

Introduction of Deadlock in Operating System

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 .

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.

3. Assume
for
bytes, and
sequence of

P0	P1
wait(A);	wait(B)
wait(B);	wait(A)

the space is available
allocation of 200K
the following
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 {P0P0,P1P1,P2P2,P3P3} such that P0P0 depends on P1P1, P1P1 depends on P2P2, P2P2 depends on P3P3 and P3P3 depends on P0P0. 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 request 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.
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.