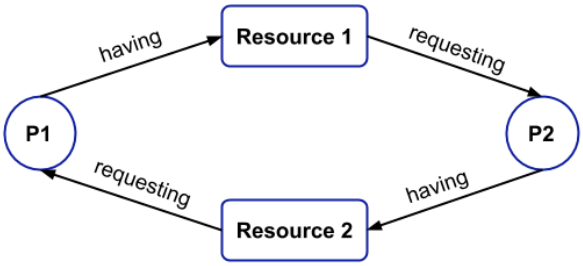


AfterAcademy

- **No preemption:** A resource can't be preempted from the process by another process, forcefully. For example, if a process P1 is using some resource R, then some other process P2 can't forcefully take that resource. If it is so, then what's the need for various scheduling algorithm. The process P2 can request for the

resource R and can wait for that resource to be freed by the process P1.

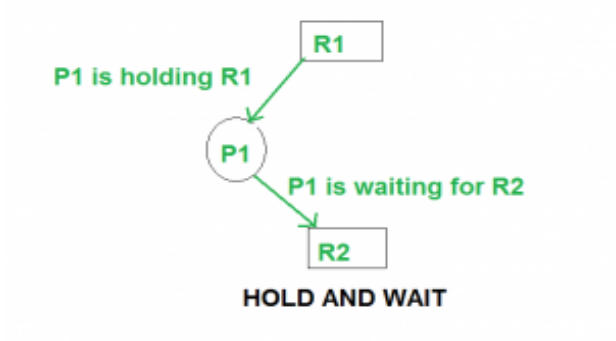
- **Circular Wait:** Circular wait is a condition when the first process is waiting for the resource held by the second process, the second process is waiting for the resource held by the third process, and so on. At last, the last process is waiting for the resource held by the first process. So, every process is waiting for each other to release the resource and no one is releasing their own resource. Everyone is waiting here for getting the resource. This is called a circular wait.



AfterAcademy

Deadlock will happen if all the above four conditions happen simultaneously.

Eliminate Hold and wait: Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires a printer at a later time and we have allocated a printer before the start of its execution printer will remain blocked till it has completed its execution. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.



Eliminate No Preemption : Preempt resources from the process when resources are required by other high-priority processes.
Eliminate Circular Wait : Each resource will be assigned a numerical number. A process can request the resources to increase/decrease. order of numbering. For Example, if the P1 process is allocated R5 resources, now next time if P1 asks for R4, R3 lesser than R5 such a request will not be granted, only a request for resources more than R5 will be granted.
Detection and Recovery: Another approach to dealing with deadlocks is to detect and recover from them when they occur. This can involve killing one or more of the processes involved in the deadlock or releasing some of the resources they hold.

Deadlock Avoidance

A deadlock avoidance policy grants a resource request only if it can establish that granting the request cannot lead to a deadlock either immediately or in the future. The kernal lacks detailed knowledge about future behavior of processes, so it cannot accurately predict deadlocks.
To facilitate deadlock avoidance under these conditions, it uses the following conservative approach:
Each process declares the maximum number of resource units of each class that it may require.
The kernal permits a process to request these resource units in stages- i.e. a few resource units at a time- subject to the maximum number declared by it and uses a worst case analysis technique to check for the possibility of future deadlocks. A request is granted only if there is no possibility of deadlocks; otherwise, it remains pending until it can be granted.
This approach is conservative because a process may complete its operation without requiring the maximum number of units declared by it.

There are three ways to handle deadlock:

1. **Deadlock prevention:** The possibility of deadlock is excluded before making requests, by eliminating one of the necessary conditions for deadlock.

   Example: Only allowing traffic from one direction, will exclude the possibility of blocking the road.

2. Deadlock avoidance: The Operating system runs an algorithm on requests to check for a safe state. Any request that may result in a deadlock is not granted.

   Example: Checking each car and not allowing any car that can block the road. If there is already traffic on the road, then a car coming from the opposite direction can cause blockage.

3. Deadlock detection & recovery: OS detects deadlock by regularly checking the system state, and recovers to a safe state using recovery techniques. **Example:** Unblocking the road by backing cars from one side. Deadlock prevention and deadlock avoidance are carried out before deadlock occurs.