Name of Faculty: Dr Anoop Kumar Chaturvedi

Designation: Associate Professor

Department: CSE

Subject: Operating (MCA-201)

Unit: III

Topic: Deadlock

Deadlock in Operating System:

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.

Or in other words

A process in a multiprogramming system is said to be in a state of deadlock if its waiting for a particular event that will never occur.

One Process Deadlock:

If a process is given the task of waiting for an event to occur, and if the system includes no provision for signalling that event, then we have a one-process deadlock. Deadlock of this nature are extremely difficult to detect.

Process :

A process is an instance of a program running in a computer.

Resource :

Resources are the physical or virtual components of limited availability within a computer system. Every device connected to a computer system is a resource. Every internal system component is a resource. Virtual system resources include files, network connections and memory areas.

A process in operating systems uses different resources and uses resources in following way.

1) Requests a resource

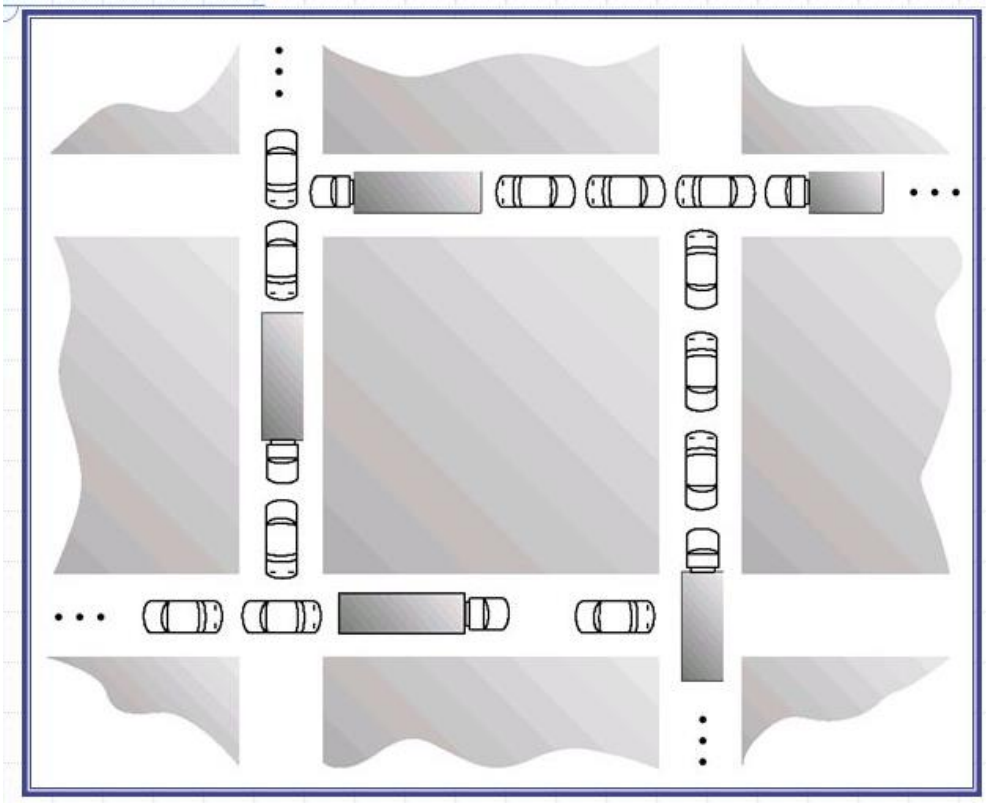
2) Use the resource

2) Releases the resource

Example of deadlock:

1. A traffic deadlock:

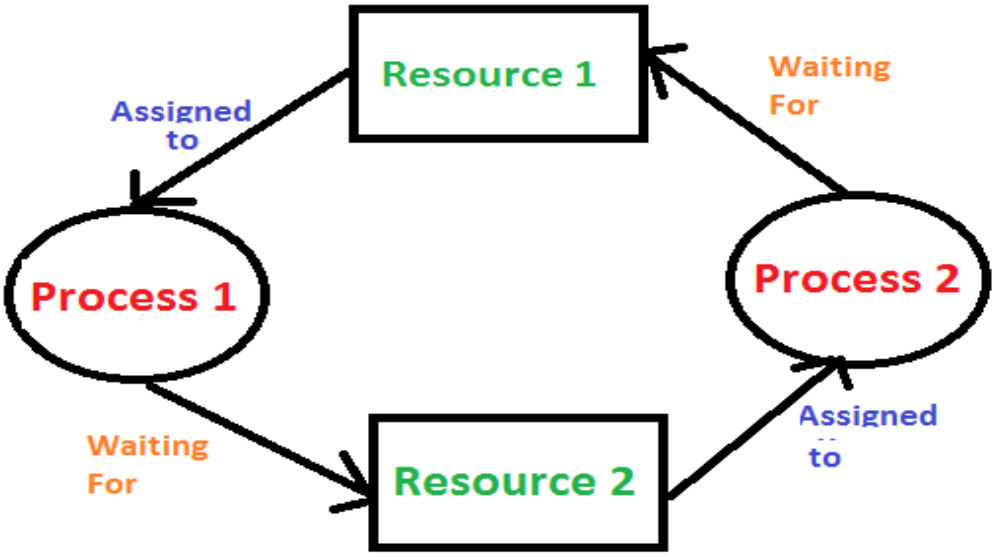
   A number of automobiles are attempting to drive through a busy section of the city, but traffic has becomes completely snarled. Traffic comes to a halt.



Traffic Deadlock

2. Assume a system with four tape drives and two processes. If each process has 2 tape drives and needs a third one in order to proceed, then each will wait for the other and processes are is in deadlocked.
3. Deadlock can occur if: Person starts to cross river without first checking to see if someone else is trying to cross from the other side in the opposite direction. Two people start crossing river from opposite sides and there is single stepping stone and meet in the middle.
4. A Simple Resource deadlock

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.



Four Necessary Conditions for Deadlock:

i. Mutual Exclusion: One or more than one resource are non-sharable (Only one process can use at a time)

ii. Hold and Wait: A process is holding at least one resource and waiting for resources.

iii. No Preemption: A resource cannot be taken from a process unless the process releases the resource.

iv. Circular Wait: Each process holds one or more resources that are requested by the next process in the chain.

Deadlock Prevention : Deadlock prevention ensure that at least one of the necessary conditions (Mutual exclusion, hold and wait, no preemption and circular wait) does not hold true.

i. Denying the Mutual Exclusion Condition

ii. Denying the "Hold and Wait " Condition

iii. Denying the "No-Preemption" Condition

iv. Denying the "Circular Wait" Condition

Safe and Unsafe State:

Safe State, is when there is no chance of deadlock occuring, while **unsafe state** doesn't mean a deadlock has occurred yet, but means that a deadlock could happen.

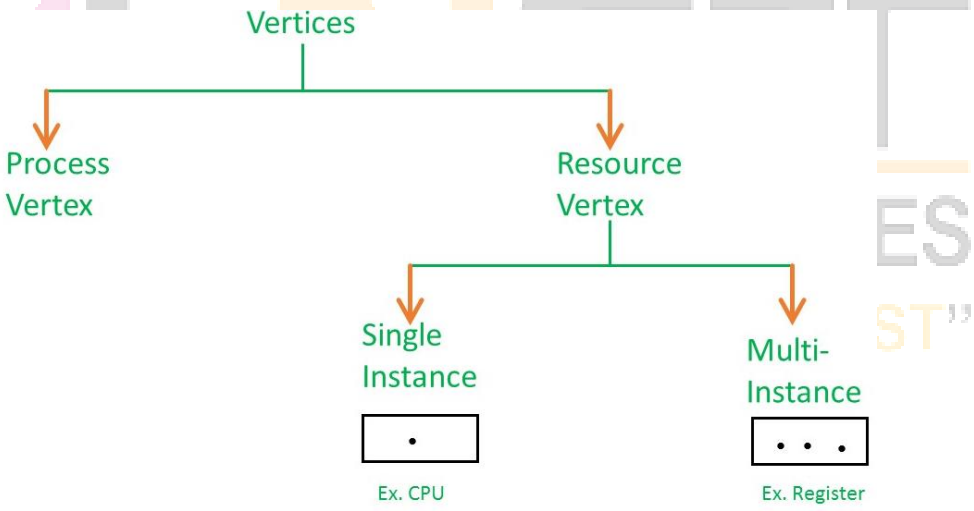
A **state** is **safe** if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a **deadlock state**. If a **safe** sequence does not exist, then the system is in an unsafe **state**, which MAY lead to **deadlock**.

Resource Allocation Graph (RAG):

Resource Allocation Graph (RAG) is explained to us what is the state of the system in terms of processes and resources. Like how many resources are available, how many are allocated and what is the request of each process. Everything can be represented in terms of the diagram. One of the advantages of having a diagram is, sometimes it is possible to see a deadlock directly by using RAG.

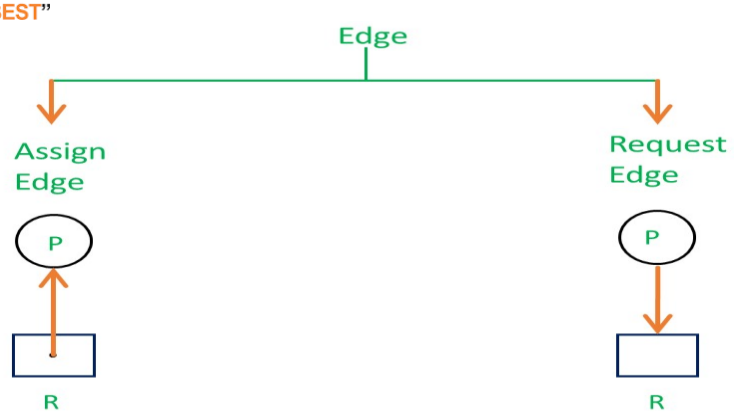
We know that any graph contains vertices and edges. In RAG vertices are of two type –

1. **Process vertex –** Every process will be represented as a process vertex. Generally, the process will be represented with a circle.
2. **Resource vertex –** Every resource will be represented as a resource vertex. It is also two type –
   - **i.** **Single instance type resource –** It represents as a box, inside the box, there will be one dot. So the number of dots indicate how many instances are present of each resource type.
   - **ii.** **Multi-resource instance type resource –** It also represents as a box, inside the box, there will be many dots present.



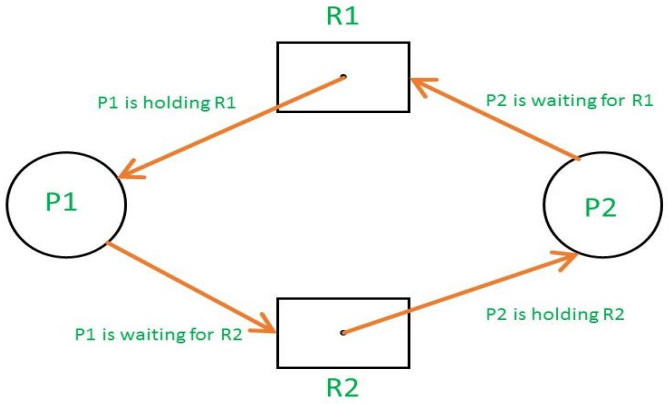
There are two types of edges in RAG –

1. Assign Edge – If you already assign a resource to a process then it is called Assign edge.
2. Request Edge – It means in future the process might want some resource to complete the execution, that is called request edge.



So, if a process is using a resource, an arrow is drawn from the resource node to the process node. If a process is requesting a resource, an arrow is drawn from the process node to the resource node.

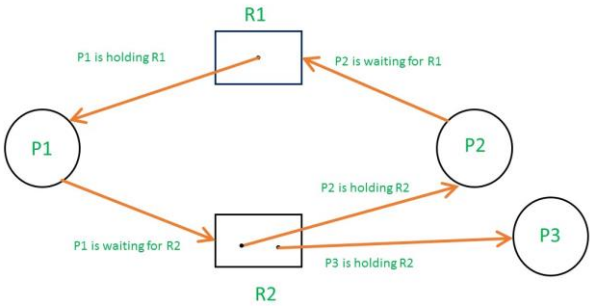
Example 1 (Single instances RAG) –



SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock. For example, if process P1 holds resource R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in deadlock.

Example 2 (Multi-instances RAG) –



MULTI INSTANCES WITHOUT DEADLOCK

From the above example, it is not possible to say the RAG is in a safe state or in an unsafe state. So to see the state of this RAG, let's construct the allocation matrix and request matrix.

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

- The total number of processes are three; P1, P2 & P3 and the total number of resources are two; R1 & R2.

Allocation matrix –

- For constructing the allocation matrix, just go to the resources and see to which process it is allocated.
- R1 is allocated to P1, therefore write 1 in allocation matrix and similarly, R2 is allocated to P2 as well as P3 and for the remaining element just write 0.

Request matrix –

- In order to find out the request matrix, you have to go to the process and see the outgoing edges.
- P1 is requesting resource R2, so write 1 in the matrix and similarly, P2 requesting R1 and for the remaining element write 0.

So now available resource is = (0, 0).

Checking deadlock (safe or not) –

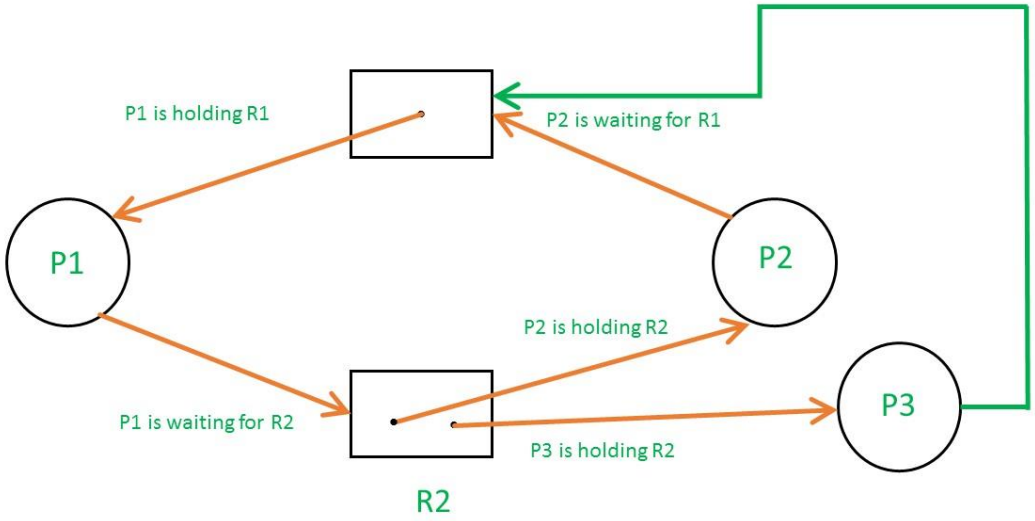
Available = 0 0 (As P3 does not require any extra resource to complete the execution and after
P3 0 1 completion P3 release its own resource)

New Available = 0 1 (As using new available resource we can satisfy the requirment of process P1
P1 1 0 and P1 also release its prevous resource)

New Available = 1 1 (Now easily we can satisfy the requirement of process P2)
P2 0 1

New Available = 1 2

So, there is no deadlock in this RAG. Even though there is a cycle, still there is no deadlock. Therefore in multi-instance resource cycle is not sufficient condition for deadlock.



MULTI INSTANCES WITH DEADLOCK

Above example is the same as the previous example except that, the process P3 requesting for resource R1.
So the table becomes as shown in below.

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

So,the Available resource is = (0, 0), but requirement are (0, 1), (1, 0) and (1, 0).So you can't fulfill any one requirement.Therefore, it is in deadlock.

Therefore, every cycle in a multi-instance resource type graph is not a deadlock, if there has to be a deadlock, there has to be a cycle.So, in case of RAG with multi-instance resource type, the cycle is a necessary condition for deadlock, but not sufficient.

Deadlock avoidance:

The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition. 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.

Deadlock avoidance can be done with Banker's Algorithm.

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.

Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting 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.

Following **Data structures** are used to implement the Banker's Algorithm:

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max :

- It is a 2-d array of size '**n*m**' that defines the maximum demand of each process in a system.
- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$ for its execution.
- Need [i, j] = Max [i, j] – Allocation [i, j]

$Allocation_i$ specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

Banker's algorithm consists of Safety algorithm and Resource request algorithm

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

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) $Need_i$ <= 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 otherwise the system is in unsafe 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 of resource 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
$Allocation_i = Allocation_i + Request_i$
$Need_i = Need_i – Request_i$

This step is done because the system needs to assume that resources have been allocated. So there will be less resources available after allocation. The number of allocated instances will increase. The need of the resources by the process will reduce. That's what is represented by the above three operations.

After completing the above three steps, check if the system is in safe state by applying the safety algorithm. If it is in safe state, proceed to allocate the requested resources. Else, the process has to wait longer.

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	

Answer the following:

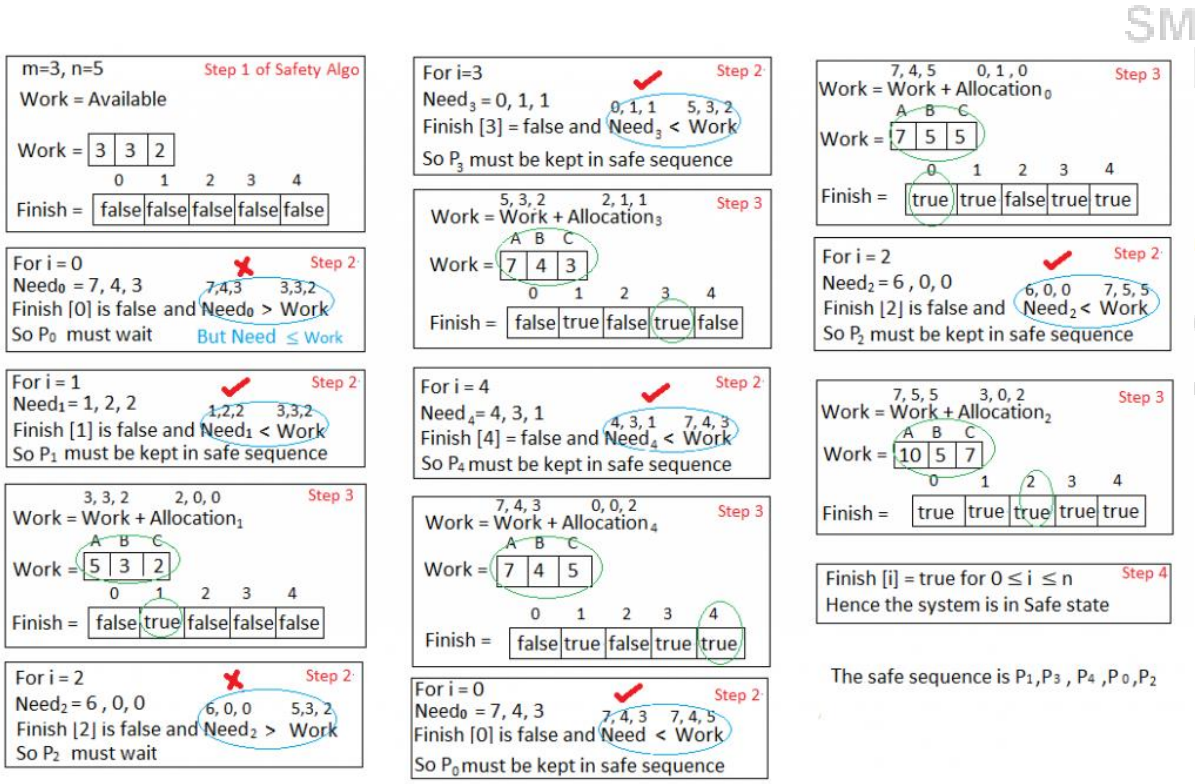
1. What will be the content of Need Matrix?

Answer:-

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

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

Answer :-



3. What will happen if process P_1 requests one additional instance of resource type A and two instances of resource type C?

Answer:-

A B C
Request$_1$ = 1, 0, 2

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

Step 1

1, 0, 2 < 1, 2, 2
Request$_1$ < Need$_1$

Step 2

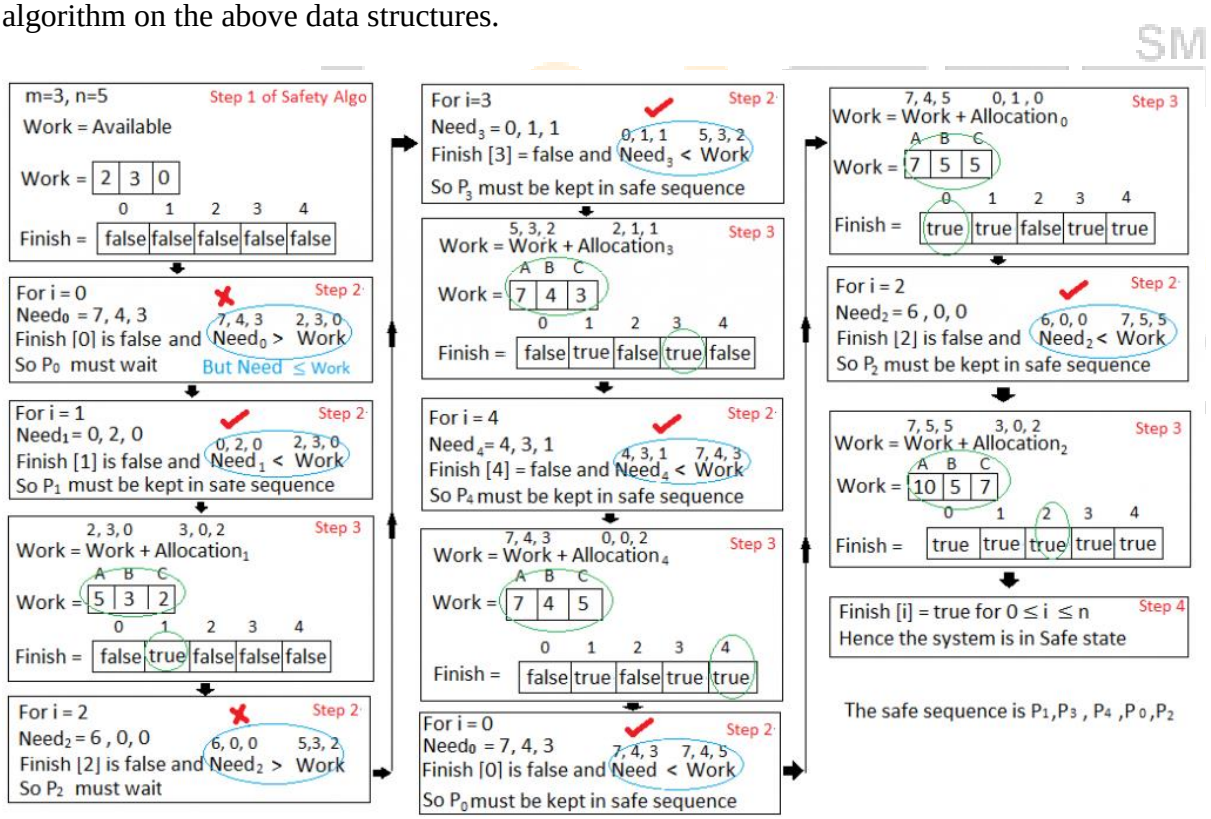
1, 0, 2 < 3, 3, 2
Request$_1$ < Available

Step 3

Available = Available – Request$_1$
Allocation$_1$ = Allocation$_1$ + Request$_1$
Need$_1$ = Need$_1$ - Request$_1$

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

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



Starvation: **Starvation** is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.

Difference between Deadlock and Starvation:

S.NO	Deadlock	Starvation
1.	All processes keep waiting for each	High priority processes keep

	other to complete and none get executed	executing and low priority processes are blocked
2.	Resources are blocked by the processes	Resources are continuously utilized by high priority processes
3.	Necessary conditions Mutual Exclusion, Hold and Wait, No preemption, Circular Wait	Priorities are assigned to the processes
4.	Also known as Circular wait	Also know as lived lock
5.	It can be prevented by avoiding the necessary conditions for deadlock	It can be prevented by Aging

Aging :

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example, if priority range from 127(low) to 0(high), we could increase the priority of a waiting process by 1 Every 15 minutes.

Livelock occurs when two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work. These processes are not in the waiting state, and they are running concurrently. This is different from a deadlock because in a deadlock all processes are in the waiting state.

References –

1. A. Silberschatz, P. Galvin, G. Gagne, "Operating Systems Concepts (8th Edition)", Wiley India Pvt. Ltd.

2. Internet.

3. H.M. Deitel, "Operating System".