A deadlock is a situation where a set of processes is blocked because each process is holding a resource and waiting for another resource acquired by some other process. In this article, we will discuss deadlock, its necessary conditions, etc. in detail.

- **Deadlock** is a situation in computing where two or more processes are unable to proceed because each is waiting for the other to release resources.
- Key concepts include mutual exclusion, resource holding, circular wait, and no preemption.

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. This is a practical example of deadlock.

How Does Deadlock occur in the Operating System?

Before going into detail about how deadlock occurs in the Operating System, let's first discuss how the Operating System uses the resources present. A process in an operating system uses resources in the following way.

- **Requests a resource**
- **Use the resource**
- **Releases the resource**

A 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

There are several examples of deadlock. Some of them are mentioned below.

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.

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.

Necessary Conditions for Deadlock in OS

Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)

- **Mutual Exclusion:** Only one process can use a resource at any given time i.e. the resources are non-sharable.
- **Hold and Wait:** A process is holding at least one resource at a time and is waiting to acquire other resources held by some other process.
- **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
- **Circular Wait:** set of processes are waiting for each other in a circular fashion. For example, lets say there are a set of processes {P0*P*0 ,P1*P*1,P2*P*2,P3*P*3} such that P0*P*0 depends on P1*P*1, P1*P*1 depends on P2*P*2, P2*P*2 depends on P3*P*3 and P3*P*3 depends on P0*P*0. This creates a circular relation between all these processes and they have to wait forever to be executed.

Methods of Handling Deadlocks in Operating System

There are three ways to handle deadlock:

1. Deadlock Prevention or Avoidance

2. Deadlock Detection and Recovery
3. Deadlock Ignorance

Deadlock Prevention or Avoidance

Deadlock Prevention and Avoidance is the one of the methods for handling deadlock. First, we will discuss Deadlock Prevention, then Deadlock Avoidance.

Deadlock Prevention

In deadlock prevention the aim is to not let full-fill one of the required condition of the deadlock. This can be done by this method:

(i) Mutual Exclusion

We only use the Lock for the non-share-able resources and if the resource is share- able (like read only file) then we not use the locks here. That ensure that in case of share -able resource , multiple process can access it at same time. Problem- Here the problem is that we can only do it in case of share-able resources but in case of no-share-able resources like printer , we have to use Mutual exclusion.

(ii) Hold and Wait

To ensure that Hold and wait never occurs in the system, we must guarantee that whenever process request for resource , it does not hold any other resources.

- we can provide the all resources to the process that is required for it's execution before starting it's execution . **problem –** for example if there are three resource that is required by a process and we have given all that resource before starting execution of process then there might be a situation that initially we required only two resource and after one hour we want third resources and this will cause starvation for the another process that wants this resources and in that waiting time that resource can allocated to other process and complete their execution.
- We can ensure that when a process request for any resources that time the process does not hold any other resources. Ex- Let there are three resources DVD, File and Printer . First the process request for DVD and File for the copying data into the file and let suppose it is going to take 1 hour and after it the process free all resources then again request for File and Printer to print that file.

(iii) No Preemption

If a process is holding some resource and requestion other resources that are acquired and these resource are not available immediately then the resources that current process is holding are preempted. After some time process again request for the old resources and other required resources to re-start.

For example – Process p1 have resource r1 and requesting for r2 that is hold by process p2. then process p1 preempt r1 and after some time it try to restart by requesting both r1 and r2 resources.

Problem – This can cause the Live Lock Problem .

Live Lock **:** Live lock is the situation where two or more processes continuously changing their state in response to each other without making any real progress.

Example:

- suppose there are two processes p1 and p2 and two resources r1 and r2.
- Now, p1 acquired r1 and need r2 & p2 acquired r2 and need r1.
- so according to above method- Both p1 and p2 detect that they can't acquire second resource, so they release resource that they are holding and then try again.
- continuous cycle- p1 again acquired r1 and requesting to r2 p2 again acquired r2 and requesting to r1 so there is no overall progress still process are changing there state as they preempt resources and then again holding them. This the situation of Live Lock.

(iv) Circular Wait:

To remove the circular wait in system we can give the ordering of resources in which a process needs to acquire.

Ex: If there are process p1 and p2 and resources r1 and r2 then we can fix the resource acquiring order like the process first need to acquire resource r1 and then resource r2. so the process that acquired r1 will be allowed to acquire r2 , other process needs to wait until r1 is free.

This is the Deadlock prevention methods but practically only fourth method is used as all other three condition removal method have some disadvantages with them.

Deadlock Avoidance

Avoidance is kind of futuristic. By using the strategy of "Avoidance", we have to make an assumption. We need to ensure that all information about resources that the process will need is known to us before the execution of the process. We use Banker's algorithm to avoid deadlock.

In prevention and avoidance, we get the correctness of data but performance decreases.

Deadlock Detection and Recovery

If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:

1. In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
2. If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.

In Deadlock detection and recovery, we get the correctness of data but performance decreases.

Deadlock Detection

Deadlock detection is a process in computing where the system checks if there are any sets of processes that are stuck waiting for each other

indefinitely, preventing them from moving forward. In simple words, deadlock detection is the process of finding out whether any process are stuck in loop or not. There are several algorithms like;

- Resource Allocation Graph
- Banker's Algorithm

These algorithms helps in detection of deadlock in Operating System.

Deadlock Recovery

There are several Deadlock Recovery Techniques:

- Manual Intervention
- Automatic Recovery
- Process Termination
- Resource Preemption

1. Manual Intervention

When a deadlock is detected, one option is to inform the operator and let them handle the situation manually. While this approach allows for human judgment and decision-making, it can be time-consuming and may not be feasible in large-scale systems.

2. Automatic Recovery

An alternative approach is to enable the system to recover from deadlock automatically. This method involves breaking the deadlock cycle by either aborting processes or preempting resources. Let's delve into these strategies in more detail.

3. Process Termination

- **Abort all Deadlocked Processes**
  This approach breaks the deadlock cycle, but it comes at a significant cost. The processes that were aborted may have executed for a considerable amount of time, resulting in the loss of partial computations. These computations may need to be recomputed later.
- **Abort one process at a time**
  Instead of aborting all deadlocked processes simultaneously, this strategy involves selectively aborting one process at a time until the deadlock cycle is eliminated. However, this incurs overhead as a deadlock-detection algorithm must be invoked after each process termination to determine if any processes are still deadlocked.
- **Factors for choosing the termination order:**
  The process's priority
  Completion time and the progress made so far
  Resources consumed by the process
  Resources required to complete the process
  Number of processes to be terminated
  Process type (interactive or batch)

4. Resource Preemption

- **Selecting a Victim**
  Resource preemption involves choosing which resources and

processes should be preempted to break the deadlock. The selection order aims to minimize the overall cost of recovery. Factors considered for victim selection may include the number of resources held by a deadlocked process and the amount of time the process has consumed.

- **Rollback**
  If a resource is preempted from a process, the process cannot continue its normal execution as it lacks the required resource. Rolling back the process to a safe state and restarting it is a common approach. Determining a safe state can be challenging, leading to the use of total rollback, where the process is aborted and restarted from scratch.
- **Starvation Prevention**
  To prevent resource starvation, it is essential to ensure that the same process is not always chosen as a victim. If victim selection is solely based on cost factors, one process might repeatedly lose its resources and never complete its designated task. To address this, it is advisable to limit the number of times a process can be chosen as a victim, including the number of rollbacks in the cost factor.

Deadlock Ignorance

If a deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take. we use the ostrich algorithm for deadlock ignorance.

“In Deadlock, ignorance performance is better than the above two methods but the correctness of data is not there.”

Safe State

A safe state can be defined as a state in which there is no deadlock. It is achievable if:

- If a process needs an unavailable resource, it may wait until the same has been released by a process to which it has already been allocated. if such a sequence does not exist, it is an unsafe state.
- All the requested resources are allocated to the process.

Difference between Starvation and Deadlocks

Aspect	Deadlock	Starvation
Definition	A condition where two or more processes are blocked forever, each waiting for a resource held by another.	A condition where a process is perpetually denied necessary resources, despite resources being available.
Resource Availability	Resources are held by processes involved in the deadlock.	Resources are available but are continuously allocated to other processes.

Aspect	Deadlock	Starvation
Cause	Circular dependency between processes, where each process is waiting for a resource from another.	Continuous preference or priority given to other processes, causing a process to wait indefinitely.
Resolution	Requires intervention, such as aborting processes or preempting resources to break the cycle.	Can be mitigated by adjusting scheduling policies to ensure fair resource allocation.

What is Deadlock?

Deadlock is a situation in computing where two or more processes are unable to proceed because each is waiting for the other to release resources. Key concepts include mutual exclusion, resource holding, circular wait, and no preemption. Consider a practical 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.

Deadlock is an infinite Process it means that once a process goes into deadlock it will never come out of the loop and the process will enter for an indefinite amount of time. There are only detection, resolution, and prevention techniques. But, there are no Deadlock-stopping techniques.

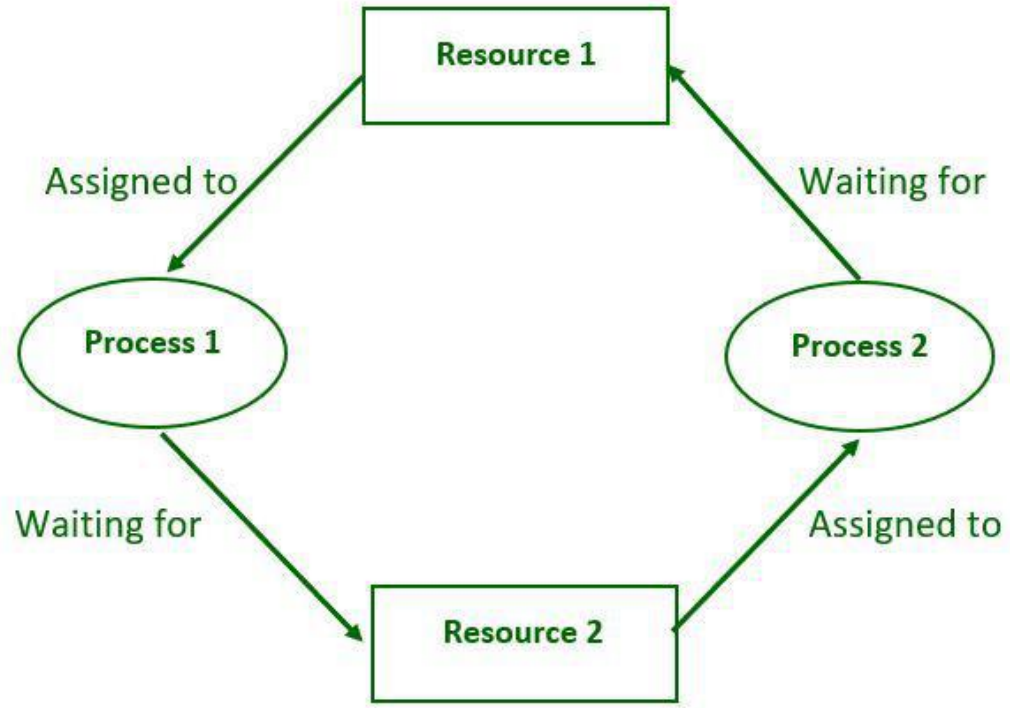


Figure: Deadlock in Operating system

Deadlock

How a Deadlock Can Occur?

Consider a simple scenario that includes two processes (Process A and Process B) and two resources (Resource 1 and Resource 2). Let's see that both processes begin execution at the same time.

- Process 1 obtains Resource 1.
- Process 2 obtains Resource 2.

We are currently in the following situation:

- Process 1 possesses Resource 1.
- Process 2 possesses Resource 2.

Let us now see what happens next: Process 1 requires Resource 2 to continue execution but is unable to do so because Process 2 is currently holding Resource 2. Similarly, Process 2 requires Resource 1 to continue execution but is unable to do so because Process 1 is currently holding Resource 1. Both processes are now stuck in a loop:

- Process 1 is awaiting Resource 2 from Process 2.
- Process 2 is awaiting Resource 1 from Process 1.

We have a deadlock because neither process can release the resource it is holding until it completes its task, and neither can proceed without the resource the other process is holding. Both processes are effectively

“deadlocked,” unable to move forward. To break the deadlock and free up resources for other processes in this situation, an external intervention, such as the operating system killing one or both processes, would be required. Deadlocks are undesirable in operating systems because they waste resources and have a negative impact on overall system performance and responsiveness. To prevent deadlocks, various resource allocation and process scheduling algorithms, such as deadlock detection and avoidance, are employed.

Necessary Conditions for the Occurrence of a Deadlock

Let’s explain all four conditions related to deadlock in the context of the scenario with two processes and two resources:

- Mutual Exclusion
- Hold and Wait
- No Pre Emption
- Circular Wait

1. Mutual Exclusion

Mutual Exclusion condition requires that at least one resource be held in a non-shareable mode, which means that only one process can use the resource at any given time. Both Resource 1 and Resource 2 are non-shareable in our scenario, and only one process can have exclusive access to each resource at any given time. As an example:

- Process 1 obtains Resource 1.
- Process 2 acquires Resource 2.

2. Hold and Wait

The hold and wait condition specifies that a process must be holding at least one resource while waiting for other processes to release resources that are currently held by other processes. In our example,

- Process 1 has Resource 1 and is awaiting Resource 2.
- Process 2 currently has Resource 2 and is awaiting Resource 1.
- Both processes hold one resource while waiting for the other, satisfying the hold and wait condition.

3. No Preemption

Preemption is the act of taking a resource from a process before it has finished its task. According to the no preemption condition, resources cannot be taken forcibly from a process a process can only release resources voluntarily after completing its task.

For example – Process p1 have resource r1 and requesting for r2 that is hold by process p2. then process p1 can not preempt resource r2 until process p2 can finish his execution. After some time it try to restart by requesting both r1 and r2 resources.

Problem – This can cause the Live Lock Problem .

What is Live Lock?

Live lock is the situation where two or more processes continuously changing their state in response to each other without making any real progress. Example:

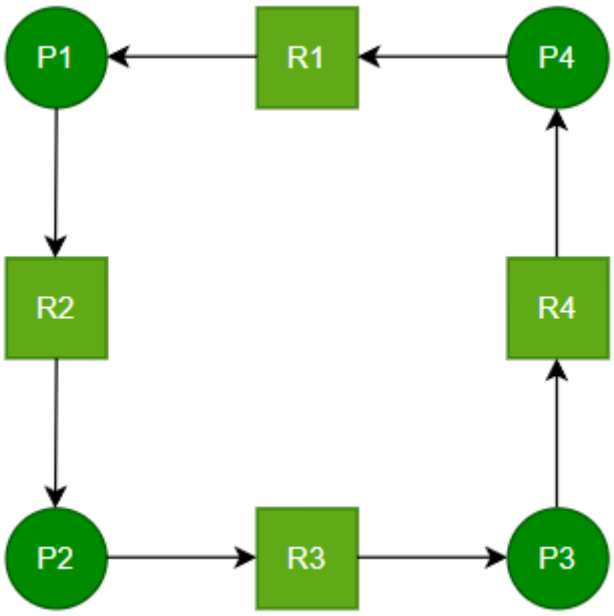
- Suppose there are two processes 1 and 2 and two resources r1 and r2.
- Now, p1 acquired r1 and need r2 & p2 acquired r2 and need r1.
- so according to above method- Both p1 and p2 detect that they can't acquire second resource, so they release resource that they are holding and then try again.
- continuous cycle- p1 again acquired r1 and requesting to r2 p2 again acquired r2 and requesting to r1 so there is no overall progress still process are changing there state as they preempt resources and then again holding them. This the situation of Live Lock.

4. Circular Wait

Circular wait is a condition in which a set of processes are waiting for resources in such a way that there is a circular chain, with each process in the chain holding a resource that the next process needs. This is one of the necessary conditions for a **deadlock** to occur in a system.

Example: Imagine four processes—**P1**, **P2**, **P3**, and **P4**—and four resources—**R1**, **R2**, **R3**, and **R4**.

- **P1** is holding **R1** and waiting for **R2** (which is held by P2).
- **P2** is holding **R2** and waiting for **R3** (which is held by P3).
- **P3** is holding **R3** and waiting for **R4** (which is held by P4).
- **P4** is holding **R4** and waiting for **R1** (which is held by P1).



Circular wait example

This forms a circular chain where every process is waiting for a resource held by another, creating a situation where no process can proceed, leading to a deadlock.

Conclusion

Deadlocks are major problems in computers in which two or more processes remain blocked forever, each waiting for the other to release resources. A deadlock requires four conditions: mutual exclusion, hold and wait, no preemption, and circular wait. Deadlocks consume resources and reduce system performance, hence their avoidance, detection, and resolution are critical in operating systems. Deadlocks are managed and mitigated using a variety of algorithms and tactics, which ensure system stability and efficiency.

Deadlock prevention and avoidance are strategies used in computer systems to ensure that different processes can run smoothly without getting stuck waiting for each other forever. Think of it like a traffic system where cars (processes) must move through intersections (resources) without getting into a gridlock.

Necessary Conditions for Deadlock

- Mutual Exclusion
- Hold and Wait
- No Preemption
- Circular Wait

Please refer Conditions for Deadlock in OS for details.

Deadlock Prevention

We can prevent a Deadlock by eliminating any of the above four conditions.

Eliminate Mutual Exclusion

It is not possible to violate mutual exclusion because some resources, such as the tape drive, are inherently non-shareable. For other resources, like printers, we can use a technique called Spooling (Simultaneous Peripheral Operations Online).

In spooling, when multiple processes request the printer, their jobs (instructions of the processes that require printer access) are added to the queue in the spooler directory. The printer is allocated to jobs on a First-Come, First-Served (FCFS) basis. In this way, a process does not have to wait for the printer and can continue its work after adding its job to the queue.

Eliminate Hold and Wait

Hold and wait is a condition in which a process holds one resource while simultaneously waiting for another resource that is being held by a different process. The process cannot continue until it gets all the required resources.

Hold & Wait

There are two ways to eliminate hold and wait:

- **By eliminating wait**: The process specifies the resources it requires in advance so that it does not have to wait for allocation after execution starts.
  For Example, Process1 declares in advance that it requires both Resource1 and Resource2.
- **By eliminating hold**: The process has to release all resources it is currently holding before making a new request.
  For Example: Process1 must release Resource2 and Resource3 before requesting Resource1.

Eliminate No Preemption

Preemption is temporarily interrupting an executing task and later resuming it. Two ways to eliminate No Preemption:

- **Processes must release resources voluntarily**: A process should only give up resources it holds when it completes its task or no longer needs them.
- **Avoid partial allocation**: Allocate all required resources to a process at once before it begins execution. If not all resources are available, the process must wait.

Eliminate Circular Wait

To eliminate circular wait for deadlock prevention, we can use **order on resource allocation**.

- Assign a unique number to each resource.

- Processes can only request resources in an increasing order of their numbers.

This prevents circular chains of processes waiting for resources, as no process can request a resource lower than what it already holds.

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

Deadlock avoidance ensures that a resource request is only granted if it won't lead to deadlock, either immediately or in the future. Since the kernel can't predict future process behavior, it uses a conservative approach. Each process declares the maximum number of resources it may need. The kernel allows requests in stages, checking for potential deadlocks before granting them. A request is granted only if no deadlock is possible; otherwise, it stays pending. This approach is conservative, as a process may finish without using the maximum resources it declared.
Banker's Algorithm is the technique used for Deadlock Avoidance.

Banker's Algorithm

Bankers' Algorithm is a **resource allocation and deadlock avoidance algorithm** that tests all resource requests made by processes. It checks for the **safe state**, and if granting a request keeps the system in safe state, the request is allowed. However, if no safe state exists, the request is denied.

Inputs to Banker's Algorithm

- Max needs of resources by each process.
- Currently, allocated resources by each process.
- Max free available resources in the system.

The request will only be granted under the below condition

- If the request made by the process is less than equal to the max needed for that process.
- If the request made by the process is less than equal to the freely available resource in the system.

Timeouts

To avoid deadlocks caused by indefinite waiting, a timeout mechanism can be used to limit the amount of time a process can wait for a resource. If the help is unavailable within the timeout period, the process can be forced to release its current resources and try again later.

Example

Below is an example of a Banker's algorithm

Total resources in system:

A	B	C	D
6	5	7	6

Available system resources are:

A	B	C	D
3	1	1	2

Processes (currently allocated resources):

	A	B	C	D
P_1	1	2	2	1
P_2	1	0	3	3
P_3	1	2	1	0

Maximum resources we have for a process:

	A	B	C	D
P_1	3	3	2	2
P_2	1	2	3	4
P_3	1	3	5	0

Need = Maximum Resources Requirement – Currently Allocated Resources

	A	B	C	D
P_1	2	1	0	1
P_2	0	2	0	1
P_3	0	1	4	0