MODULE-4

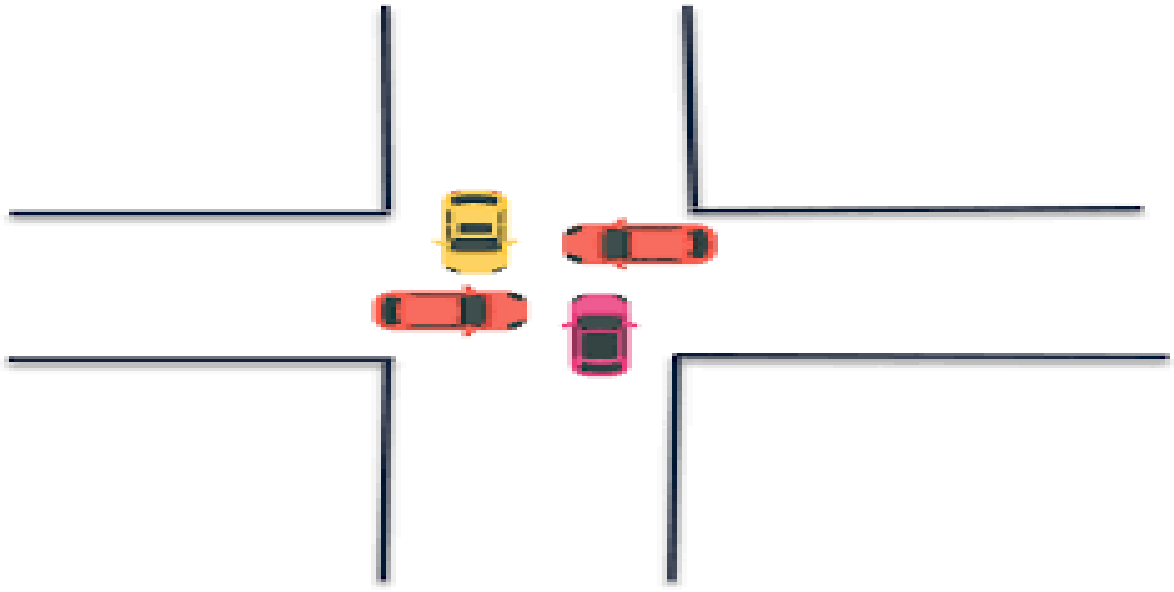
Deadlocks:

A deadlock in OS is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process. The four necessary conditions for a deadlock situation to occur are mutual exclusion, hold and wait, no preemption and circular set. We can prevent a deadlock by preventing any one of these conditions. There are different ways to detect and recover a system from deadlock.

What is Deadlock?

All the processes in a system require some resources such as central processing unit(CPU), file storage, input/output devices, etc to execute it. Once the execution is finished, the process releases the resource it was holding. However, when many processes run on a system they also compete for these resources they require for execution. This may arise a deadlock situation.

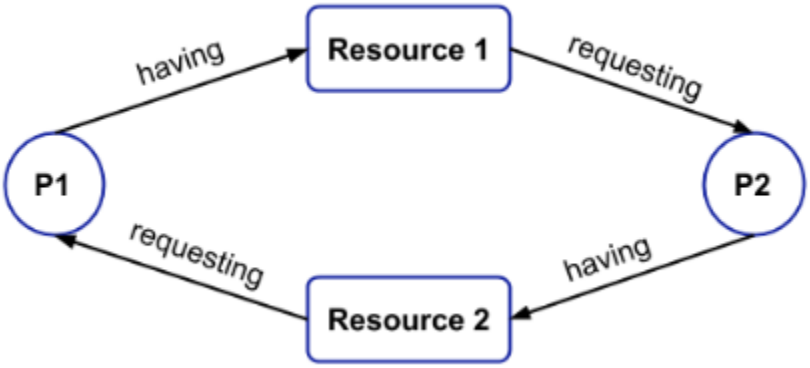
A **deadlock** is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process. Therefore, none of the processes gets executed.



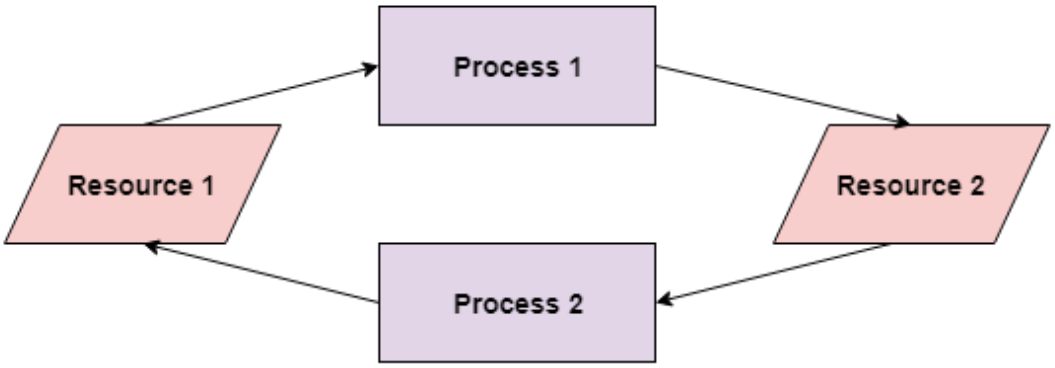
Neccessary Conditions for Deadlock

The four necessary conditions for a deadlock to arise are as follows.

- **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**: The resource can be released by a process voluntarily i.e. after execution of the process.
- **Circular Wait**: A set of processes are waiting for each other in a circular fashion. For example, lets say there are a set of processes {P_0P_0,P_1P_1,P_2P_2,P_3P_3} such that P_0P_0 depends on P_1P_1, P_1P_1 depends on P_2P_2, P_2P_2 depends on P_3P_3 and P_3P_3 depends on P_0P_0. This creates a circular relation between all these processes and they have to wait forever to be executed.



- A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.



Deadlock in Operating System

- In the above diagram, the process 1 has resource 1 and needs to acquire resource 2. Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other's resource to complete their execution but neither of them is willing to relinquish their resources.

Basically in the Normal mode of Operation utilization of resources by a process is in the following sequence:

1. **Request:** Firstly, the process requests the resource. In a case, if the request cannot be granted immediately(e.g: resource is being used by any other process), then the requesting process must wait until it can acquire the resource.
2. **Use:** The Process can operate on the resource (e.g: if the resource is a printer then in that case process can print on the printer).
3. **Release:** The Process releases the resource.

Let us take a look at the differences between starvation and deadlock.

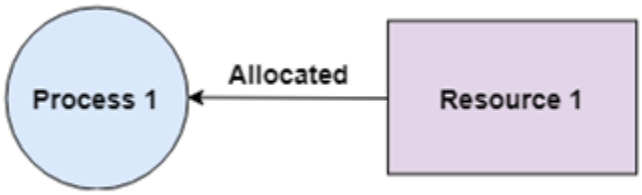
Starvation vs Deadlock

Starvation	Deadlock
When all the low priority processes got blocked, while the high priority processes execute then this situation is termed as Starvation.	Deadlock is a situation that occurs when one of the processes got blocked.
Starvation is a long waiting but it is not an infinite process.	Deadlock is an infinite process.
It is not necessary that every starvation is a deadlock.	There is starvation in every deadlock.
Starvation is due to uncontrolled priority and resource management.	During deadlock, preemption and circular wait does not occur simultaneously.

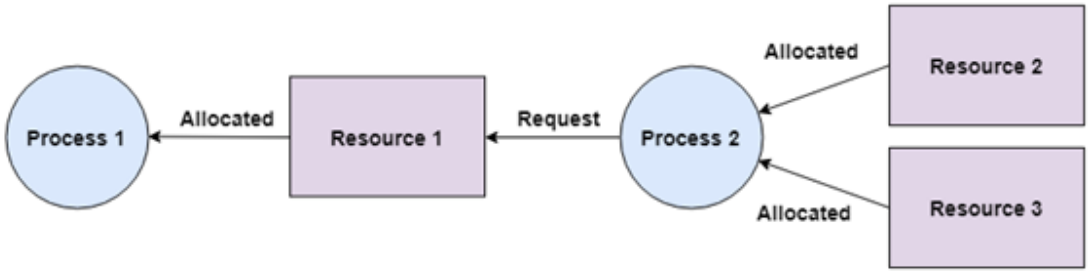
Necessary and sufficient conditions for Deadlock

The Coffman conditions are given as follows −

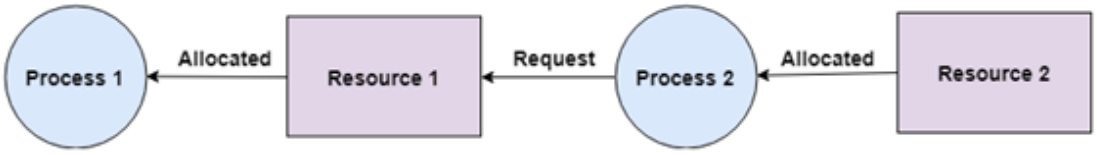
- **Mutual Exclusion**
  There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



- **Hold and Wait**
  A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.

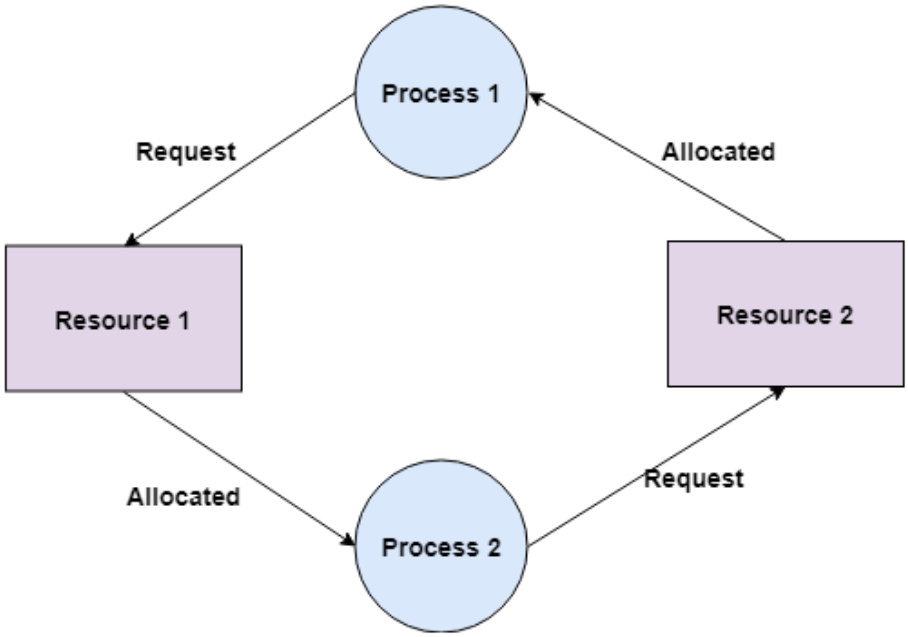


- **No Preemption**
  A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



- **Circular Wait**

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Deadlock Prevention in Operating System

let us take an example of a chair, as we know that chair always stands on its four legs. Likewise, for the deadlock problem, all the above given four conditions are needed. If anyone leg of the chair gets broken, then definitely it will fall. The same is the situation with the deadlock if we become able to violate any condition among the four and do not let them occur together then there can be prevented from the deadlock problem.

We will elaborate deadlock prevention approach by examining each of the four necessary conditions separately.

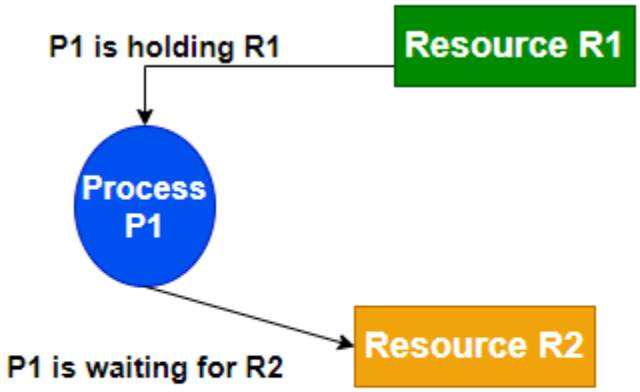
Mutual Exclusion

This condition must hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes. In contrast, Sharable resources do not require mutually exclusive access and thus cannot be involved in a deadlock. A good example of a sharable resource is Read-only files because if several processes attempt to open a read-only file at the same time, then they can be granted simultaneous access to the file.

A process need not to wait for the sharable resource. Generally, deadlocks cannot be prevented by denying the mutual exclusion condition because there are some resources that are intrinsically non-sharable.

Hold and Wait

Hold and wait condition occurs when a process holds a resource and is also waiting for some other resource in order to complete its execution. Thus if we did not want the occurrence of this condition then we must guarantee that when a process requests a resource, it does not hold any other resource.



Hold and wait condition

There are some protocols that can be used in order to ensure that the Hold and Wait condition never occurs:

- According to the first protocol; Each process must request and gets all its resources before the begining of its execution.

- The second protocol allows a process to request resources only when it does not occupy any resource.

Let us illustrate the difference between these two protocols:

We will consider a process that mainly copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer. If all the resources must be requested at the beginning of the process according to the first protocol, then the process requests the DVD drive, disk file, and printer initially. It will hold the printer during its entire execution, even though the printer is needed only at the end.

While the second method allows the process to request initially only the DVD drive and disk file. It copies the data from the DVD drive to the disk and then releases both the DVD drive and the disk file. The process must then again request the disk file and printer. After copying the disk file to the printer, the process releases these two resources as well and then terminates.

Disadvantages of Both Protocols

- Utilization of resources may be low, since resources may be allocated but unused for a long period. In the above-given example, for instance, we can release the DVD drive and disk file and again request the disk file and printer only if we can be sure that our data will remain on the disk file. Otherwise, we must request all the resources at the beginning of both protocols.
- There is a possibility of starvation. A process that needs several popular resources may have to wait indefinitely because at least one of the resources that it needs is always allocated to some other process.

No Preemption

The third necessary condition for deadlocks is that there should be no preemption of resources that have already been allocated. In order to ensure that this condition does not hold the following protocols can be used :

- According to the First Protocol: "If a process that is already holding some resources requests another resource and if the requested resources cannot be allocated to it, then it must release all the resources currently allocated to it."
- According to the Second Protocol: "When a process requests some resources, if they are available, then allocate them. If in case the requested resource is not available then we will check whether it is being used or is allocated to some other

process waiting for other resources. If that resource is not being used, then the operating system preempts it from the waiting process and allocate it to the requesting process. And if that resource is being used, then the requesting process must wait".

The second protocol can be applied to those resources whose state can be easily saved and restored later for example CPU registers and memory space, and cannot be applied to resources like printers and tape drivers.

Circular Wait

The Fourth necessary condition to cause deadlock is circular wait, In order to ensure violate this condition we can do the following:

Assign a priority number to each resource. There will be a condition that any process cannot request for a lesser priority resource. This method ensures that not a single process can request a resource that is being utilized by any other process and due to which no cycle will be formed.

Example: Assume that R5 resource is allocated to P1, if next time P1 asks for R4, R3 that are lesser than R5; then such request will not be granted. Only the request for resources that are more than R5 will be granted.

S.No	Necessary Conditions	Approach	Practical Implementation
1	Mutual Exclusion	The approach used to violate this condition is spooling.	Not possible
2	Hold and wait	In order to violate this condition, the approach is to request all the resources for a process initially	Not possible

S.No	Necessary Conditions	Approach	Practical Implementation
3	No Preemption	In order to violate this condition, the approach is: snatch all the resources from the process.	Not possible
4	Circular Wait	In this approach is to assign priority to each resource and order them numerically	**possible**

Deadlock avoidance

A deadlock can be avoided using the Bankers' Algorithm.

Bankers' Algorithm:

The **Bankers' Algorithm** is a resource allocation and deadlock avoidance algorithm that examines all resource requests made by systems. It checks for the safe state and makes the request if the system remains in the safe state after request approval. If there is no safe state, the request is denied.

Inputs required for the Bankers' Algorithm:

1. Maximum need or resources required by each process
2. The resources currently allocated by each process
3. Free resources available in the system

Request for the resource will only be granted:

1. If the request made by the process is </= the freely available resource in the system

2. If the request made by the process is </= maximum amount of resources required for the process

Inputs to Banker's Algorithm:

1. Max need of resources by each process.
2. Currently, allocated resources by each process.
3. Max free available resources in the system.

The request will only be granted under the below condition:

1. If the request made by the process is less than equal to max need to that process.
2. If the request made by the process is less than equal to the freely available resource in the system.

Example:

```
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

P1  1 2 2 1

P2  1 0 3 3

P3  1 2 1 0


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 - 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
```

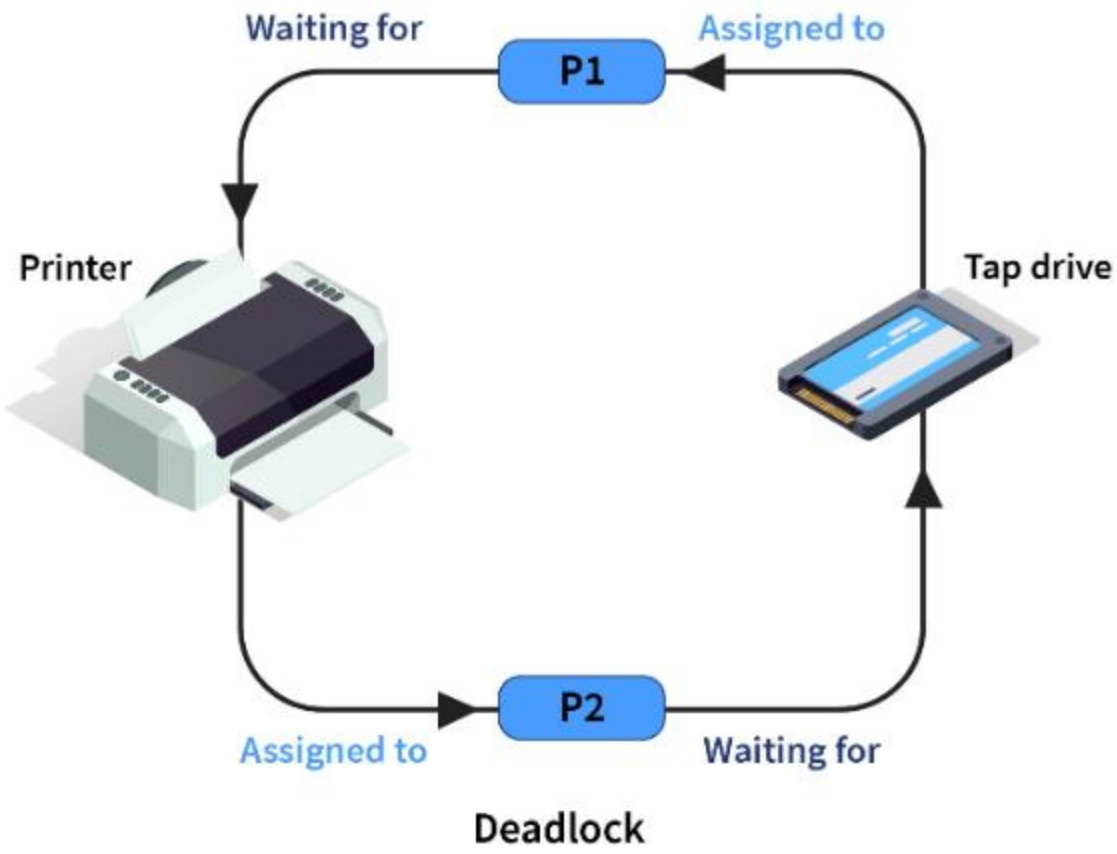
Note: Deadlock prevention is more strict than Deadlock Avoidance.

Deadlock Avoidance

Deadlock Avoidance is a process used by the Operating System to avoid Deadlock. Let's first understand what is Deadlock in Operating System. Deadlock is a situation that occurs in Operating System when any Process enters a waiting state because another waiting process is holding the demanded resource. Deadlock is a common problem in multi-processing where several processes share a specific type of mutually exclusive resource known as a soft lock or software.

But how can Operating System avoid Deadlock?

Operating System avoids Deadlock by knowing the maximum resources requirements of the processes initially, and also Operating System knows the free resources available at that time. Operating System tries to allocate the resources according to the process requirements and checks if the allocation can lead to a safe state or an unsafe state. If the resource allocation leads to an unsafe state, then Operating System does not proceed further with the allocation sequence.



Let's understand the working of Deadlock Avoidance with the help of an intuitive example.

Process	Maximum Required	current Available	Need
P1	9	5	4
P2	5	2	3
P3	3	1	2

Let's consider three processes P1, P2, P3. Some more information on which the processes tells the Operating System are :

- P1 process needs a maximum of 9 resources (Resources can be any software or hardware Resource like tape drive or printer etc..) to complete its execution. P1 is currently allocated with 5 Resources and needs 4 more to complete its execution.
- P2 process needs a maximum of 5 resources and is currently allocated with 2 resources. So it needs 3 more resources to complete its execution.
- P3 process needs a maximum of 3 resources and is currently allocated with 1 resource. So it needs 2 more resources to complete its execution.

- Operating System knows that only 2 resources out of the total available resources are currently free.

But only 2 resources are free now. Can P1, P2, and P3 satisfy their requirements? Let's try to find out.

As only 2 resources are free for now, then only P3 can satisfy its need for 2 resources. If P3 takes 2 resources and completes its execution, then P3 can release its 3 (1+2) resources. Now the three free resources which P3 released can satisfy the need of P2. Now, P2 after taking the three free resources, can complete its execution and then release 5 (2+3) resources. Now five resources are free. P1 can now take 4 out of the 5 free resources and complete its execution. So, with 2 free resources available initially, all the processes were able to complete their execution leading to Safe State. The order of execution of the processes was <P3, P2, P1>.

What if initially there was only 1 free resource available? None of the processes would be able to complete its execution. Thus leading to an unsafe state.

We use two words, safe and unsafe states. What are those states? Let's understand these concepts.

Safe State and Unsafe State

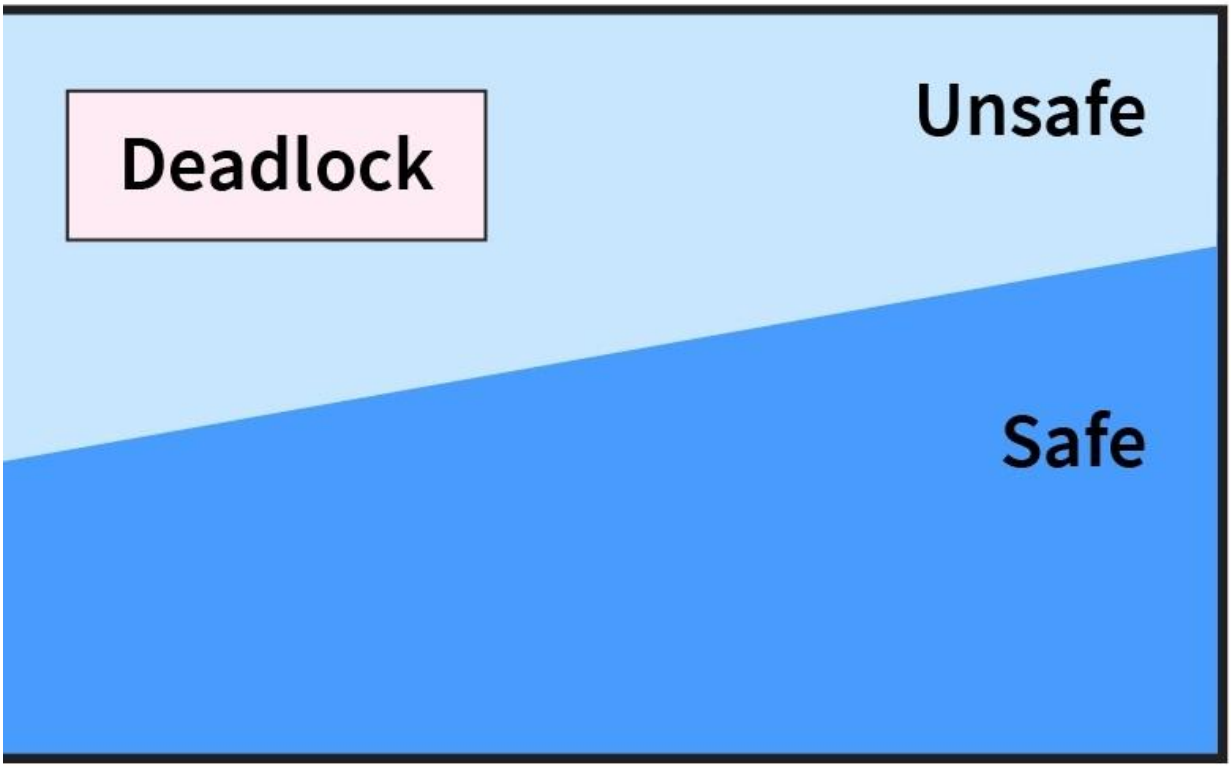
Safe State - In the above example, we saw that Operating System was able to satisfy the need of all three processes, P1, P2, and P3, with their resource requirements. So all the processes were able to complete their execution in a certain order like P3->P2->P1.

So, **If Operating System is able to allocate or satisfy the maximum resource requirements of all the processes in any order then the system is said to be in Safe State.**

So safe state does not lead to Deadlock.

Unsafe State - If Operating System is not able to prevent Processes from requesting resources which can also lead to Deadlock, then the System is said to be in an Unsafe State.

Unsafe State does not necessarily cause deadlock it may or may not causes deadlock.



So, in the above diagram shows the three states of the System. An unsafe state does not always cause Deadlock. Some unsafe states can lead to Deadlock, as shown in the diagram.

Deadlock Avoidance Example

Let's take an example that has multiple resources requirement for every Process. Let there be three Processes P1, P2, P3, and 4 resources R1, R2, R3, R4. The maximum resources requirements of the Processes are shown in the below table.

Process	R1	R2	R3	R4
P1	3	2	3	2
P2	2	3	1	4
P3	3	1	5	0

A number of currently allocated resources to the processes are:

Process	R1	R2	R3	R4
P1	1	2	3	1
P2	2	1	0	2
P3	2	0	1	0

The total number of resources in the System are :

R1	R2	R3	R4
7	4	5	4

We can find out the no of available resources for each of P1, P2, P3, P4 by subtracting the currently allocated resources from total resources.

Available Resources are :

R1	R2	R3	R4
2	1	1	1

Now, The need for the resources for the processes can be calculated by :

Need = Maximum Resources Requirement - Currently Allocated Resources.

The need for the Resources is shown below:

Process	R1	R2	R3	R4
P1	2	1	0	1
P2	0	2	1	2
P3	1	1	4	0

The available free resources are <2,1,1,1> of resources of R1, R2, R3, and R4 respectively, which can be used to satisfy only the requirements of process P1 only initially as process P2 requires 2 R2 resources which are not available. The same is the case with Process P3, which requires 4 R3 resources which is not available initially.

The Steps for resources allotment is explained below:

1. Firstly, Process P1 will take the available resources and satisfy its resource need, complete its execution and then release all its allocated resources. Process P1 is initially allocated <1,2,3,1> resources of R1, R2, R3, and R4 respectively. Process P1 needs <2,1,0,1> resources of R1, R2, R3 and R4 respectively to complete its execution. So, process P1 takes the available free resources <2,1,1,1> resources of R1, R2, R3, R4 respectively and can complete its execution and then release its current allocated resources and also the free resources it used to complete its execution. Thus P1 releases <1+2,2+1,3+1,1+1> = <3,3,4,2> resources of R1, R2, R3, and R4 respectively.
2. After step 1 now, available resources are now <3,3,4,2>, which can satisfy the need of Process P2 as well as process P3. After process P2 uses the available Resources and completes its execution, the available resources are now <5,4,4,4>.
3. Now, the available resources are <5,4,4,4>, and the only Process left for execution is Process P3, which requires <1,1,4,0> resources each of R1, R2, R3, and R4. So it can easily use the available resources and complete its execution. After P3 is executed, the

resources available are <7,4,5,4>, which is equal to the maximum resources or total resources available in the System.

So, **the process execution sequence in the above example was <P1, P2, P3>**. But it could also have been <P1, P3, P2> if process P3 would have been executed before process P2, which was possible as there were sufficient resources available to satisfy the need of both Process P2 and P3 after step 1 above.

Deadlock Avoidance Solution

Deadlock Avoidance can be solved by two different algorithms:

- **Resource allocation Graph**
- **Banker's Algorithm**

We will discuss both algorithms in detail in their separate article.

Resource Allocation Graph

Resource Allocation Graph (RAG) is used to represent the state of the System in the form of a Graph. The Graph contains all processes and resources which are allocated to them and also the requesting resources of every Process. Sometimes if the number of processes is less, We can easily identify a deadlock in the System just by observing the Graph, which can not be done easily by using tables that we use in Banker's algorithm.

Deadlock Detection

If a system does not employ either a deadlock prevention or deadlock avoidance algorithm then a deadlock situation may occur. In this case-

- Apply an algorithm to examine state of system to determine whether deadlock has occurred or not.
- Apply an algorithm to recover from the deadlock. For more refer- Deadlock Recovery

Deadlock Avoidance Algorithm/ Bankers Algorithm:

The algorithm employs several times varying data structures:

- **Available –**
  A vector of length m indicates the number of available resources of each type.
- **Allocation –**
  An n*m matrix defines the number of resources of each type currently

allocated to a process. Column represents resource and rows represent process.

- **Request –**
  An n*m matrix indicates the current request of each process. If request[i][j] equals k then process P_i is requesting k more instances of resource type R_j.

This algorithm has already been discussed here

Now, Bankers algorithm includes a **Safety Algorithm / Deadlock Detection Algorithm**
The algorithm for finding out whether a system is in a safe state can be described as follows:
Steps of Algorithm:

1. Let *Work* and *Finish* be vectors of length m and n respectively.
   Initialize *Work= Available*. For *i=0, 1, …., n-1*, if $Request_i$ = 0, then *Finish[i]* = true; otherwise, *Finish[i]*= false.
2. Find an index i such that both
   a) *Finish[i] == false*
   b) $Request_i$ <= *Work*
   If no such *i* exists go to step 4.
3. *Work= Work+* $Allocation_i$
   Finish[i]= true
   Go to Step 2.
4. If *Finish[i]== false* for some i, 0<=i<n, then the system is in a deadlocked state. Moreover, if *Finish[i]==false* the process P_i is deadlocked.

For example,

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

1. In this, Work = [0, 0, 0] &
   Finish = [false, false, false, false, false]

2. *i=0* is selected as both Finish[0] = false and [0, 0, 0]<=[0, 0, 0].

3. Work =[0, 0, 0]+[0, 1, 0] =>[0, 1, 0] &
   Finish = [true, false, false, false, false].

4. *i=2* is selected as both Finish[2] = false and [0, 0, 0]<=[0, 1, 0].
5. Work =[0, 1, 0]+[3, 0, 3] =>[3, 1, 3] &
   Finish = [true, false, true, false, false].

6. *i=1* is selected as both Finish[1] = false and [2, 0, 2]<=[3, 1, 3].
7. Work =[3, 1, 3]+[2, 0, 0] =>[5, 1, 3] &
   Finish = [true, true, true, false, false].

8. *i=3* is selected as both Finish[3] = false and [1, 0, 0]<=[5, 1, 3].
9. Work =[5, 1, 3]+[2, 1, 1] =>[7, 2, 4] &
   Finish = [true, true, true, true, false].

10. *i=4* is selected as both Finish[4] = false and [0, 0, 2]<=[7, 2, 4].

11. Work =[7, 2, 4]+[0, 0, 2] =>[7, 2, 6] &
    Finish = [true, true, true, true, true].

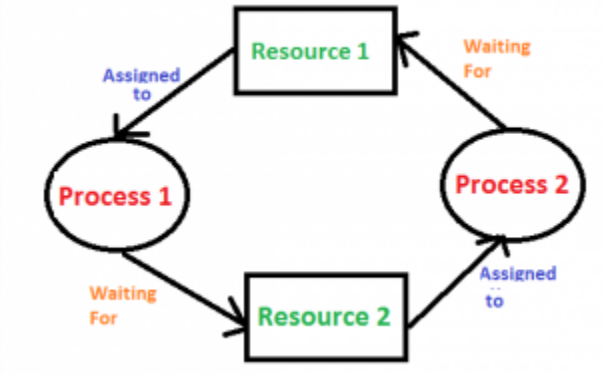
12. Since Finish is a vector of all true it means **there is no deadlock** in this example.

Deadlock Detection And Recovery

In the previous post, we have discussed Deadlock Prevention and Avoidance. In this post, Deadlock Detection and Recovery technique to handle deadlock is discussed.

Deadlock Detection

1. If resources have single instance:
   In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient condition for deadlock.



2. In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1 → P1 → R2 → P2. So, Deadlock is Confirmed.

3. If there are multiple instances of resources:
   Detection of the cycle is necessary but not sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

Deadlock Recovery

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is time and space consuming process. Real-time operating systems use Deadlock recovery.

Recovery method

1. **Killing the process:** killing all the process involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till system recover from deadlock.
2. **Resource Preemption:** Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.