What is Deadlock in Operating System (OS)?

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

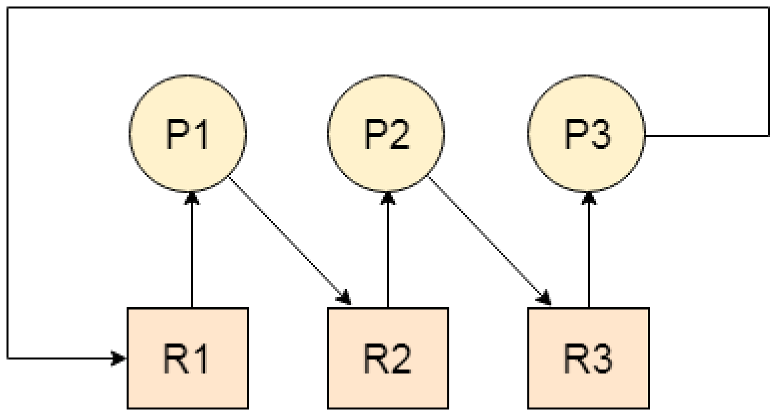
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



Difference between Starvation and Deadlock

Sr.	Deadlock	Starvation
1	Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
2	Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
3	Every Deadlock is always a starvation.	Every starvation need not be deadlock.
4	The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
5	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

Necessary conditions for Deadlocks

1. **Mutual Exclusion**

   A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

2. **Hold and Wait**

   A process waits for some resources while holding another resource at the same time.

3. **No preemption**

   The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

4. **Circular Wait**

   All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

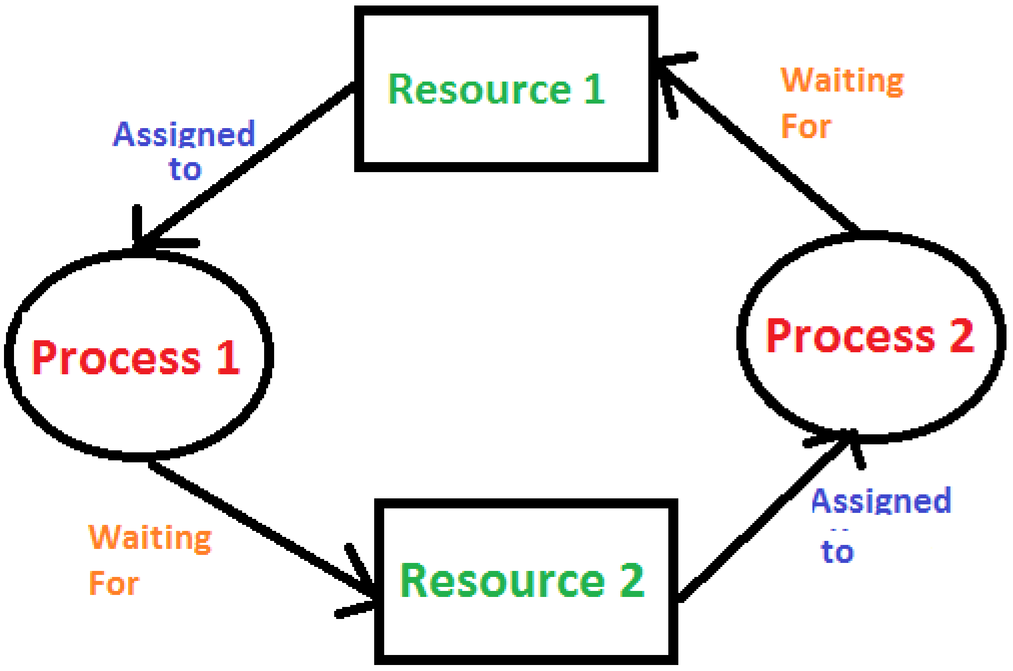
Deadlock(Repeat)

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 an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. 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. Resources A and B, initialized to 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.

P0	**P1**
wait(A)	wait(B)
wait(B)	wait(A)

3. Assume the space is available for allocation of 200K bytes, and the following sequence of events occurs.

P0	P1
Request 80KB	Request 70KB
Request 60KB	Request 80KB

Deadlock occurs if both processes progress to their second request.

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.

Deadlock Handling

There are four methods of handling deadlocks - **deadlock prevention**, **deadlock avoidance**, **deadline detection and recovery** and **deadlock ignorance**.

Methods of handling deadlocks

1. Deadlock Prevention
2. Deadlock avoidance (Banker's Algorithm)
3. Deadlock detection & recovery
4. Deadlock Ignorance (Ostrich Method)

1. Deadlock Prevention

The strategy of deadlock prevention is to design the system in such a way that the possibility of deadlock is excluded. The indirect methods prevent the occurrence of one of three necessary conditions of deadlock i.e., mutual exclusion, no pre-emption, and hold and wait. The direct method prevents the occurrence of circular wait. **Prevention techniques – Mutual exclusion –** are supported by the OS. **Hold and Wait –** the condition can be prevented by requiring that a process requests all its required resources at one time and blocking the process until all of its requests can be granted at the same time simultaneously. But this prevention does not yield good results because:

- long waiting time required
- inefficient use of allocated resource
- A process may not know all the required resources in advance

No pre-emption – techniques for 'no preemption are:

- If a process that is holding some resource, requests another resource that cannot be immediately allocated to it, all resources currently being held are released and if necessary, request again together with the additional resource.
- If a process requests a resource that is currently held by another process, the OS may preempt the second process and require it to release its resources. This works only if both processes do not have the same priority.

Circular wait One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in increasing order of enumeration, i.e., if a process has been allocated resources of type R, then it may subsequently request only those resources of types following R in ordering.

2. Deadlock avoidance (Banker's Algorithm)

The deadlock avoidance Algorithm works by actively looking for potential deadlock situations before they occur. It does this by tracking the resource usage of each process and identifying conflicts that could potentially lead to a deadlock. If a potential deadlock is identified, the algorithm will take steps to resolve the conflict, such as rolling back one of the processes or pre-emptively allocating resources to other processes. The Deadlock Avoidance Algorithm is designed to minimize the chances of a deadlock occurring, although it cannot guarantee that a deadlock will never occur. This approach allows the three necessary conditions of deadlock but makes judicious choices to assure that the deadlock point is never reached. It allows more concurrency than avoidance detection A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to deadlock. It requires knowledge of future process requests. Two techniques to avoid deadlock:

1. Process initiation denial
2. Resource allocation denial

Advantages of deadlock avoidance techniques:

- Not necessary to pre-empt and rollback processes
- Less restrictive than deadlock prevention

Disadvantages:

- Future resource requirements must be known in advance
- Processes can be blocked for long periods
- Exists a fixed number of resources for allocation

The Banker's Algorithm

The Banker's Algorithm is based on the concept of resource allocation graphs. A resource allocation graph is a directed graph where each node represents a process, and each edge represents a resource. The state of the system is represented by the current allocation of resources between processes. For example, if the system has three processes, each of which is using two resources, the resource allocation graph would look like this:

Processes A, B, and C would be the nodes, and the resources they are using would be the edges connecting them. The Banker's Algorithm works by analyzing the state of the system and determining if it is in a safe state or at risk of entering a deadlock.

To determine if a system is in a safe state, the Banker's Algorithm uses two matrices: the available matrix and the need matrix. The available matrix contains the amount of each resource currently available. The need matrix contains the amount of each resource required by each process.

The Banker's Algorithm then checks to see if a process can be completed without overloading the system. It does this by subtracting the amount of each resource used by the process from the available matrix and adding it to the need matrix.

If the result is in a safe state, the process is allowed to proceed, otherwise, it is blocked until more resources become available.

The Banker’s Algorithm is an effective way to prevent deadlocks in multiprogramming systems. It is used in many operating systems, including Windows and Linux. In addition, it is used in many other types of systems, such as manufacturing systems and banking systems.

The Banker’s Algorithm is a powerful tool for resource allocation problems, but it is not foolproof. It can be fooled by processes that consume more resources than they need, or by processes that produce more resources than they need. Also, it can be fooled by processes that consume resources in an unpredictable manner. To prevent these types of problems, it is important to carefully monitor the system to ensure that it is in a safe state.

3. Deadlock detection

Deadlock detection is used by employing an algorithm that tracks the circular waiting and kills one or more processes so that the deadlock is removed. The system state is examined periodically to determine if a set of processes is deadlocked. A deadlock is resolved by aborting and restarting a process, relinquishing all the resources that the process held.

- This technique does not limit resource access or restrict process action.
- Requested resources are granted to processes whenever possible.
- It never delays the process initiation and facilitates online handling.
- The disadvantage is the inherent pre-emption losses.

4. Deadlock ignorance

In the Deadlock ignorance method, the OS acts like the deadlock never occurs and completely ignores it even if the deadlock occurs. This method only applies if the deadlock occurs very rarely. The algorithm is very simple. It says” if the deadlock occurs, simply reboot the system and act like the deadlock never occurred.” That’s why the algorithm is called the **Ostrich Algorithm**.

Advantages:

- Ostrich Algorithm is relatively easy to implement and is effective in most cases.
- It helps in avoiding the deadlock situation by ignoring the presence of deadlocks.

Disadvantages:

- Ostrich Algorithm does not provide any information about the deadlock situation.
- It can lead to reduced performance of the system as the system may be blocked for a long time.
- It can lead to a resource leak, as resources are not released when the system is blocked due to deadlock.