# DEADLOCK

Deadlock is a problem in operating systems that happens when two or more processes are waiting for each other to release a resource, but neither of them does so they get stuck and can't continue. This can happen when each process is holding onto a resource that the other process needs in order to complete its task. It's like a traffic jam where all the cars are waiting for each other to move, but none of them can move because they're all waiting. In a deadlock, none of the processes can continue, so the system is effectively stuck.

# System Model

A system model is a structure that includes a fixed number of resources that are shared among multiple processes. These resources can be of different types, such as memory space, CPU cycles, and I/O devices like printers and keyboards. Each type of resource has a specific number of identical instances available.

Under the normal mode of operation, a process may utilize a resource in  
only this sequence request, use and then release.

When a process needs to use a resource, it first requests it. If the resource is available, the process can use it. However, if the resource is already being used by another process, the requesting process must wait until it becomes available. Once the process is done using the resource, it releases it so that other processes can use it.

Having a system model helps ensure that resources are allocated fairly among competing processes and prevents conflicts and deadlocks.

# Necessary Conditions

A deadlock happens when multiple processes are blocked because each process is holding a resource and waiting for another process to release another resource. There are four necessary conditions for a deadlock to occur:

1. Mutual Exclusion: Only one process can use a resource at a time.

2. Hold and wait: A process is holding at least one resource and waiting to acquire other resources held by other processes.

3. No Preemption: The resource cannot be taken away from a process until it's done with it voluntarily.

4. Circular Wait: A set of processes are waiting for each other in a circular fashion, creating a situation where they will wait forever.

These conditions create a situation where none of the processes can proceed, and the system is deadlocked.

# Methods for Handling Deadlocks

* **Deadlock Prevention**
* **Deadlock Avoidance**
* **Deadlock Detection and Recovery**

The first two methods are used to ensure the system never enters a deadlock.

To handle deadlock situations in an operating system, there are three methods you can use.

The first one is to prevent or avoid deadlocks by employing a protocol that ensures the system never enters a deadlock state.

The second method is to let the system enter a deadlock condition, detect it, and then recover from it.

The third method is to ignore the issue and assume that deadlocks never occur in the system.

However, it is recommended to use the first method to deal with deadlocks as it ensures that the system does not enter a deadlock state in the first place.

## Deadlock Prevention

Deadlock prevention means avoiding situations where multiple processes are waiting for each other to release resources, leading to a deadlock. This can be done by avoiding at least one of the four necessary conditions for a deadlock: mutual exclusion, hold and wait, no preemption, and circular wait.

### Mutual Exclusion:

The first condition is mutual exclusion, which means that only one process can use a resource at a time.

Some resources like read-only files can be accessed by multiple processes simultaneously without causing deadlocks, but others like printers or tape drives require exclusive access, which can lead to deadlocks.

### Hold and Wait:

The second condition that can cause deadlocks is Hold and Wait. This means that a process is holding one or more resources while waiting for another resource to become available.

To prevent this condition, you can use a few strategies:

Require processes to request all resources they need at the same time. However, this can be inefficient if a process needs some resources later in its execution.

Make processes release resources they hold before requesting new ones, and then request all resources they need together. However, this can be problematic if a process needs to hold a resource to partially complete an operation.

If a resource is in high demand, either of the above methods can result in starvation.

By preventing Hold and Wait, you can avoid deadlocks.

### No Preemption:

The third condition that can cause deadlocks is called No Preemption. This means that a process cannot be forced to give up a resource it is using.

To avoid deadlocks in this situation, it can be helpful to use preemption of process resource allocations. There are two ways to do this:

- If a process is waiting for a new resource and cannot get it, all the resources it was using before can be taken away from it. This will make the process ask for all resources it needs together in one go.

- If a resource is not available and some processes are waiting for it, the system can take away some resources from one of the processes to give to the process that needs the new resource.

However, these methods might not work for some devices like printers and tape drives, where it's not easy to save and restore the state of the resources being used.

### Circular Wait :

The fourth condition that can cause deadlocks is Circular Wait. This means that there is a cycle of processes, each holding one resource and waiting for another resource that is held by a different process.

To prevent circular waits, you can number all resources and require processes to request resources in a strictly increasing or decreasing order. In other words, before a process requests resource Rj, it must release all resources Ri such that i >= j.

However, determining the relative ordering of resources can be a challenge in this scheme. It requires careful planning and management of the resources to ensure that the ordering is correct and deadlock-free.

## Deadlock Avoidance

To avoid deadlocks, we can use a strategy called "Avoidance". But to use this strategy, we have to assume that we know all the resources that a process will need before it starts running.

In both prevention and avoidance, we can make sure that the data is correct, but the performance of the system might decrease.

To avoid deadlocks, the operating system can keep track of the maximum resources that a process will need and the free resources available at any given time. The operating system then tries to allocate resources according to the process requirements and checks whether this allocation can lead to a safe or unsafe state.

If the resource allocation can lead to an unsafe state, the operating system will not proceed with the allocation sequence. This way, the operating system can avoid deadlocks by making sure that the resource allocation is safe before proceeding with it.

### Safe State

In a Safe State, all the processes in the system can complete their execution without entering a deadlock. The Operating System achieves this by ensuring that the allocation of resources to processes does not result in a deadlock.

In the safe state, the Operating System knows the maximum resource requirements of each process and the currently allocated resources. It also knows the free resources available at current time. By allocating resources to processes in a way that ensures that the system remains in a safe state, the Operating System prevents a deadlock.

### Unsafe State

If there is only one available resource at the beginning, none of the processes can complete their task, and this is called an unsafe state. This means that the Operating System cannot prevent processes from requesting resources that may lead to a deadlock. A safe state is achieved when the Operating System is able to allocate all resources needed by the processes in any order, without causing a deadlock. However, an unsafe state does not always result in a deadlock, it may or may not lead to a deadlock depending on the situation.