Ms. Vandana Sharma

BCA 306

UNIT-3

DEADLOCK

SYLLABUS: System model, Characterization, Deadlock Prevention, Deadlock Avoidance, and Detection, Recovery from deadlock.

Every process needs some resources to complete its execution. However, the resource is granted in asequential order.

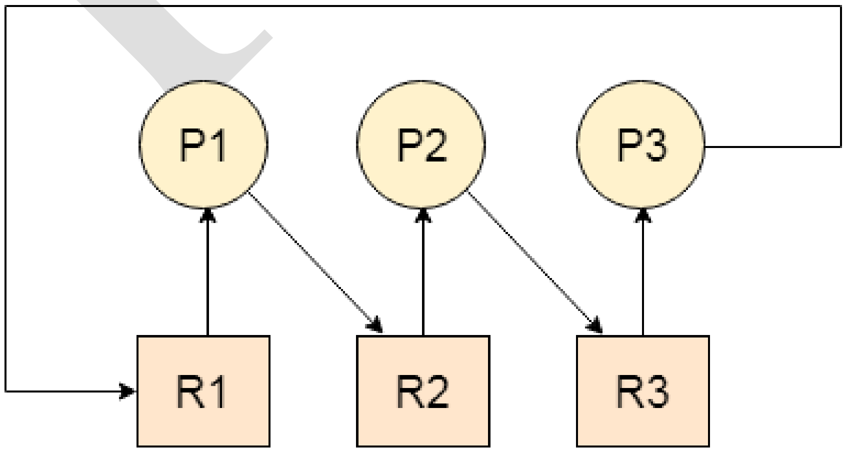
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R2 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



System Model:-

- For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources, which can be partitioned into different categories, to be allocated to a number of processes, each having different needs.
- Resource categories may include memory, printers, CPUs, open files, tape drives, CD-ROMS, etc. By
- definition, all the resources within a category are equivalent, and a request of this category canbe equally satisfied by any one of the resources in that category. If this is not the case (i.e. if thereis some difference between the resources within a category), then that category needs to be furtherdivided into separate categories. For example, "printers" may need to be separated into "laserprinters" and "color inkjet printers".
- Some categories may have a single resource.
- In normal operation a process must request a resource before using it, and release it when it is done,in the following sequence:
    1. **Request -** If the request cannot be immediately granted, then the process must wait until the resource(s) it needs become available. For example the system calls open( ), malloc( ), new( ), and request( ).
    2. **Use -** The process uses the resource, e.g. prints to the printer or reads from the file.
    3. **Release -** The process relinquishes the resource. so that it becomes available for other processes. For example, close( ), free( ), delete( ), and release( ).
- For all kernel-managed resources, the kernel keeps track of what resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available. Application-managed resources can be controlled using mutexes or wait( ) and signal( ) calls, (i.e. binary or counting semaphores.)
- A set of processes is deadlocked when every process in the set is waiting for a resource that is currently allocated to another process in the set (and which can only be released when that other waiting process makes progress.)

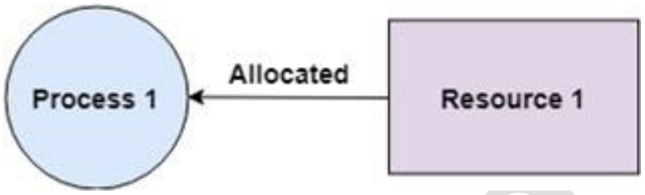
Deadlock Characterization:-

Necessary conditions for Deadlocks

1. Mutual Exclusion

A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the sameresource at the same time.

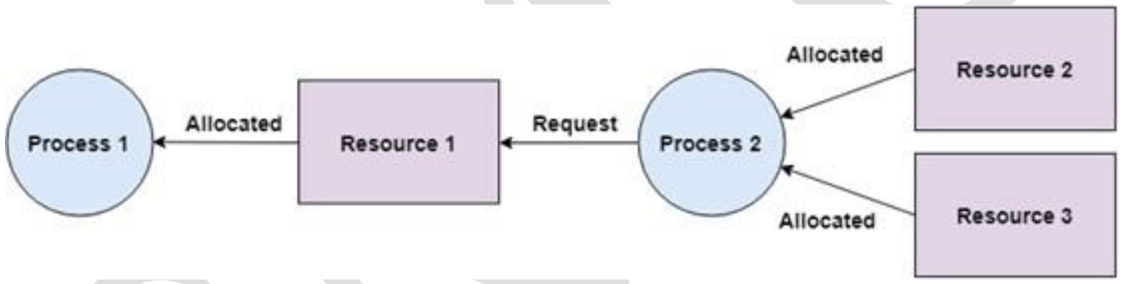
There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



2. Hold and Wait

A process waits for some resources while holding another resource at the same time.

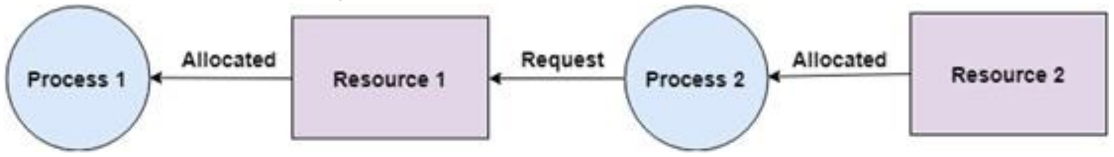
A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



3. No preemption

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

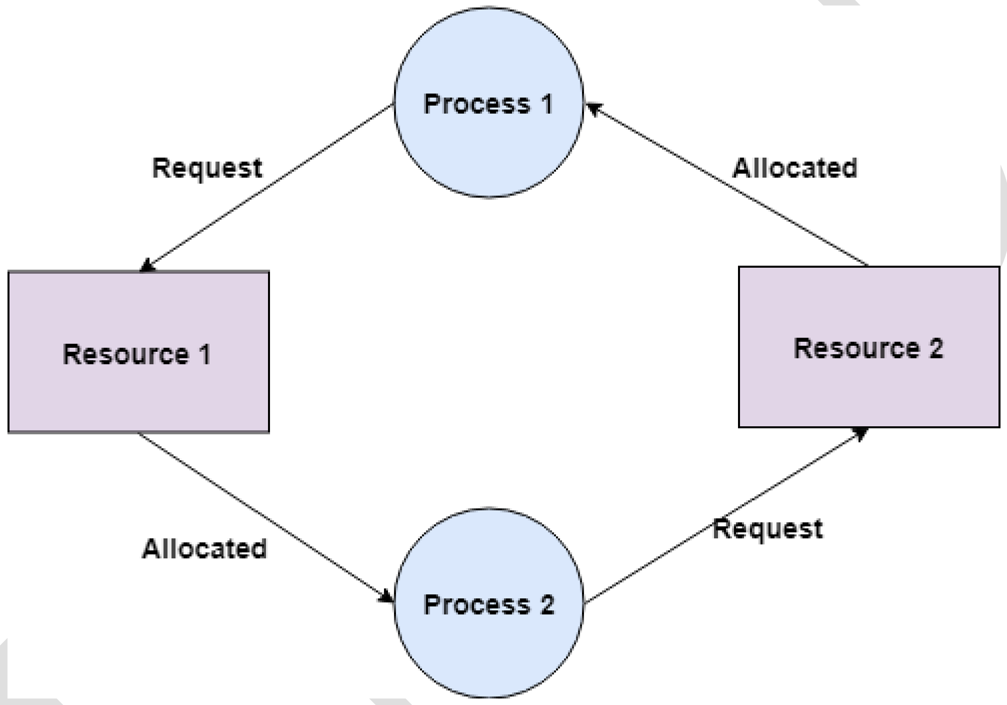
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



4. Circular Wait

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

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.



Methods for Handling Deadlocks

- Generally speaking there are three ways of handling deadlocks:

1. **Deadlock prevention or avoidance -** Do not allow the system to get into a deadlocked state. In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future. (Ranging from a simple worst-case maximum to a complete resource request and release plan for each process, depending on the particular algorithm.)

2. **Deadlock detection and recovery -** Abort a process or preempt some resources when deadlocks are detected. Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative.
3. **Ignore the problem all together -** If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take. If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes. Unfortunately this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.

1. 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.

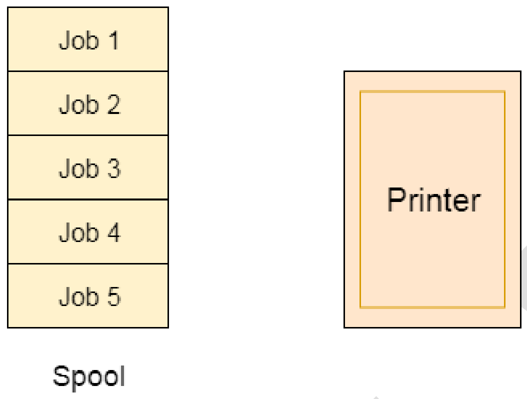
However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

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

Mutual Exclusion

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 thatspool.

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 resourcefor a very long time.

3. No Preemption

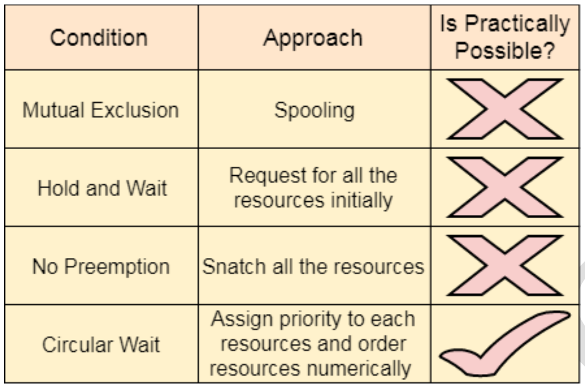
Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

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 fora 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:-

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

E = (7 6 8 4)

P = (6 2 8 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 assignedto each process.

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

Vector P represents the instances of resources that have been assigned to processes. Vector A represents thenumber 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.

Banker's Algorithm:-

Banker's algorithm is a **deadlock avoidance algorithm**. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Everytime a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Banker's algorithm works in a similar way in computers.

Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.

Let us assume that there are n processes and m resource types. Some data structures that are used to implement the banker's algorithm are:

1. Available

It is an **array** of length m. It represents the number of available resources of each type. If Available[j] = k,then there are k instances available, of resource type R(j).

2. Max

It is an n x m matrix which represents the maximum number of instances of each resource that a processcan request. If Max[i][j] = k, then the process P(i) can request atmost k instances of resource type R(j).

3. Allocation

It is an n x m matrix which represents the number of resources of each type currently allocated to eachprocess. If Allocation[i][j] = k, then process P(i) is currently allocated k instances of resource type R(j).

4. Need

It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, thenprocess P(i) may need k more instances of resource type R(j) to complete its task.

Need[i][j] = Max[i][j] - Allocation [i][j]

Safety Algorithm:-

This algo used for finding out whether a system in a safe state or not.

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both
a) Finish[i] = false
b) Needi <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)

4) if Finish [i] = true for all i
then the system is in a safe state

Resource-Request Algorithm

Let Request$_i$ be the request array for process P$_i$. Request$_i$ [j] = k means process P$_i$ wants k instances ofresource type R$_j$. When a request for resources is made by process P$_i$, the following actions are taken:

1) If Request$_i$ <= Need$_i$

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If Request$_i$ <= Available

Goto step (3); otherwise, P$_i$ must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

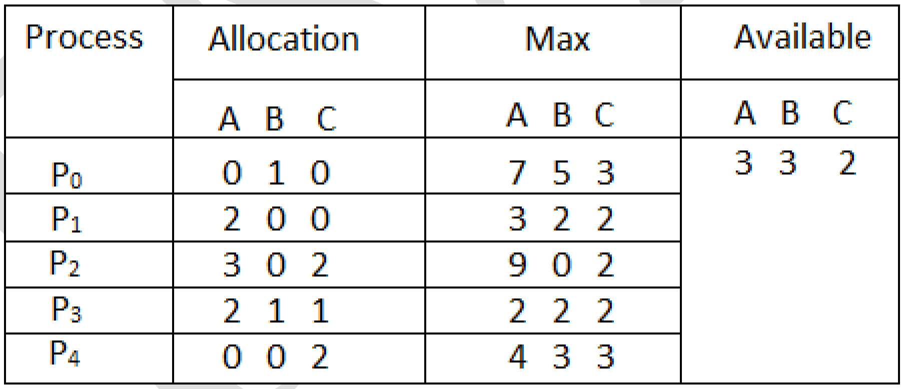
Available = Available – Requesti

Allocation$_i$ = Allocation$_i$ + Request$_i$

Need$_i$ = Need$_i$– Request$_i$

Example:

Considering a system with five processes P$_0$ through P$_4$ and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t$_0$ following snapshot of the system has been taken:



Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P$_0$	0	1	0	7	5	3	3	3	2
P$_1$	2	0	0	3	2	2			
P$_2$	3	0	2	9	0	2			
P$_3$	2	1	1	2	2	2			
P$_4$	0	0	2	4	3	3			

Question1. What will be the content of the Need matrix?

Need [i, j] = Max [i, j] – Allocation [i, j]For

P$_0$= (7,5,3) - (0,1,0)

Need for P$_0$ = (7,4,3) For

P$_1$= (3,2,2) - (2,0,0)

Need for P$_1$ = (1,2,2)

For P2= (9,0,2) - (3,0,2)

Need for P2 = (6,0,0) For

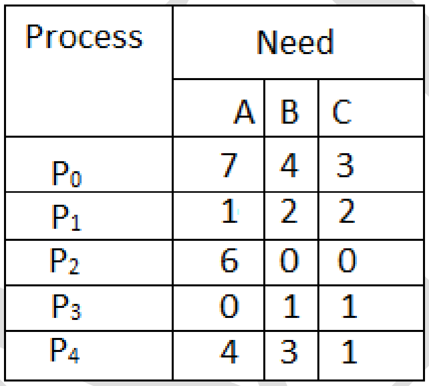
P3= (2,2,2) - (2,1,1)

Need for P3 = (0,1,1) For

P4= (4,3,3) - (0,0,2)

Need for P4 = (4,3,1)

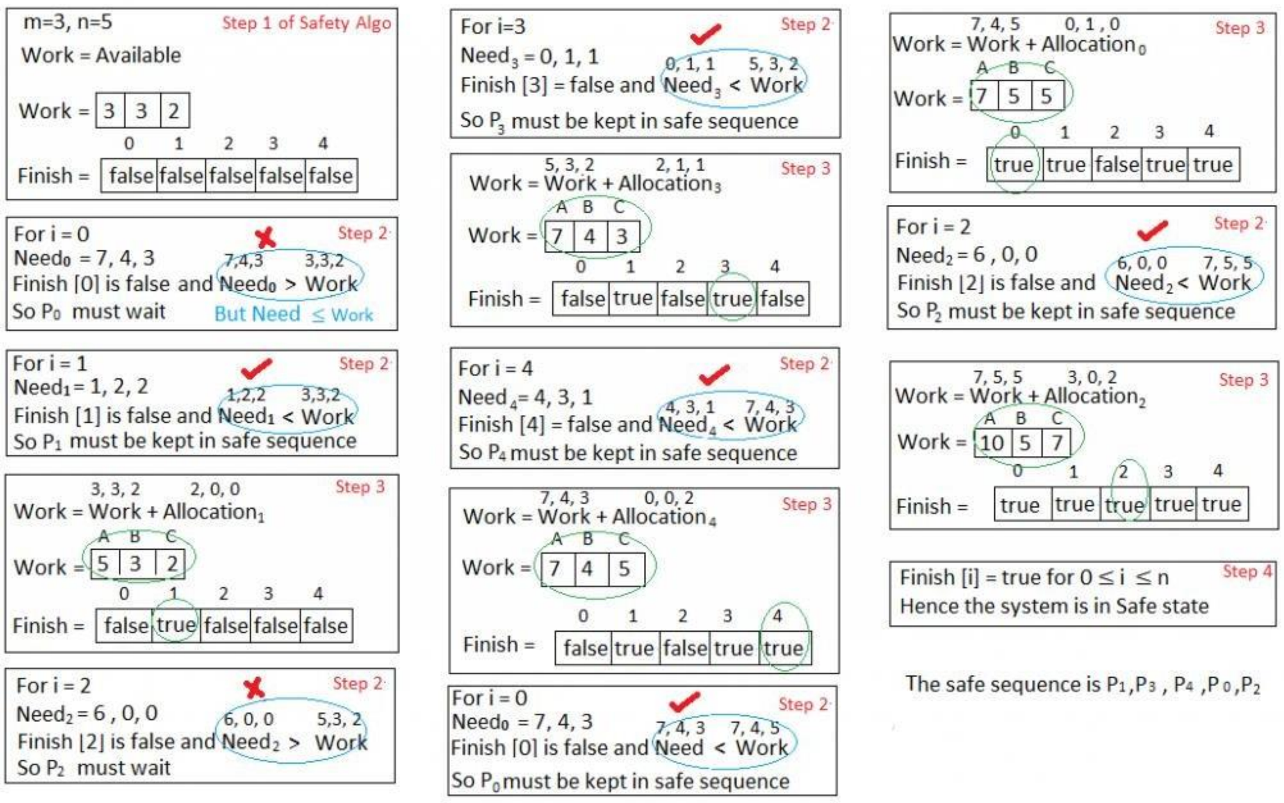
So, the content of Need Matrix is:



Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,



m=3, n=5 — Step 1 of Safety Algo

Work = Available

Work = 3 | 3 | 2

0	1	2	3	4
false	false	false	false	false

Finish = false false false false false

For i = 0 ✗ — Step 2

$Need_0$ = 7, 4, 3

Finish [0] is false and $Need_0$ > Work ($\overset{7,4,3}{Need_0} > \overset{3,3,2}{Work}$)

So P_0 must wait — But Need ≤ Work

For i = 1 ✓ — Step 2

$Need_1$ = 1, 2, 2

Finish [1] is false and $\overset{1,2,2}{Need_1} < \overset{3,3,2}{Work}$

So P_1 must be kept in safe sequence

Step 3

Work = Work + $Allocation_1$ ($\overset{3,3,2}{Work} + \overset{2,0,0}{Allocation_1}$)

Work = 5 | 3 | 2 (A B C)

Finish =

0	1	2	3	4
false	true	false	false	false

For i = 2 ✗ — Step 2

$Need_2$ = 6 , 0, 0

Finish [2] is false and $\overset{6,0,0}{Need_2} > \overset{5,3,2}{Work}$

So P_2 must wait

For i=3 ✓ — Step 2

$Need_3$ = 0, 1, 1

Finish [3] = false and $\overset{0,1,1}{Need_3} < \overset{5,3,2}{Work}$

So P_3 must be kept in safe sequence

Step 3

Work = Work + $Allocation_3$ ($\overset{5,3,2}{Work} + \overset{2,1,1}{Allocation_3}$)

Work = 7 | 4 | 3 (A B C)

Finish =

0	1	2	3	4
false	true	false	true	false

For i = 4 ✓ — Step 2

$Need_4$ = 4, 3, 1

Finish [4] = false and $\overset{4,3,1}{Need_4} < \overset{7,4,3}{Work}$

So P_4 must be kept in safe sequence

Step 3

Work = Work + $Allocation_4$ ($\overset{7,4,3}{Work} + \overset{0,0,2}{Allocation_4}$)

Work = 7 | 4 | 5 (A B C)

Finish =

0	1	2	3	4
false	true	false	true	true

For i = 0 ✓ — Step 2

$Need_0$ = 7, 4, 3

Finish [0] is false and $\overset{7,4,3}{Need} < \overset{7,4,5}{Work}$

So P_0 must be kept in safe sequence

Step 3

Work = Work + $Allocation_0$ ($\overset{7,4,5}{Work} + \overset{0,1,0}{Allocation_0}$)

Work = 7 | 5 | 5 (A B C)

Finish =

0	1	2	3	4
true	true	false	true	true

For i = 2 ✓ — Step 2

$Need_2$ = 6 , 0, 0

Finish [2] is false and $\overset{6,0,0}{Need_2} < \overset{7,5,5}{Work}$

So P_2 must be kept in safe sequence

Step 3

Work = Work + $Allocation_2$ ($\overset{7,5,5}{Work} + \overset{3,0,2}{Allocation_2}$)

Work = 10 | 5 | 7 (A B C)

Finish =

0	1	2	3	4
true	true	true	true	true

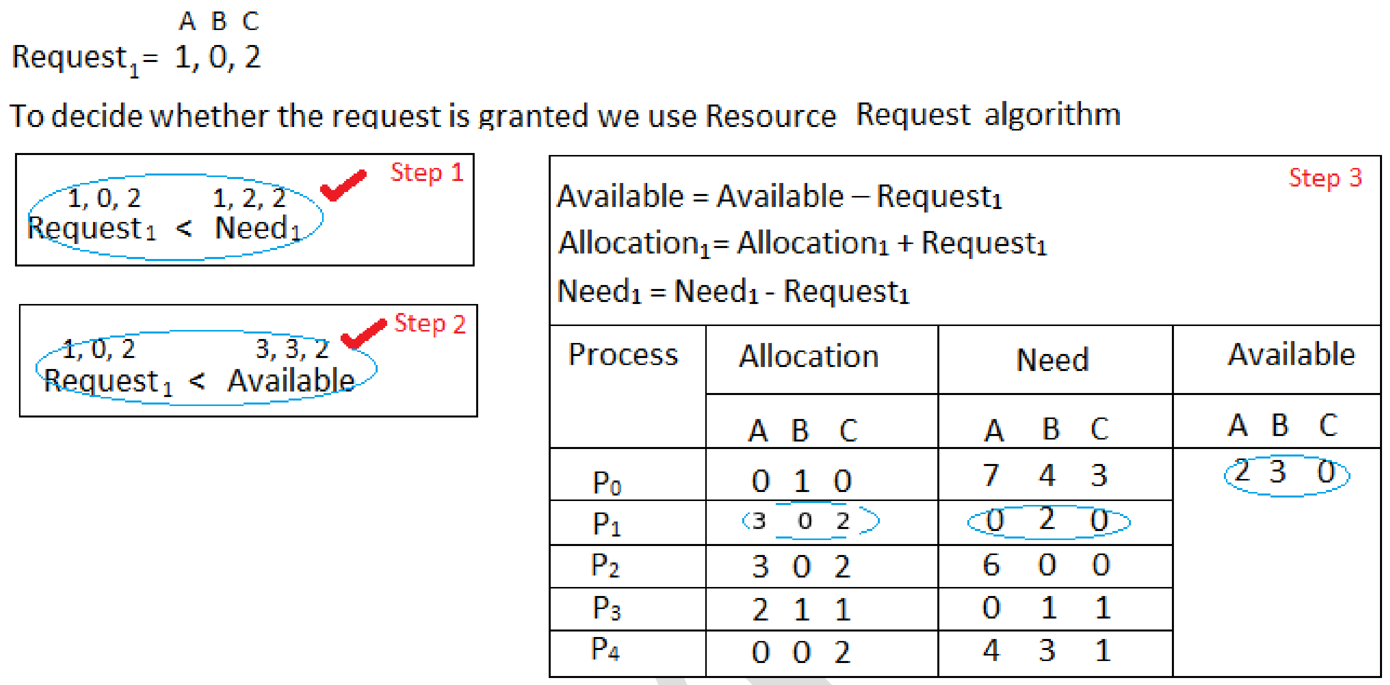
Step 4

Finish [i] = true for 0 ≤ i ≤ n

Hence the system is in Safe state

The safe sequence is P_1,P_3 , P_4 ,P_0,P_2

Question3. What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C?



$$\begin{array}{c} A\ B\ C \\ \text{Request}_1 = 1, 0, 2 \end{array}$$

To decide whether the request is granted we use Resource Request algorithm

Step 1

$\overset{1,\,0,\,2}{\text{Request}_1} < \overset{1,\,2,\,2}{\text{Need}_1}$ ✓

Step 2

$\overset{1,\,0,\,2}{\text{Request}_1} < \overset{3,\,3,\,2}{\text{Available}}$ ✓

Step 3

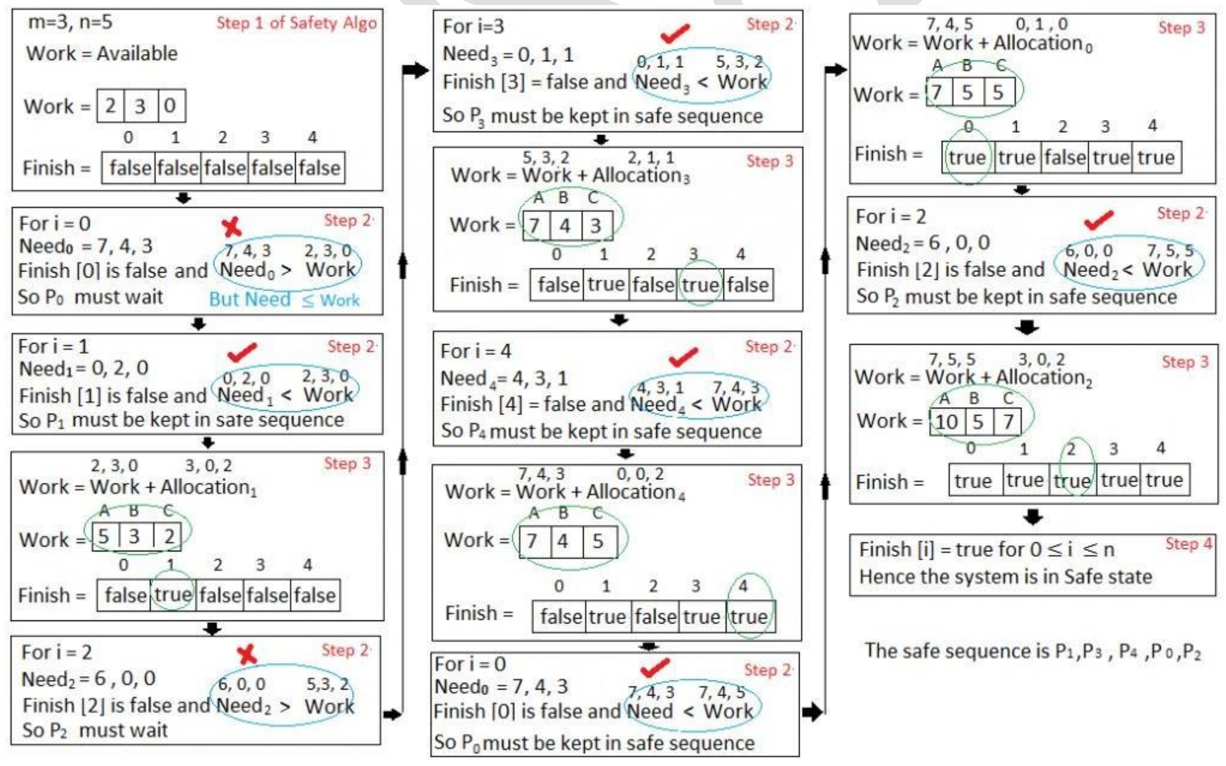
Available = Available – Request₁

Allocation₁ = Allocation₁ + Request₁

Need₁ = Need₁ - Request₁

Process	Allocation	Need	Available
	A B C	A B C	A B C
P₀	0 1 0	7 4 3	2 3 0
P₁	3 0 2	0 2 0	
P₂	3 0 2	6 0 0	
P₃	2 1 1	0 1 1	
P₄	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we again execute Safety algorithm onthe above data structures.



Step 1 of Safety Algo

m=3, n=5

Work = Available

Work = | 2 | 3 | 0 |

	0	1	2	3	4
Finish =	false	false	false	false	false

Step 2

For i = 0

Need₀ = 7, 4, 3

Finish [0] is false and $\overset{7,\,4,\,3}{\text{Need}_0} > \overset{2,\,3,\,0}{\text{Work}}$ ✗

So P₀ must wait

But Need ≤ Work

Step 2

For i = 1

Need₁ = 0, 2, 0

Finish [1] is false and $\overset{0,\,2,\,0}{\text{Need}_1} < \overset{2,\,3,\,0}{\text{Work}}$ ✓

So P₁ must be kept in safe sequence

Step 3

Work = $\overset{2,\,3,\,0}{\text{Work}} + \overset{3,\,0,\,2}{\text{Allocation}_1}$

Work = | 5 | 3 | 2 | (A B C)

	0	1	2	3	4
Finish =	false	true	false	false	false

Step 2

For i = 2

Need₂ = 6 , 0, 0

Finish [2] is false and $\overset{6,\,0,\,0}{\text{Need}_2} > \overset{5,3,\,2}{\text{Work}}$ ✗

So P₂ must wait

Step 2

For i=3

Need₃ = 0, 1, 1

Finish [3] = false and $\overset{0,\,1,\,1}{\text{Need}_3} < \overset{5,\,3,\,2}{\text{Work}}$ ✓

So P₃ must be kept in safe sequence

Step 3

Work = $\overset{5,\,3,\,2}{\text{Work}} + \overset{2,\,1,\,1}{\text{Allocation}_3}$

Work = | 7 | 4 | 3 | (A B C)

	0	1	2	3	4
Finish =	false	true	false	true	false

Step 2

For i = 4

Need₄ = 4, 3, 1

Finish [4] = false and $\overset{4,\,3,\,1}{\text{Need}_4} < \overset{7,\,4,\,3}{\text{Work}}$ ✓

So P₄ must be kept in safe sequence

Step 3

Work = $\overset{7,\,4,\,3}{\text{Work}} + \overset{0,\,0,\,2}{\text{Allocation}_4}$

Work = | 7 | 4 | 5 | (A B C)

	0	1	2	3	4
Finish =	false	true	false	true	true

Step 2

For i = 0

Need₀ = 7, 4, 3

Finish [0] is false and $\overset{7,\,4,\,3}{\text{Need}} < \overset{7,\,4,\,5}{\text{Work}}$ ✓

So P₀ must be kept in safe sequence

Step 3

Work = $\overset{7,\,4,\,5}{\text{Work}} + \overset{0,\,1,\,0}{\text{Allocation}_0}$

Work = | 7 | 5 | 5 | (A B C)

	0	1	2	3	4
Finish =	true	true	false	true	true

Step 2

For i = 2

Need₂ = 6 , 0, 0

Finish [2] is false and $\overset{6,\,0,\,0}{\text{Need}_2} < \overset{7,\,5,\,5}{\text{Work}}$ ✓

So P₂ must be kept in safe sequence

Step 3

Work = $\overset{7,\,5,\,5}{\text{Work}} + \overset{3,\,0,\,2}{\text{Allocation}_2}$

Work = | 10 | 5 | 7 | (A B C)

	0	1	2	3	4
Finish =	true	true	true	true	true

Step 4

Finish [i] = true for 0 ≤ i ≤ n

Hence the system is in Safe state

The safe sequence is P₁,P₃ , P₄ ,P₀ ,P₂

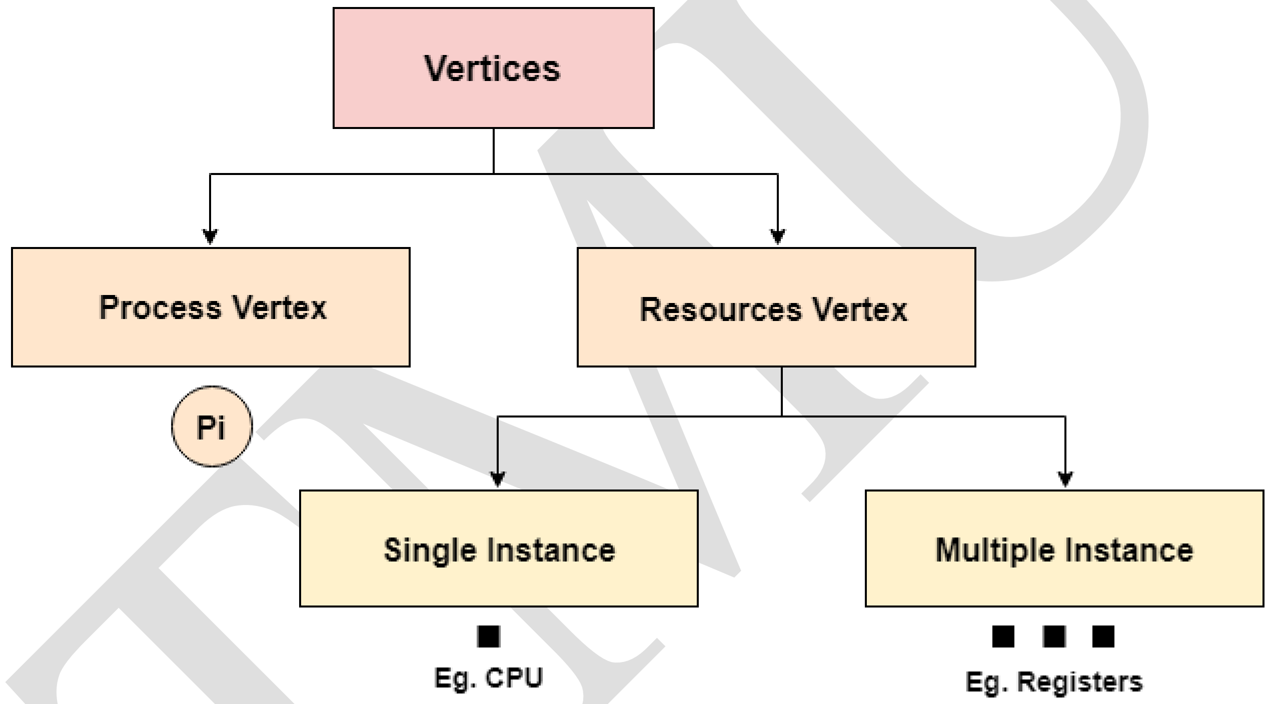
Hence the new system state is safe, so we can immediately grant the request for process **P1 .**

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.

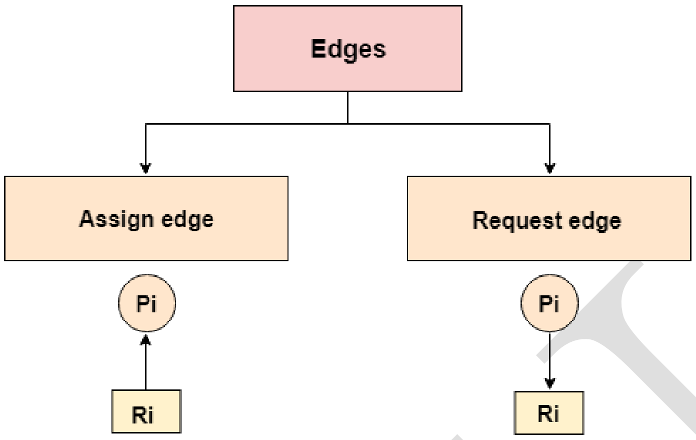
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by arectangle. 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 differentshape. Circle represents process while rectangle represents resource.

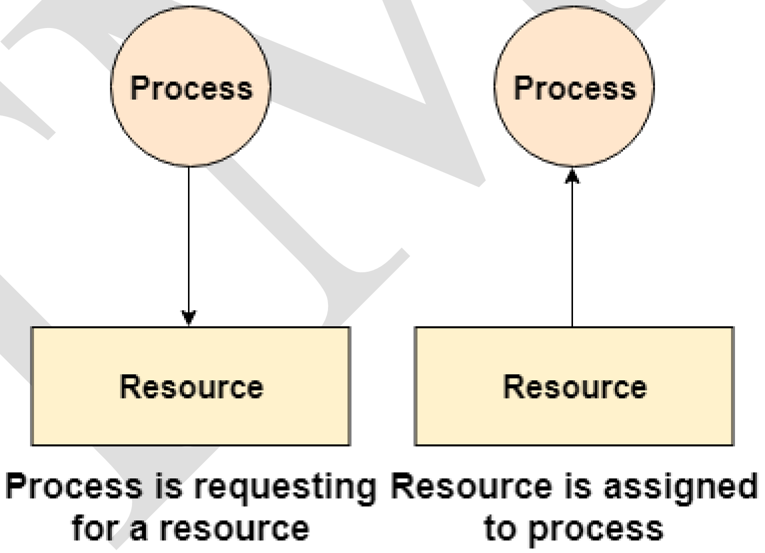
A resource can have more than one instance. Each instance will be represented by a dot inside therectangle.



Edges in RAG are also of two types, one represents assignment and other represents the wait of a processfor 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 theresource 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 headis pointing towards the resource.

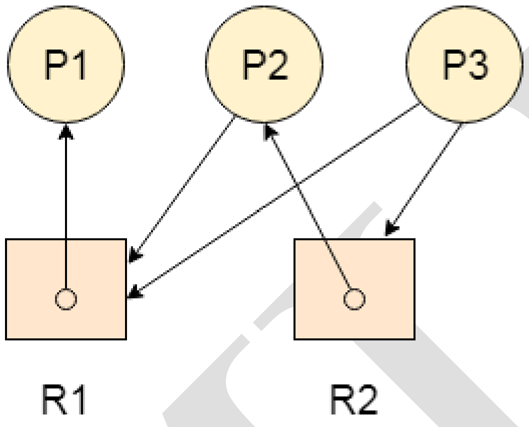

Process
Process
Resource
Resource
Process is requesting for a resource
Resource is assigned to process

Example

Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1as well as R2.

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

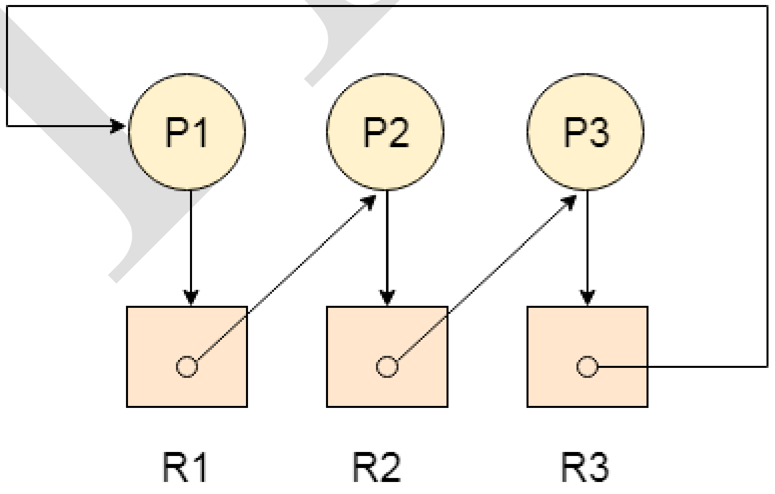


Deadlock Detection using RAG

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

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

The following example contains three processes P1, P2, P3 and three resources R2, R2, R3. All theresources 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 issatisfying 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)

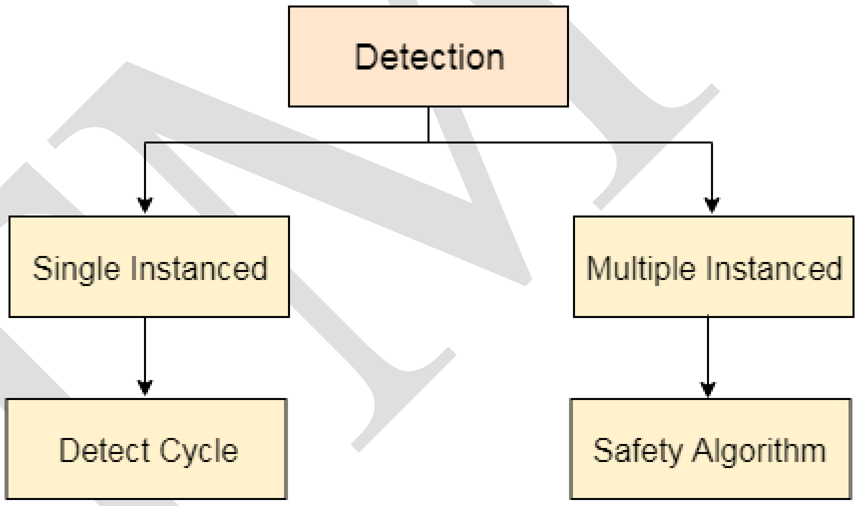
Neither we are having any resource available in the system nor a process going to release. Each of the process needs at least single resource to complete therefore they will continuously be holding each one of them.

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 can rollback 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.

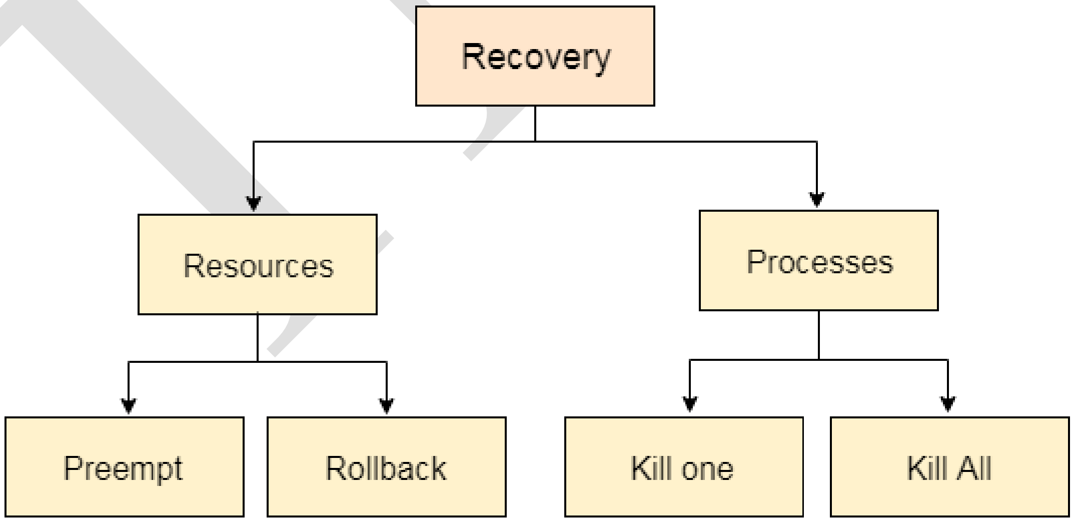
For Process

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
Resources
Processes
Preempt
Rollback
Kill one
Kill All