## Table of contents



## List of Figures

# Deadlock

A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function. In operating systems, a deadlock occurs when two or more processes are unable to proceed because each is waiting for the other to release a resource. For example, if two processes are trying to access the same resource at the same time, a deadlock may occur.

Let's consider a simple example with two processes, P1 and P2, and two resources, R1 and R2. Each process needs access to both resources to complete its task.

1. P1 have R1.
2. P2 have R2.
3. P1 now needs R2 but is blocked because P2 is holding it.
4. P2 tires to acquire R1 but is blocked because P1 is holding it.

Both processes are now deadlocked. Neither can proceed because each is holding a resource the other needs, and neither is willing to release its resource.

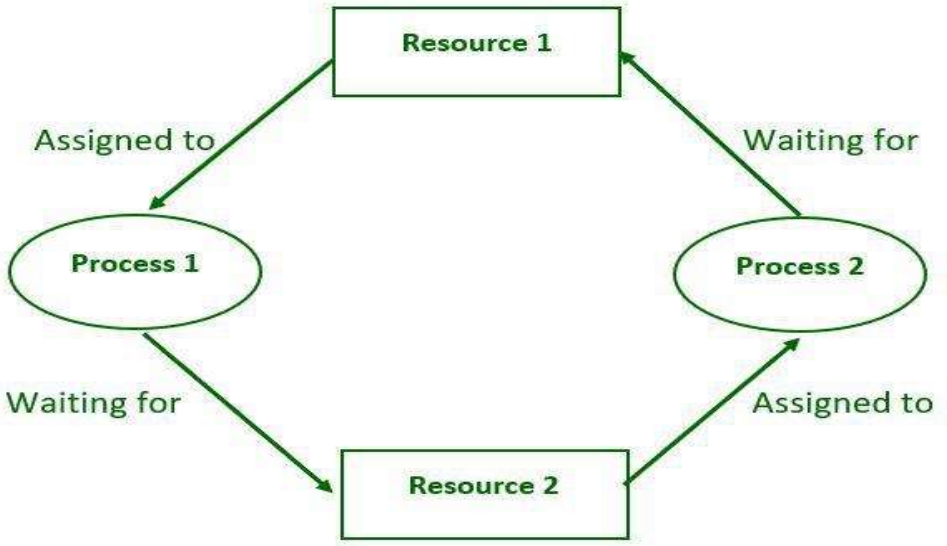


Figure: Deadlock in Operating system

*Figure 1: Deadlock*

## Necessary conditions for Deadlock

Deadlock occur when four conditions hold simultaneously in a system:

1. Mutual Exclusion: At least one resource must be held in a non-shareable mode; that is, only one process can use the resource at any given time. It's like a bathroom that can be used by only one person at a time, if someone is using it, others have to wait until it's free. This property ensures that the resource is not shared concurrently, which can prevent conflicts and errors.

2. Hold and Wait: Hold and wait refers to a situation where a process holds one resource and is waiting to acquire another resource that is currently held by another process. Imagine you need a pen to write, but your friend ahs the only pen and needs paper, which you have. You're both waiting for each other to give up your resource, creating a hold and wait situation.

3. No Preemption: It means that once a process holds a resource, it cannot be taken away from it forcibly. The process must voluntarily release the resource once it has finished using it. In simpler terms, it's like borrowing a book from a library. Once you have the book, the library can't take it back until you return it.

4. Circular Wait: Circular wait occurs in a system when processes are waiting for resources in a circular chain. If P1 holds R1 and is waiting for R2, while P2 holds R2 and waiting for R1, then a circular wait condition exists: P1->R1->P2->R2->P1. In this case, both processes are waiting for a resource held by the other, which can lead to a deadlock.

## Methods for handling Deadlock

There are several methods for handling deadlocks in operating systems:

1. Deadlock Prevention: Modify the system to ensure that at least one of the necessary conditions for deadlock cannot occur. This can involve resource allocation strategies, such as only allocation resources if they are available, or requiring a processes to request all their resources at once.

2. Deadlock Detection and Recovery: Periodically check the system for the presence of a deadlock, if one is detected, the system can recover by preemption, resources form some processes, killing processes, or rolling back processes to a previous state.

3. Deadlock Avoidance: Use information about the current state of the system to decide whether a particular resource allocation request should be granted or denied. This approach requires careful scheduling and resource allocation request should be granted or denied. This approach requires careful scheduling and resource allocation algorithms to avoid potential deadlocks. The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular wait condition.

4. Timeouts: Use timeouts to limit the amount of time a process can wait for a resource. If a timeout occurs, the process can be aborted or put into a state where it can request resources again. Timeouts are like setting a timer. If you're waiting for something for too long, the timer goes off and you stop waiting. This helps prevent situations where you wait forever for something that might never happen.

5. Resource Allocation Graph: A resource allocation graph is like a visual map showing which processes are using which resources. Each process and resource are represented by a node, and if a process is using a resource, there's an arrow from the process to the resource node. In deadlock detection, the graph is checked for certain patterns, like circles, which indicate the processes are waiting for each other's resources. If such pattern is found, it suggests a deadlock might occur, and actions can be taken to prevent it.

*It is important to understand that these methods don't guarantee to prevent deadlocks. The choice of prevention strategy depends on the specific requirements and constraints of the system.*

## Conclusion

In conclusion, deadlocks are a crucial concept in operating systems, occurring when processes are stuck waiting for resources that are held by other processes. Understanding the conditions that lead to deadlocks, such as mutual exclusion, hold and wait, no preemption, and circular wait, is essential for effective system design and management. Various methods, including deadlock prevention, avoidance, detection and recovery, can be employed to handle deadlocks and ensure system stability and efficiency. As technology advances and systems become more complex, addressing deadlocks remains a critical aspect of operating system design and management.

## References


1. Google.com
2. Wikipedia.org
3. Notebook
4. ChatGPT