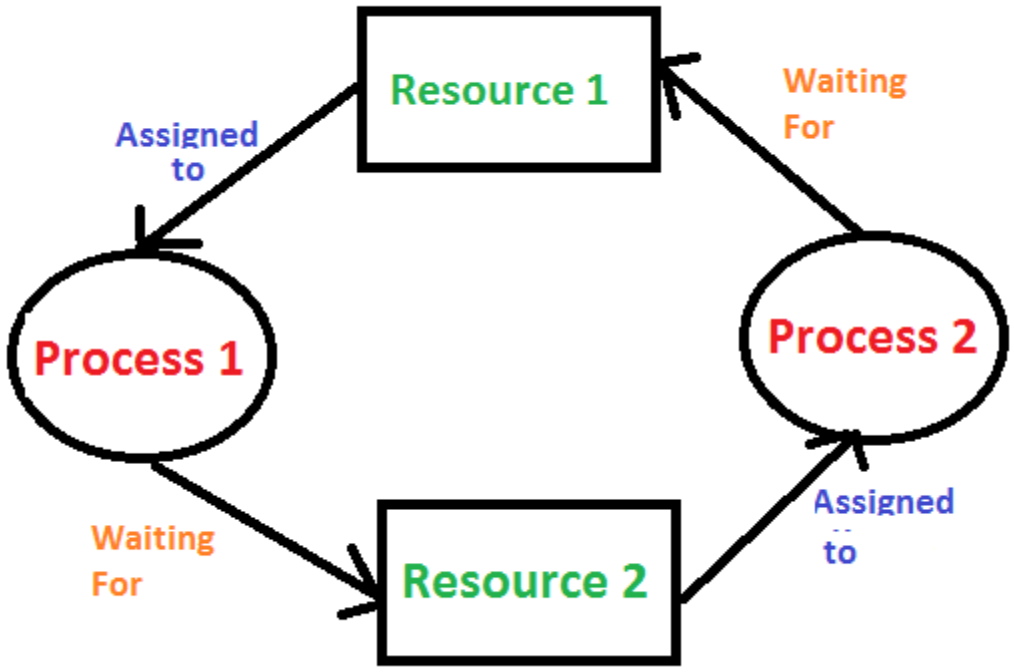
Introduction of Deadlock in Operating System

- 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. 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.

Deadlock can arise if **the** following four conditions hold simultaneously

Necessary Conditions for Deadlock

These four conditions must be met for a deadlock to happen in an operating system.

1. Mutual Exclusion

In this, two or more processes must compete for the same resources. There must be some resources that can only be used one process at a time. This means the resource is non-sharable. This could be a physical resource like a printer or an abstract concept like a lock on a shared **data structure**.

2. Hold and Wait

Hold and wait is when a process is holding a resource and waiting to acquire another resource that it needs but cannot proceed because another process is keeping the first resource. Each of these processes must have a hold on at least one of the resources it's requesting. If one process doesn't have a hold on any of the resources, it can't wait and will give up immediately.

3. No Preemption

Preemption means temporarily interrupting a task or process to execute another task or process. Preemption can occur due to an external event or internally within the system. If we take away the resource from the process that is causing deadlock, we can avoid deadlock. But is it a good approach? The answer is NO because that will lead to an inconsistent state. For example, if we take away memory from any process(whose data was in the process of getting stored) and assign it to some other process. Then will lead to an inconsistent state.

4. Circular Wait

The circular wait is when two processes wait for each other to release a resource they are holding, creating a deadlock. There must be a cycle in the graph below. As you can see, process 1 is holding on to a resource R1 that process 2 in the cycle is waiting for. This is an example of a circular wait.To better understand let's understand with another example. For example, Process A might be holding on to Resource X while waiting for

Resource Y, while Process B is holding on to Resource Y while waiting for Resource Z, and so on around the cycle.

Difference between Deadlock and Starvation

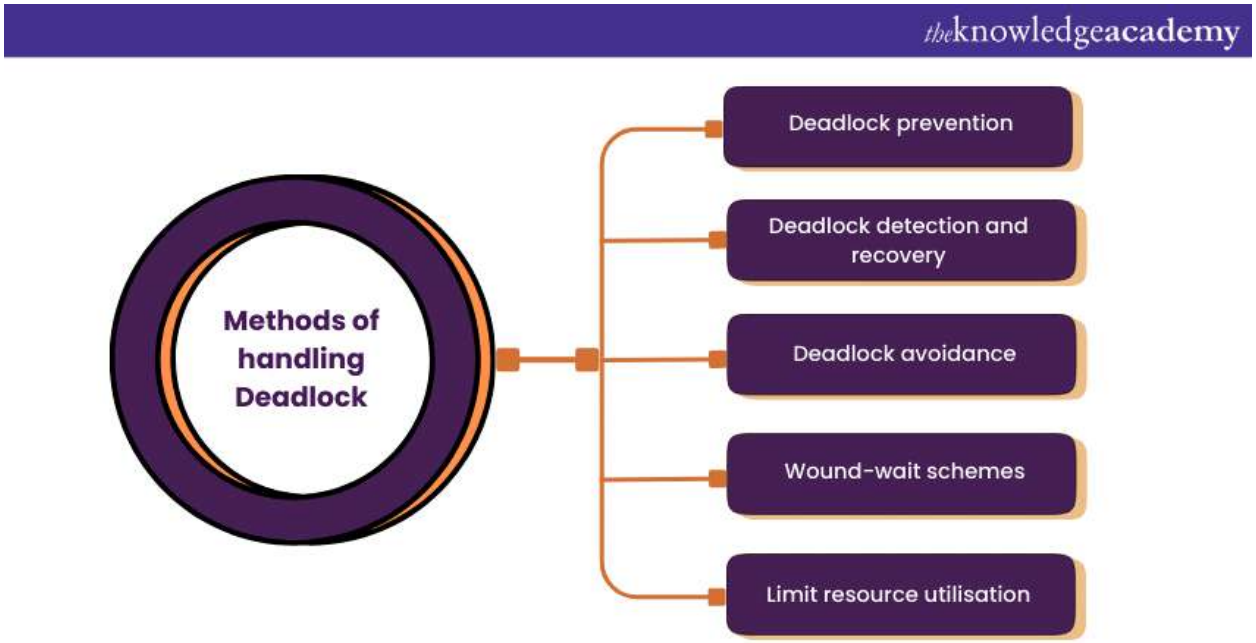
Deadlock	**Starvation**
In this, two or more processes are each waiting for the other to release a resource, and neither process is able to continue.	In this, a process is unable to obtain the resources it needs to continue running.
Different processes are unable to proceed because they are each waiting for the other to do something.	A process is unable to proceed due to the unavailability of that resource.
Deadlock is also called Circular wait.	Starvation is also called lived lock.
Avoiding the necessary conditions for deadlock can be prevented.	Starvation can be easily prevented by Aging.

What are the Consequences of a Deadlock?

When a deadlock occurs, it can cause your computer to freeze up, making it difficult to even restart. This can cause you to lose important work or data and in some cases, may even damage your computer.

To prevent a deadlock state, it's important to be aware of what causes deadlocks and how to avoid them.

Methods For Handling Deadlocks



1. Deadlock avoidance

Deadlock avoidance is the process of taking steps to prevent deadlock from occurring. Operating system uses the deadlock Avoidance method to ensure the system is in a safe state(when the system can allocate resources and can avoid being in a deadlock state). We have a Deadlock avoidance algorithm-**Banker's algorithm** for this. When a new process is to be executed, it requires some resources. So banker's algorithm needs to know

- How many resources the process could request
- Which processes hold many resources.
- How many resources the system has.

And accordingly, resources are being assigned if available resources are more than requested to avoid deadlock. Tell the operating system about the maximum number of resources a process can request to complete its execution. The deadlock avoidance graph(shown in fig-2) assesses the resource-allocation state to check if a circular wait situation is not occurring.

If a deadlock does occur, it can sometimes be resolved by terminating one of the processes involved. However, this can cause data loss or corruption, so it's always preferable to try and prevent the deadlock from happening in the first place.

Bankers algorithm Pseudocode:

1. In ***s***tarting all the processes are to be executed. Define two data structure finish and work:

Finish[n]=False.
Work=Available

Where n is a number of processes to be executed.

2. Find the process for which Finish[i]=False

And ***Need <=Work***(This means a request is valid as the number of requested resources of each resource type is less than the **available resources**, In case no such process is there then go to step
3. ***Work=Work+Allocation***
Finish[i]=True
Go to step 2 to find other processes

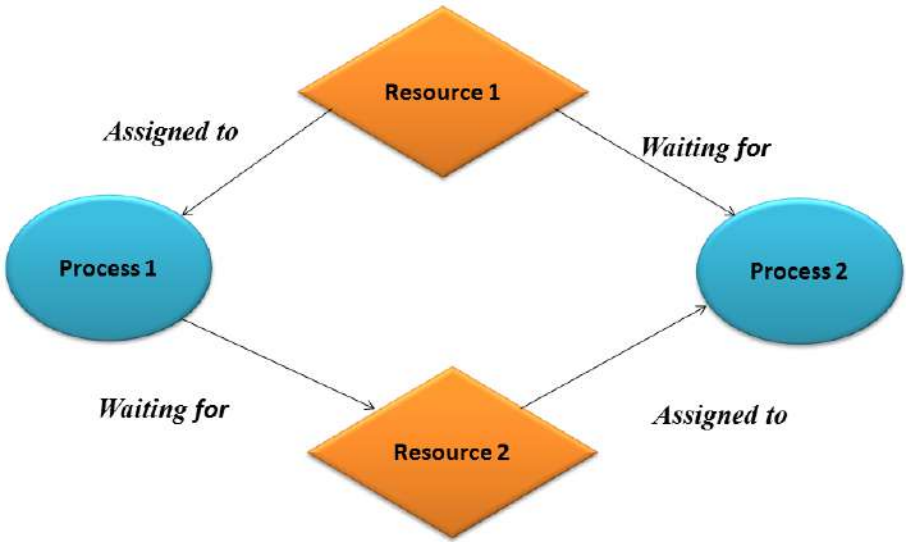
Any process says process 'i' finishes its execution. So that means the resources allocated to it previously, get free. So these resources are added to Work and Finish(i) of the process is set as true.

4. If ***Finish[i]=True*** for n processes then the system is in a safe state(If all the processes are executed in some sequence). Otherwise, it is in an unsafe state

2. Deadlock Detection

Detecting deadlocks is one of the most important steps in preventing them. A deadlock can happen anytime when two or more processes are trying to acquire a resource, and each process is waiting for other processes to release the resource.

The deadlock can be detected in the resource-allocation graph as shown in fig below.



This graph checks if there is a cycle in the **Resource Allocation Graph** and each resource in the cycle provides only one instance, If there is a cycle in this graph then the processes will be in a **deadlock state.**

So always remember detecting deadlocks is one of the most important steps in preventing them.

3. Deadlock Prevention

The best way to prevent deadlocks is by understanding how they form in the first place. Deadlock can be prevented by eliminating the necessary conditions for deadlock(explained above).

Some ways of prevention are as follows

1. **Preempting resources:** Take the resources from the process and assign them to other processes.
2. **Rollback:** When the process is taken away from the process, roll back and restart it.
3. **Aborting:** Aborting the deadlocked processes.
4. **Sharable resource:** If the resource is sharable, all processes will get all resources, and a deadlock situation won't come.

Conclusion

A deadlock is a situation that can occur in an operating system when two or more processes, each waiting for an event that only the other can cause, lock up the system so that no process can continue execution.

Deadlock Prevention

If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

Let's see how we can prevent each of the conditions.

1. Mutual Exclusion

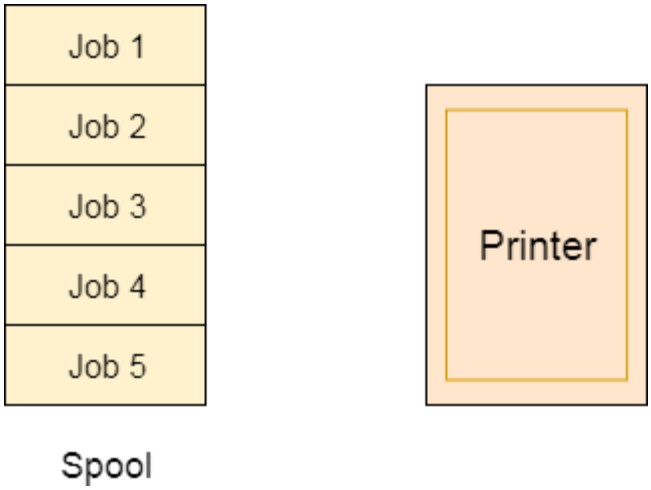
Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock.

If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.



Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.

2. After some point of time, there may arise a race condition between the processes to get space in that spool.

We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance. Therefore, we cannot violate mutual exclusion for a process practically.

2. Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)

This can be implemented practically if a process declares all the resources initially. However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

3. No Preemption

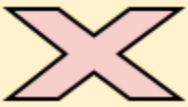
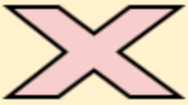
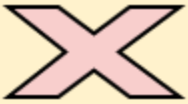
Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

4. Circular Wait

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	
Hold and Wait	Request for all the resources initially	
No Preemption	Snatch all the resources	
Circular Wait	Assign priority to each resources and order resources numerically	

Among all the methods, violating Circular wait is the only approach that can be implemented practically.

Deadlock Avoidance

A deadlock avoidance policy grants a resource request only if it can establish that granting the request cannot lead to a deadlock either immediately or in the future. The kernal lacks detailed knowledge about future behavior of processes, so it cannot accurately predict deadlocks. To facilitate deadlock avoidance under these conditions, it uses the following conservative approach: Each process declares the maximum number of resource units of each class that it may require. The kernal permits a process to request these resource units in stages- i.e. a few resource units at a time- subject to the maximum number declared by it and uses a worst case analysis technique to check for the possibility of future deadlocks. A request is granted only if there is no possibility of deadlocks; otherwise, it remains pending until it can be granted. This approach is conservative because a process may

complete its operation without requiring the maximum number of units declared by it.

Resource Allocation Graph

The resource allocation graph (RAG) is used to visualize the system's current state as a graph. The Graph includes all processes, the resources that are assigned to them, as well as the resources that each Process requests. Sometimes, if there are fewer processes, we can quickly spot a deadlock in the system by looking at the graph rather than the tables we use in Banker's algorithm. Deadlock avoidance can also be done with Banker's Algorithm.

Banker's Algorithm

Bankers's Algorithm is a resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, and after granting a request system remains in the safe state it allows the request, and if there is no safe state it doesn't allow the request made by the process.

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:

Total resources in system:

A B C D

6 5 7 6

The total number of resources are

Available system resources are:

A B C D

3 1 1 2

Available resources are

Processes (currently allocated resources):

	A	B	C	D
P1	1	2	2	1
P2	1	0	3	3
P3	1	2	1	0

Maximum resources we have for a process

Processes (maximum resources):

	A	B	C	D
P1	3	3	2	2
P2	1	2	3	4
P3	1	3	5	0

Need = Maximum Resources Requirement – Currently Allocated Resources.

Need = maximum resources - currently allocated resources.

Processes (need resources):

	A	B	C	D
P1	2	1	0	1
P2	0	2	0	1
P3	0	1	4	0

OR

Deadlock avoidance

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

A state of a system recorded at some random time is shown below.

Resources Assigned

Process	Type 1	Type 2	Type 3	Type 4
A	3	0	2	2
B	0	0	1	1
C	1	1	1	0
D	2	1	4	0

Resources still needed

Process	Type 1	Type 2	Type 3	Type 4
A	1	1	0	0
B	0	1	1	2
C	1	2	1	0
D	2	1	1	2

1. E = (7 6 8 4)
2. P = (6 2 8 3)
3. A = (1 4 0 1)

Above tables and vector E, P and A describes the resource allocation state of a system. There are 4 processes and 4 types of the resources in a system. Table 1 shows the instances of each resource assigned to each process.

Table 2 shows the instances of the resources, each process still needs. Vector E is the representation of total instances of each resource in the system.

Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use.

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfill the request of all processes then the state of the system is called unsafe.

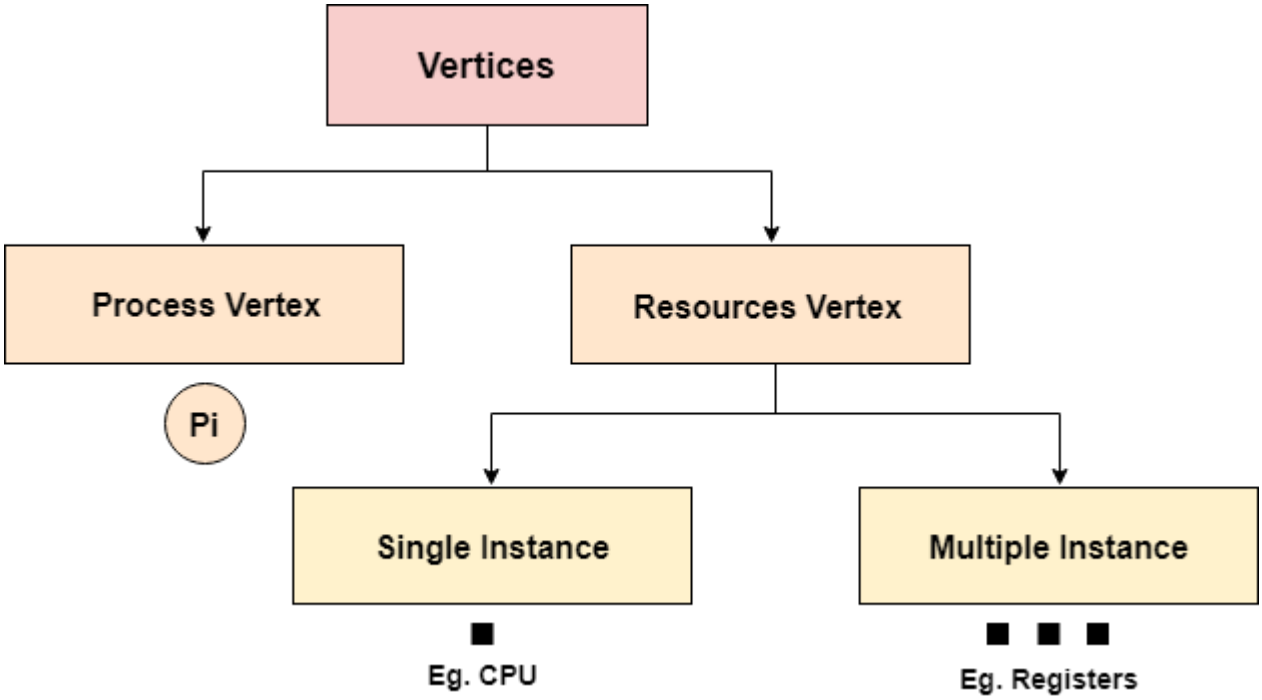
The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case if the resulting state is also a safe state.

Resource Allocation Graph

The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

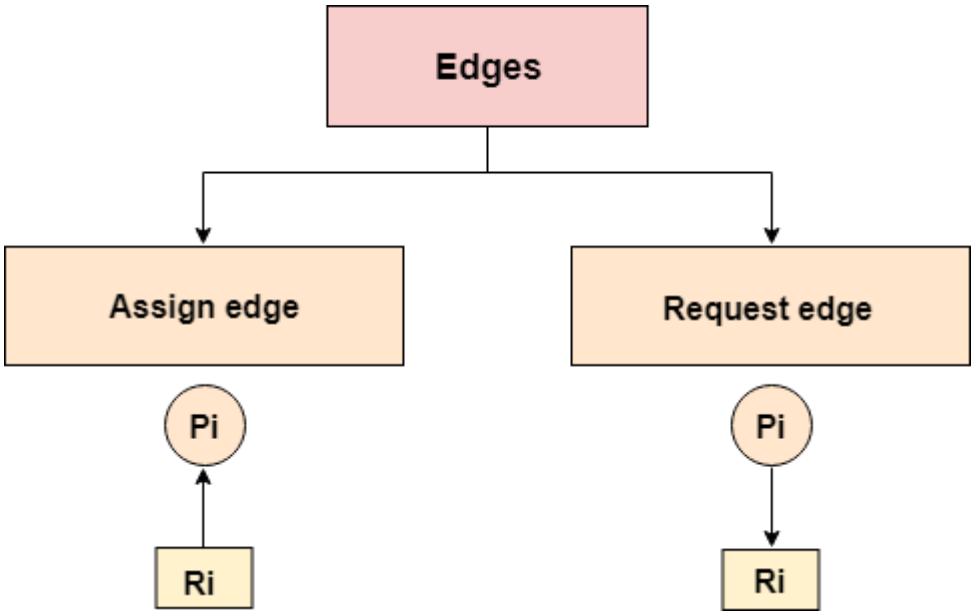
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.



Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.

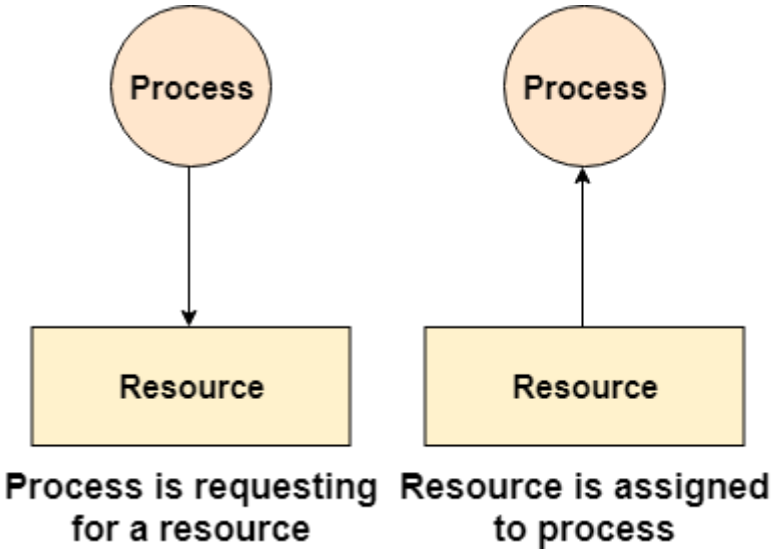
A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.

A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.

A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

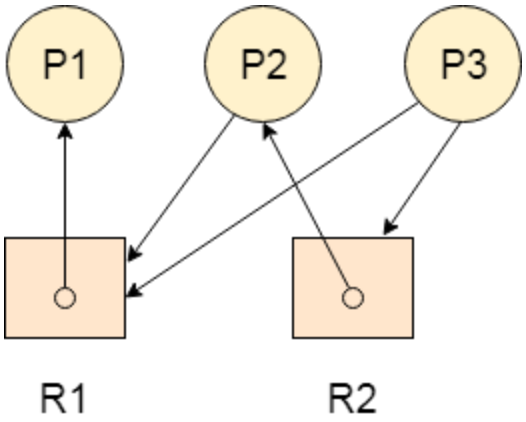


Example

Let'sconsider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

The graph is deadlock free since no cycle is being formed in the graph.

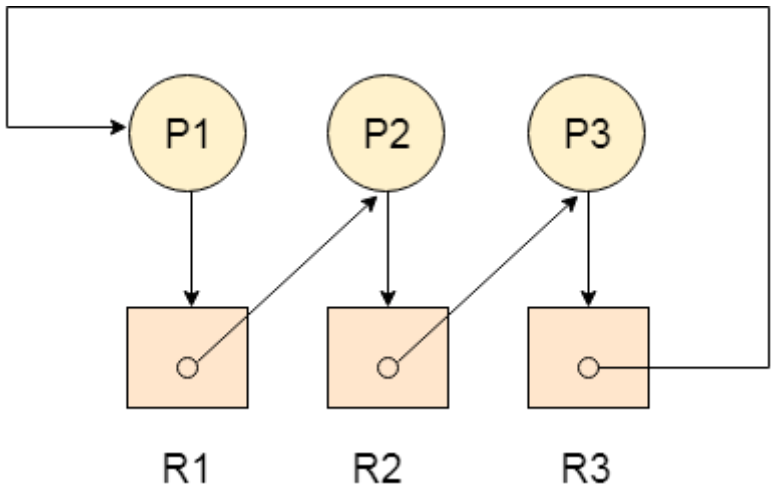


Deadlock Detection using RAG

If a cycle is being formed in a Resource allocation graph where all the resources have the single instance then the system is deadlocked.

In Case of Resource allocation graph with multi-instanced resource types, Cycle is a necessary condition of deadlock but not the sufficient condition.

The following example contains three processes P1, P2, P3 and three resources R2, R2, R3. All the resources are having single instances each.



If we analyze the graph then we can find out that there is a cycle formed in the graph since the system is satisfying all the four conditions of deadlock.

Allocation Matrix

Allocation matrix can be formed by using the Resource allocation graph of a system. In Allocation matrix, an entry will be made for each of the resource assigned. For Example, in the following matrix, en entry is being made in front of P1 and below R3 since R3 is assigned to P1.

Process	R1	R2	R3

P1	0	0	1
P2	1	0	0
P3	0	1	0

Request Matrix

In request matrix, an entry will be made for each of the resource requested. As in the following example, P1 needs R1 therefore an entry is being made in front of P1 and below R1.

Process	R1	R2	R3
P1	1	0	0
P2	0	1	0
P3	0	0	1

Avial = (0,0,0)

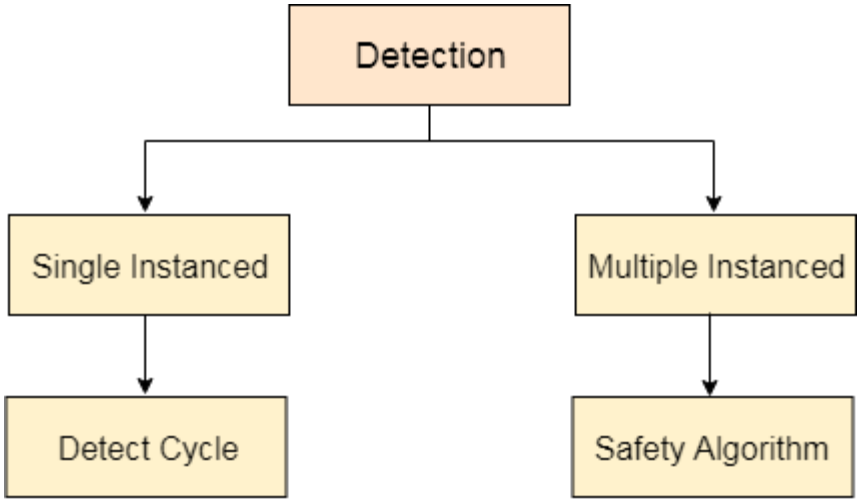
Neither we are having any resource available in the system nor a process going to release. Each of the process needs at least single resource to complete therefore they will continuously be holding each one of them.

We cannot fulfill the demand of at least one process using the available resources therefore the system is deadlocked as determined earlier when we detected a cycle in the graph.

Deadlock Detection and Recovery

In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

In order to recover the system from deadlocks, either OS considers resources or processes.

For Resource

Preempt the resource

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state

System passes through various states to get into the deadlock state. The operating system canrollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

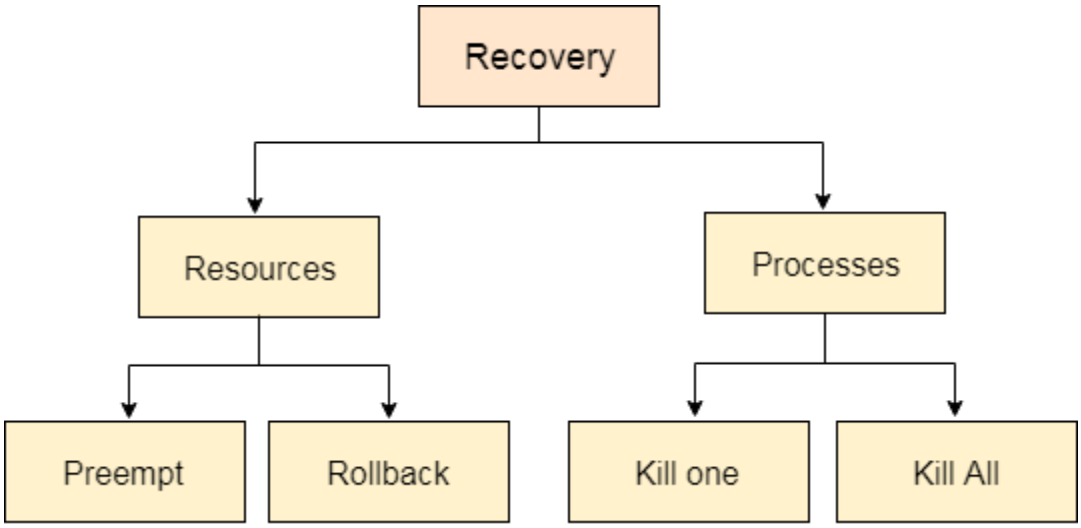
For Process

Kill a process

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.



Recovery from Deadlock in Operating System

In today's world of computer systems and multitasking environments, deadlock is an undesirable situation that can bring operations to a grinding halt. When multiple processes compete for exclusive access to resources and end up in a circular waiting pattern, a deadlock occurs. To maintain the smooth functioning of an operating system, it is crucial to implement recovery mechanisms that can break these deadlocks and restore the system's productivity.

"Recovery from Deadlock in Operating Systems" refers to the set of techniques and algorithms designed to detect, resolve, or mitigate deadlock situations. These methods ensure that the system can continue processing tasks efficiently without being trapped in an eternal standstill. Let's take a closer look at some of the key strategies employed.

There is no mechanism implemented by the OS to avoid or prevent deadlocks. The system, therefore, assumes that a deadlock will undoubtedly occur. The OS periodically checks the system for any deadlocks in an effort to break them. The OS will use various recovery techniques to restore the system if it encounters any deadlocks.

When a Deadlock Detection Algorithm determines that a deadlock has occurred in the system, the system must recover from that deadlock.

Approaches To Breaking a Deadlock

Process Termination

To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:

1. **Abort all the Deadlocked Processes**: Aborting all the processes will certainly break the deadlock but at a great expense. The deadlocked processes may have been computed for a long time, and the result of those partial computations must be discarded and there is a probability of recalculating them later.

2. **Abort one process at a time until the deadlock is eliminated**: Abort one deadlocked process at a time, until the deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because, after aborting each process, we have to run a deadlock detection algorithm to check whether any processes are still deadlocked.

Advantages of Process Termination

- It is a simple method for breaking a deadlock.
- It ensures that the deadlock will be resolved quickly, as all processes involved in the deadlock are terminated simultaneously.
- It frees up resources that were being used by the deadlocked processes, making those resources available for other processes.

Disadvantages of Process Termination

- It can result in the loss of data and other resources that were being used by the terminated processes.
- It may cause further problems in the system if the terminated processes were critical to the system's operation.
- It may result in a waste of resources, as the terminated processes may have already completed a significant amount of work before being terminated.

For Process

1. **Destroy a process**: Although killing a process can solve our problem, choosing which process to kill is more important. The operating system typically terminates a process after it has completed the least amount of work.

2. **End all processes**: Although not suggestible, this strategy can be used if the issue worsens significantly. Because each process will have to start from scratch after being killed, the system will become inefficient.

Resource Preemption

To eliminate deadlocks using resource preemption, we preempt some resources from processes and give those resources to other processes. This method will raise three issues –

1. **Selecting a victim**: We must determine which resources and which processes are to be preempted and also in order to minimize the cost.

2. **Rollback**: We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means aborting the process and restarting it.

3. **Starvation**: In a system, it may happen that the same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided. One solution is that a process must be picked as a victim only a finite number of times.

Advantages of Resource Preemption

1. It can help in breaking a deadlock without terminating any processes, thus preserving data and resources.
2. It is more efficient than process termination as it targets only the resources that are causing the deadlock.
3. It can potentially avoid the need for restarting the system.

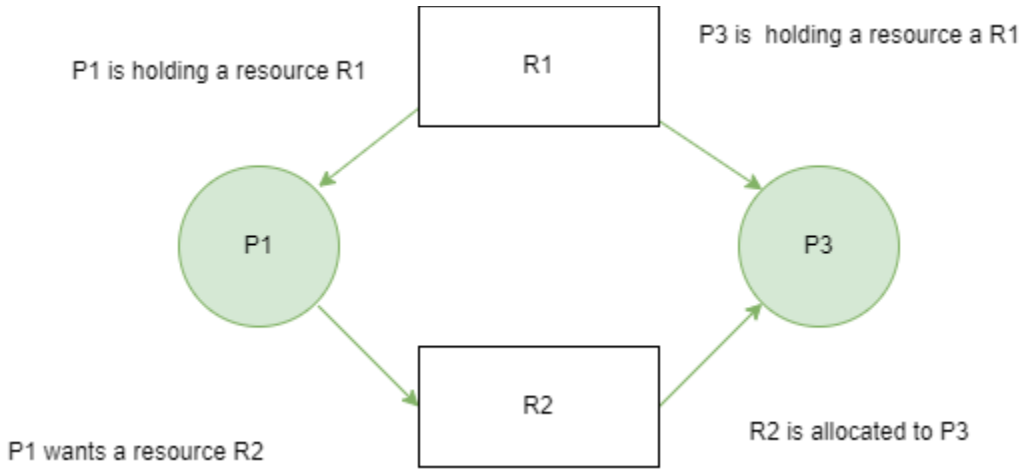
Disadvantages of Resource Preemption

1. It may lead to increased overhead due to the need for determining which resources and processes should be preempted.
2. It may cause further problems if the preempted resources were critical to the system's operation.

3. It may cause delays in the completion of processes if resources are frequently preempted.

Resource Allocation Graph (RAG)

The resource allocation graph (RAG) is a popular technique for computer system deadlock detection. The RAG is a visual representation of the processes holding the resources and their current state of allocation. The resources and processes are represented by the graph's nodes, while their allocation relationships are shown by the graph's edges. A cycle in the graph of the RAG method denotes the presence of a deadlock. When a cycle is discovered, at least one resource needed by another process in the cycle is being held by each process in the cycle, causing a deadlock. The RAG method is a crucial tool in contemporary operating systems due to its high efficiency and ability to spot deadlocks quickly.



Priority Inversion

A technique for breaking deadlocks in real-time systems is called priority inversion. This approach alters the order of the processes to prevent stalemates. A higher priority is given to the process that already has the needed resources, and a lower priority is given to the process that is still awaiting them. The inversion of priorities that can result from this approach can impair system performance and cause performance issues. Additionally, because higher-priority processes may continue to take precedence over lower-priority processes, this approach may starve lower-priority processes of resources.

RollBack

In database systems, rolling back is a common technique for breaking deadlocks. When using this technique, the system reverses the transactions of the involved processes to a time before the deadlock. The system must keep a log of all transactions and the system's condition at various points in time in order to use this method. The transactions can then be rolled back to the initial state and executed again by the system. This approach may result in significant delays in the transactions' execution and data loss.

OR

Handling Deadlocks

- **Deadlock** is a situation where a process or a set of processes is blocked, waiting for some other resource that is held by some other waiting process. It is an undesirable state of the system. The following are the four conditions that must hold simultaneously for a deadlock to occur.
  1. **Mutual Exclusion –** A resource can be used by only one process at a time. If another process requests for that resource then the requesting process must be delayed until the resource has been released.
  2. **Hold and wait –** Some processes must be holding some resources in the non-shareable mode and at the same time must be waiting to acquire some more resources, which are currently held by other processes in the non-shareable mode.
  3. **No pre-emption –** Resources granted to a process can be released back to the system only as a result of voluntary action of that process after the process has completed its task.
  4. **Circular wait –** Deadlocked processes are involved in a circular chain such that each process holds one or more resources being requested by the next process in the chain.

Methods of handling deadlocks: There are four approaches to dealing with deadlocks.

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

These are explained below.

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 pre-emption are'

- If a process that is holding some resource, requests another resource that can not 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 pre-empt 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: The deadlock avoidance Algorithm works by proactively 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

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.