Deadlock

In Multiprogramming environment, several processes are competing for number of resources. A process requests resources and if the resource is not available at that time, the process enters into waiting state. The waiting state of the processes may never again be change because the resources they are requested are held by some other waiting processes in the queue. This situation is called **Deadlock**.

"Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process."

Definition: *"A set of processes in a deadlock state when every process in the set is waiting for an event that can only be caused by another process in the set"*

System Model:

- A system consists of a finite number of resources to be distributed among a number of competing processes.
- The resources are partitioned into several types, each of which consists of some number of identical instances. **e.g.** memory space, CPU cycle, files and I/O devices are of resource type. If the system has two CPUs then the resource type CPU has TWO instances.
- If the process requests an instance of a resource type, the allocation of any instance of the type will satisfy the request.
- A process must request a resource before using it and must release the resources after using it. A process may request as many resources as it may require carrying out its designated task.
- The Number of resources requested may not exceed the total number of resources available in the system.
- A Process may utilize a resource in the following sequence:
    1. **Request:** If the request cannot be granted immediately, then the requested process must wait until it can acquire the resources.

ii. **Use:** The process can operate on the resources.

iii. **Release:** The process releases the resources.

- For each use, the OS checks to make sure that the using process has requested and been allocated the resources.
- A system table records whether each resource is free or allocated and if resources are allocated to which process.
- If a process requests resources that are currently allocated to another process, it can be added to a queue of processes waiting for this resource.
- The events with which we are mainly concerned are resource acquisition and release.
- The resources may be physically (like printers, CPU cycles, memory space, tape drives etc) or logical (like files, semaphore, monitors etc)
- Multithreaded programs are good candidate for deadlocks because multiple threads can compete for shared resources. Therefore multithreaded programs should be carefully developed.

Deadlock Characteristics

In deadlock process never finish executing and the system resources are tied up preventing other job from starting. The following feature characterizes the deadlock.

1. **Necessary Conditions:** A deadlock situation can arise if the following **FOUR** conditions hold simultaneously in the system.

i. **Mutual Exclusion:** At least, one resource must be non-sharable mode. That is only one process at time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

ii. **Hold and Wait:** A Process must be holding some resource and waiting to acquire some additional resources that are currently being held by other processes.

iii. **No Preemption:** Resources cannot be preempted i.e a resource can be released only voluntarily by the process holding it, after the process has completed its task.

iv. **Circular Wait:** A set **{P_0,P_1,P_2,P_3,P_4………. P_n}** of waiting process must exist such that P_0 is waiting for P_1, P_1 is waiting for P_2 and so on and P_n is waiting for P_0 making the circular condition.

- All the above FOUR condition must occur to deadlock to occur in the system.
- The circular wait condition implies that Hold and Wait Condition, so the Four conditions are not completely independent.

Resources Allocation Graph

Deadlock can be described more precisely in the term of directed graph called Resource Allocation Graph.

This Graph consists of set of vertices V and set edges E. The set of vertices V is partitioned into Two types of nodes **P** = **{P_0, P_1, P_2, P_3, P_4………. P_n} :** set of all active processes in the system **and R = {R_0,R_1,R_2….. R_m}:** Set of resources in the system**.**

The directed edge from process **P_i** to resource type **R_j** is denoted by **$P_i \rightarrow R_j$;** it signifies process **P_i** requests for resource type **R_j.** This is called **Request Edge**.

The directed edge from resource type **R_j** to process **P_i** is denoted by **$R_j \rightarrow P_i$;** it signifies resource type **R_j** is allotted to process **P_i .** This is called **Assignment Edge**.

- Processes are represented by circle and resources are represented by Square.
- Each instance is represented by a dot within a square.
- Request edge points to only the square Rj, whereas an assignment edge must also designated one of the dot in the square.
- When Process Pi requests an instance of resource type Rj. A request Edge is inserted in the graph.
- When a request can be fulfilled, the request edge is transformed into assignment edge.
- When the process no longer needs access to the resource it releases the resources first and assignment edge is deleted.

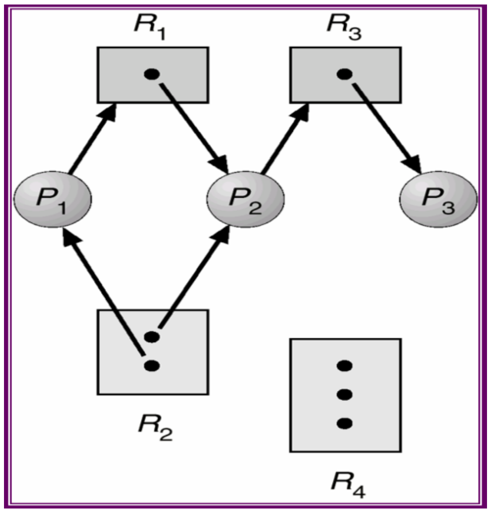
Example:

P = {P_1, P_2, P_3}

R = {R_1, R_2 , R_3, R_4}

E = { $P_1 \rightarrow$R1, P2$\rightarrow$R3, R1$\rightarrow$P2, R2$\rightarrow$P2, R2$\rightarrow$P1, R3$\rightarrow$P3}

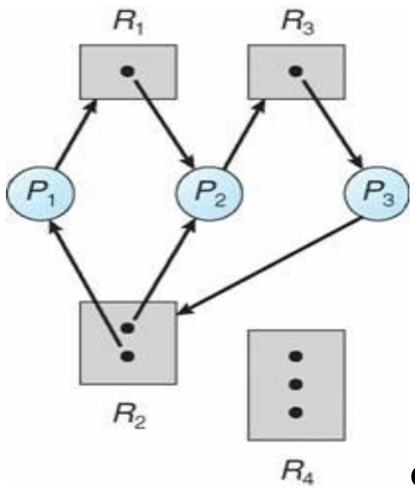
There are ONE, TWO, ONE and THREE instances of R_1, R_2 , R_3 and R_4 type resources.



- If the graph contains a cycle, Deadlock may exist.
- If all resources contain only Single Instances of all resource type then cycle implies Deadlock.
- Each process involved in the cycle is deadlocked.
- Cycle is necessary and sufficient condition for system to be in deadlock. In this case, cycle is necessary but not sufficient condition for system to be in deadlock.

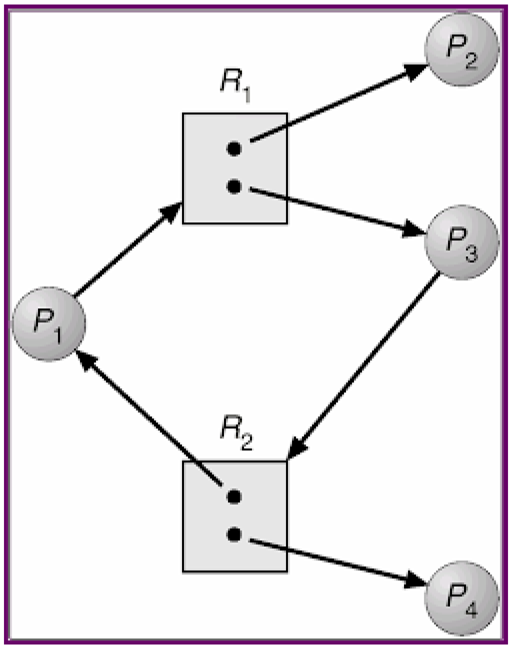
  So, if the Resource Allocation Graph does not have the cycle, then the system is not in deadlock state. On the other hand, if there is a cycle, then the system may or may not be in dead lock state.

If E = {P1 →R1→ P2→R3→P3→R2→P1, P2→R3→P3→R2→P2}



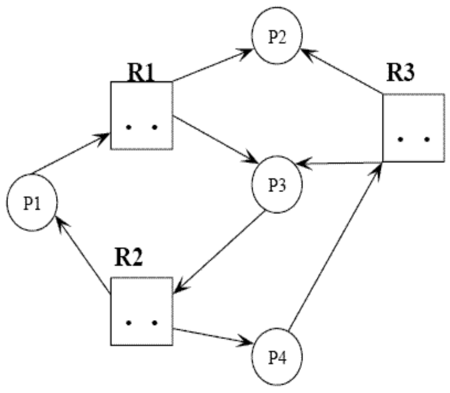
Graph with Deadlock

Example:

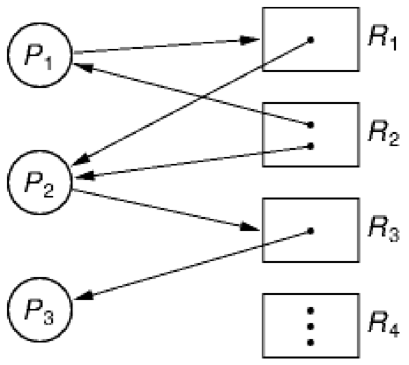


(Resource Allocation Graph with No Dead Lock)

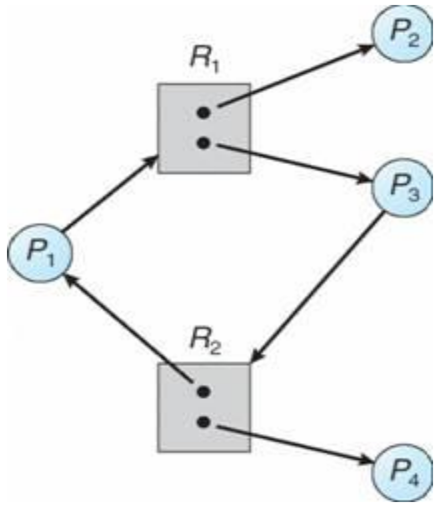
Example: Find Deadlock state or No Deadlock state



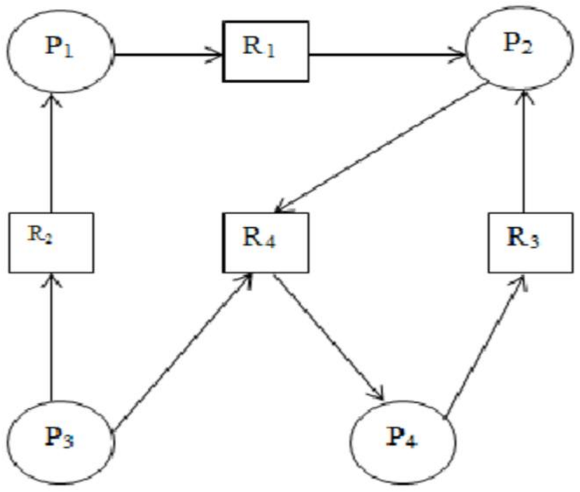
Example: Find Deadlock state or No Deadlock state



Example: Find Deadlock state or No Deadlock state



Example: Find Deadlock state or No Deadlock state



Methods for Deadlock Handling:

We can deal the deadlock in one of the **THREE** ways:

1. Do not allow the system to get into a deadlocked State: [**Deadlock Prevention and Avoidance**]. By using protocol to prevent or avoid deadlocks ensuring that the system will never enter a deadlock state.
2. We can allow the system to enter a deadlock state, detecting it and recovering it. [**Deadlock detection and recovery**]. Abort a process or preempt some resources when deadlocks are detected.
3. **Ignore the problem all together**

To ensure that deadlock never occurs, the system can use either deadlock prevention or deadlock avoidance schemes.

- If the system does not employ either deadlock prevention or a deadlock avoidance algorithm, then deadlock may occur.
- Then, the system can provide the algorithm that examine the state of the system for detecting deadlock and an algorithm to recover from the deadlock.
- If the system does not ensure that a deadlock will never occur and also does not provide a mechanism for deadlock detection and recovery, then the deadlock may occur and yet no way to recognize what has happened. In this case, the undetected deadlock will result in the deterioration of the system performance.
- Sometimes, the system is in frozen state but not in deadlock state. In this case systems must have manual recovery methods for non deadlock condition and may simply use those techniques for deadlock recovery.

Deadlock Prevention:

For a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one these four conditions cannot hold, we can prevent the occurrence of deadlock.

i. **Mutual Exclusion:** This condition must hold for non-sharable resources. Sharable resources on the other hand, do not require mutually exclusive access and thus cannot be involved in a deadlock.
A process never needs to wait for sharable resources. In general, we cannot prevent deadlocks by denying the mutual exclusion condition: **some resources are intrinsically Non-sharable.**

ii. **Hold and Wait:** To ensure that this condition never occurs in the system, we must guarantee that whenever process request a resource, it does not hold any other resource.

- "One Protocol that can be used requires each process to request and be allocated all its resources before it execution. We can implement this provision by requiring that system calls requesting other resources for a process precede all other system call. "

- "An Alternative protocol allows a process to request a resource only when the process has none. A process may request some resources and use them. Before it can request any additional resources, however it must release all the resources that it is currently allocated"

There are TWO disadvantages of this protocol

- Low Resource Utilization
- Starvation

iii. **No Preemption:** There should be No Preemption of Resources that has already been allocated. To ensure this, we can use the following protocol:

"If a Process is holding some resources and requests another resources that cannot be immediately allocated to it (**that is Process must wait**), then all resources currently being held are preempted. In other words, these resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting. The process will be restated only when it can regain its old resources as well as the new ones that it is requesting**"**

Alternatively, if a process requests some resources which are available then allocate them to process. If they are not available and are allocating to some waiting process which is waiting for some additional resources, we preempt the desired resources from the waiting process and allocate them to the requesting process.

If the resources are not either available or hold by a waiting process, the requesting process must wait. This protocol is often applied to the resources whose state can i.e. easily save and restored later, such as CPU registers and memory space.

iv. **Circular Wait:** To ensure that this condition never holds is to impose to total ordering of all resources types and to require that each process requests resources in an increasing order of enumeration.

R = {$\mathbf{R_0}$, $\mathbf{R_1}$, $\mathbf{R_2}$..... $\mathbf{R_m}$} be the set of resource type. We assign to each resources type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering.

For example: - F (Tape Drive) =1

F(Disk Drive) = 5;

F(Printer) = 12,

The following protocol is used to prevent deadlock:

"Each process can request resources only in an increasing order of enumeration. That is, a process can initially request any number of instances of a resource type say R_i. After that the process can request instances of resources type R_j if and only **if F(R_j) > F(R_i).**

If several instances of the same resources type are needed, a single request for all of them must be issued"

"Alternatively, we can require that whenever a process requests an instances of resources type R_j, it has released any resources R_i such that **F(R_i) >= F(R_j)."**

If these two protocols are used, then the circular wait condition cannot hold.

Remark: - Deadlock *prevention algorithm prevents deadlock by restraining how requests can be made. The restraints ensure that at least one of the necessary conditions for deadlock cannot occur and hence, that deadlock cannot hold. Possible side effects of preventing deadlock by this method however are low device utilizing and reduced system throughput*

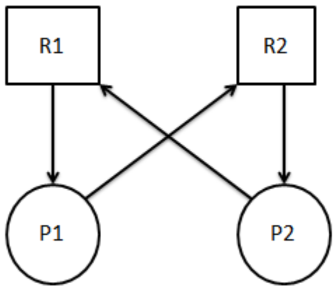
Deadlock Prevention PROTOCOLS: (Short Notes)

1. ***Mutual Exclusion:***

   Sharable resources and non-sharable are intrinsic property of the resource type cannot be changed. This problem can be overcome by adding more instances of the resources.

2. ***Hold and Wait***
   - *Conservative Property*
   - *Do not Hold*
3. *No Preemption*
   - *Forceful Preemption*
4. *Circular Wait*
   - *Order of Acquisition of Resource*

P1	*P2*
R1	*R2*
R2	*R1*



- **Fix the Request Order of the Resources :**

P1	*P2*
R1	*R1*
R2	*R2*

But request order cannot be determined initially.

- Assign unique natural number to each resource type and process can request either in increasing order or decreasing of resource type. If Process is having some high order resources and request low order resources then process must leave all high order resources first.

Deadlock Avoidance:

By additional information about resources are to be requested, the deadlocks can be avoided.
With the knowledge of the complete sequence of requests and releases for each process, we can decide for each request whether or not the process should wait.
Each request requires that the system considers the resources currently available, the resources currently allocated to each process, to decide whether the current request can be satisfied or must wait to avoid a possible future deadlock.
The algorithm differs in the amount and the type of information required.
A deadlock avoidance algorithm dynamically examine the resources the resource allocation state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources and the maximum demands of the process.

Safe State: A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock.
A system is in safe state only of the there exist **a safe sequence**.

- For example, consider a system with 12 tape drives, allocated as follows. Is this a safe state? What is the safe sequence?

	Maximum Needs	**Current Allocation**	**Future Need = Max- Allocation**
P0	10	5	05
P1	4	2	05
P2	9	2	07

Number of free Resource Type is **(Total Available) - (Currently Allocated)**

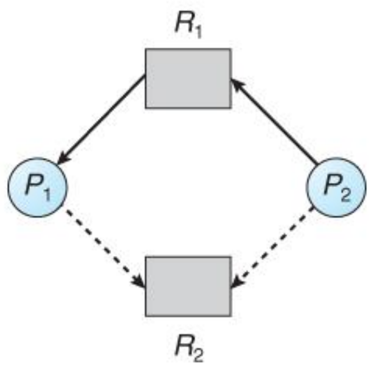
Currently Available = 12 -09 = 03

Safe Sequence = < P1, P0, P2>

- What happens to the above table if process P2 requests and is granted one more tape drive?

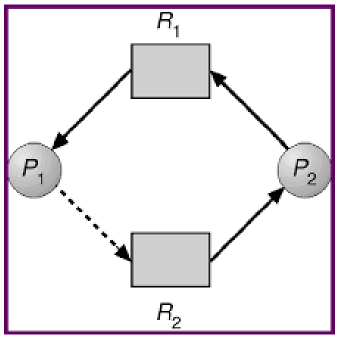
Resource Allocation Graph Algorithm for Deadlock Avoidance:

In the Resource Allocation Graph, We introduce, a one new edge called **claim Edge**. Pi→Rj claim edge means Process Pi claims for resource type Rj in future. It is similar with request edge but the only difference is that it is represented by dashed lines.



The claim edge is converted to request edge. Similarly when resource is released the assignment edge is again converted to claim edge.
In this algorithm, every process has claim edge to the resources needed in future. The resources are allocated to a process or claim edge is converted to request edge/ assignment edge if there is no cycle in the graph.
Cycle detection algorithm determines a cycle in the Resource Allocation Graph.

Suppose that P2 request for resource R2. Although R2 is currently free but R2 is not allocated to P2 as it creates cycle or system is in unsafe state.



Similarly, P1 request for resource R2. Here, R2 is currently free and allocated to P1 as it will not create cycle or system is in safe state.

Banker's Algorithm:

- The Resource Allocation Graph is not applicable for the resources having multiple instances of each resource type.
- Banker's Algorithm is application to such system but less efficient than Resource Allocation Graph algorithm.
- The Banker's Algorithm for deadlock avoidance works in the same as Banks does. The never allocate its available resources in such a way that it could no longer satisfy the needs of all its customer.
- New process that enters the system, must declare the maximum number of instances of each resource type that it may need.

- When the user request a set of resources, the system must determine whether the allocation of these resources will leave the system in safe state.

Several data structure is maintained to implement the Banker's Algorithm:

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:

- **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
- **Max:** It is an **n x m** matrix which represents the maximum number of instances of each resource that a process can request.

  If Max[i][j] = k, then the process P_i can request almost k instances of resource type R_j
- **Allocation:** It is an n x m matrix which represents the number of resources of each type currently allocated to each process.

  If Allocation[i][j] = k, then process P(i) is currently allocated k instances of resource type R(j).
- **Need:** It is an n x m matrix which indicates the remaining resource needs of each process.

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

  If Need[i][j] = k, then process P(i) may need k more instances of resource type R(j) to complete its task.

Safety Algorithm:

The algorithm for finding out whether is safe state or not is described as follow:

1. Let Work and Finish be vectors of length m and n, respectively. Initially,

    Work = Available

    Finish[i] =false for i = 0, 1, 2, ..., n - 1.

    This means, initially, no process has finished and the number of available resources is repre sented by the **Available** array.

2. Find an index **i** such that both
    a. Finish[i] ==false
    b. Need_i <= Work

    If there is no such i present, then proceed to **step 4**.

    It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:
    a. Work = Work + Allocation;
    b. Finish[i] = true;

    Go to **step 2**.

    When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

4. If Finish[i] == true for all i, then the system is in a safe state.

    OR

    If Finish[i] == False for some 0<=i<=n then system is in deadlock state.

Note: This algorithm may require an order of m x n2 operation to decide whether state is safe or not.

Resource Request Algorithm:

Let $Request_i$ be the request vector for process Pi. If the Request[i][j]=k then process Pi may request instance of Resource type Rj. When a request for resource is made by process Pi, the following actions are taken:

1. **If** $Request_i$ <= $Need_i$ **Goto Step-2** Raise an error
2. **If** $Request_i$ <= $Available_i$ **Goto Step-3** otherwise Pi must wait until resources are not available.
3. Available = Available - Request(i)

   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 types 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	

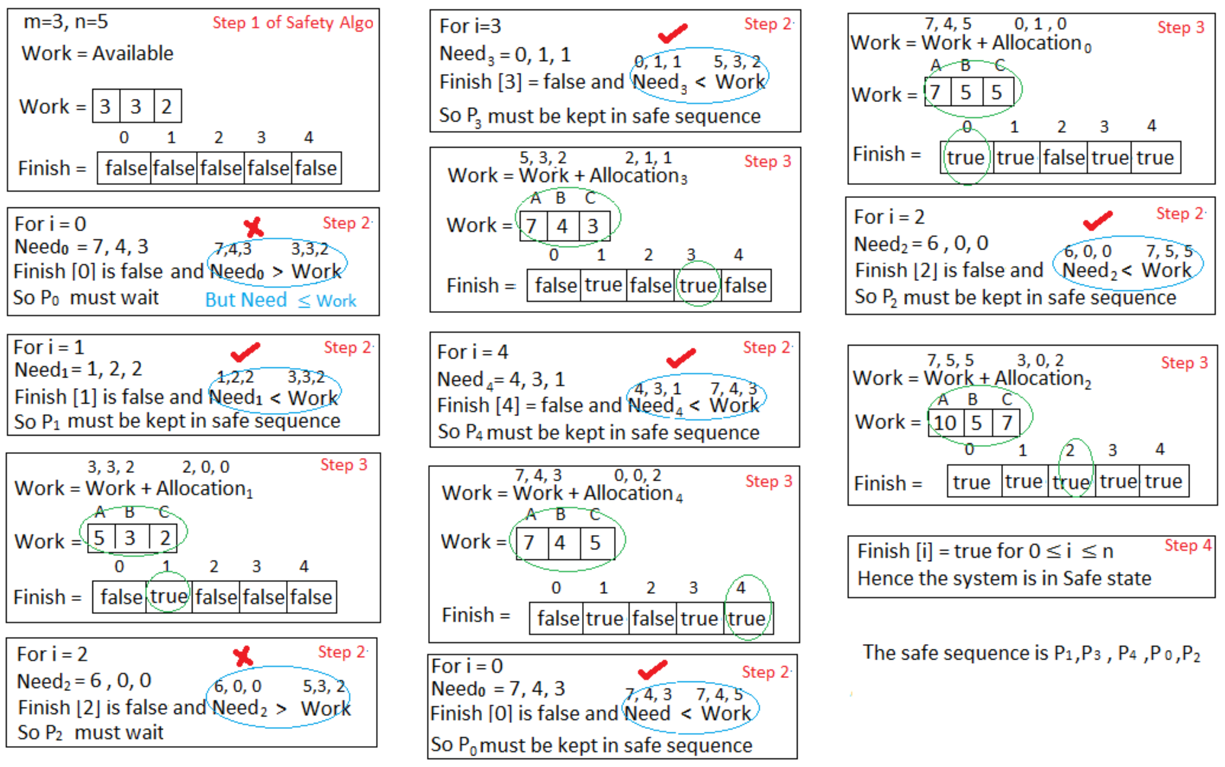
a. What will be the content of the Need matrix?

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

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

:

b. Is the system in safe state? If yes, then what is the safe sequence?



m=3, n=5 Step 1 of Safety Algo

Work = Available

Work = $\begin{array}{|c|c|c|}\hline 3 & 3 & 2 \\ \hline\end{array}$

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

For i = 0 Step 2

$Need_0$ = 7, 4, 3

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

So P_0 must wait But Need $\leq$ 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 = $\overset{3,\,3,\,2}{Work}$ + $\overset{2,\,0,\,0}{Allocation_1}$

Work = $\begin{array}{|c|c|c|} A & B & C \\ \hline 5 & 3 & 2 \\ \hline\end{array}$

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

Work = $\begin{array}{|c|c|c|} A & B & C \\ \hline 7 & 4 & 3 \\ \hline\end{array}$

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

Work = $\begin{array}{|c|c|c|} A & B & C \\ \hline 7 & 4 & 5 \\ \hline\end{array}$

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

Work = $\begin{array}{|c|c|c|} A & B & C \\ \hline 7 & 5 & 5 \\ \hline\end{array}$

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

Work = $\begin{array}{|c|c|c|} A & B & C \\ \hline 10 & 5 & 7 \\ \hline\end{array}$

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

Finish [i] = true for $0 \leq i \leq n$ Step 4

Hence the system is in Safe state

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

Safe Sequence = <P1, P3, P4, P0, P2>

c. What will happen if process P_1 requests one additional instance of resource type A and two instan ces of resource type C? A = < 1, 0, 2>

$$\mathrm{Request}_1 = \begin{array}{ccc} A & B & C \\ 1, & 0, & 2 \end{array}$$

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

Step 1

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

Step 2

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

Step 3

Available = Available – Request₁

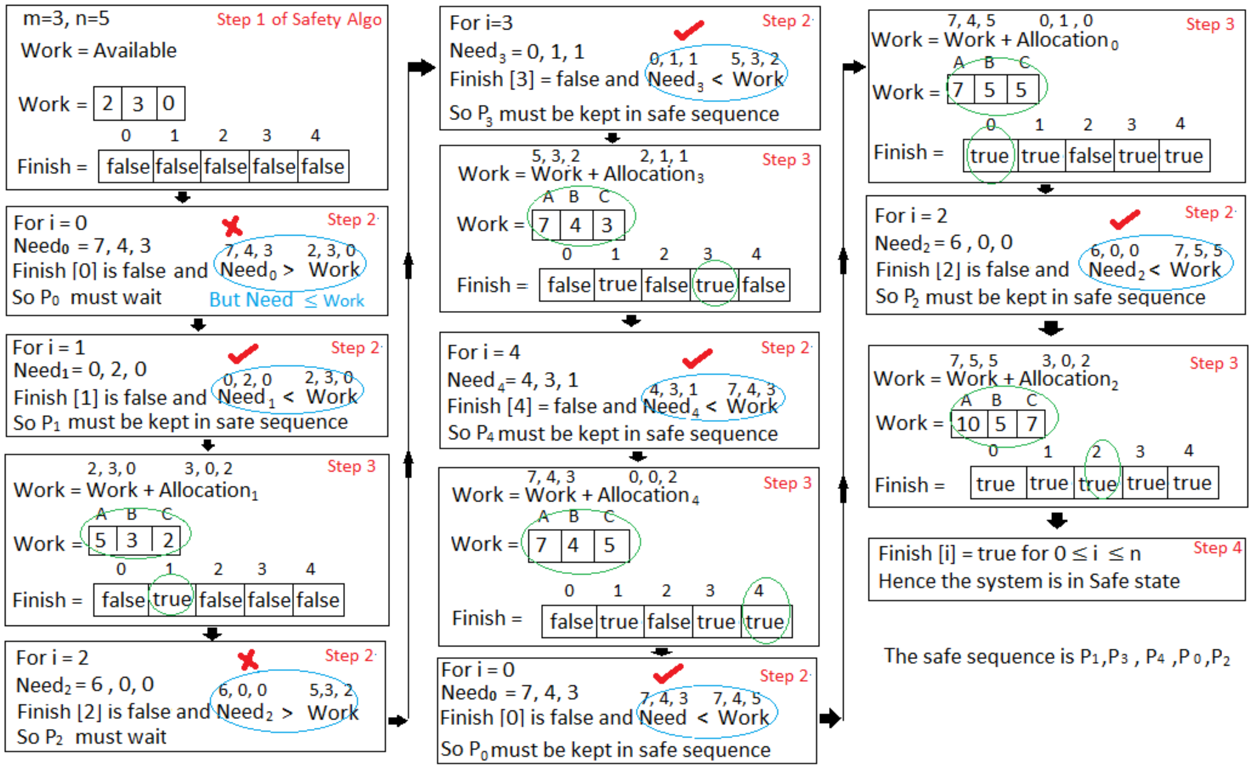
Allocation₁ = Allocation₁ + Request₁

Need₁ = Need₁ - Request₁

Process	Allocation	Need	Available
	A B C	A B C	A B C
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	

Now, determine whether this new system state is safe. If the State is Safe, Request will be granted immediately.

To do so, we again execute Safety algorithm on the 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 $\overset{7,4,3}{\mathrm{Need}_0} > \overset{2,3,0}{\mathrm{Work}}$
So P₀ must wait
But Need ≤ Work

Step 2
For i = 1
Need₁ = 0, 2, 0
Finish [1] is false and $\overset{0,2,0}{\mathrm{Need}_1} < \overset{2,3,0}{\mathrm{Work}}$
So P₁ must be kept in safe sequence

Step 3
Work = $\overset{2,3,0}{\mathrm{Work}} + \overset{3,0,2}{\mathrm{Allocation}_1}$
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 $\overset{6,0,0}{\mathrm{Need}_2} > \overset{5,3,2}{\mathrm{Work}}$
So P₂ must wait

Step 2
For i=3
Need₃ = 0, 1, 1
Finish [3] = false and $\overset{0,1,1}{\mathrm{Need}_3} < \overset{5,3,2}{\mathrm{Work}}$
So P₃ must be kept in safe sequence

Step 3
Work = $\overset{5,3,2}{\mathrm{Work}} + \overset{2,1,1}{\mathrm{Allocation}_3}$
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 $\overset{4,3,1}{\mathrm{Need}_4} < \overset{7,4,3}{\mathrm{Work}}$
So P₄ must be kept in safe sequence

Step 3
Work = $\overset{7,4,3}{\mathrm{Work}} + \overset{0,0,2}{\mathrm{Allocation}_4}$
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 $\overset{7,4,3}{\mathrm{Need}} < \overset{7,4,5}{\mathrm{Work}}$
So P₀ must be kept in safe sequence

Step 3
Work = $\overset{7,4,5}{\mathrm{Work}} + \overset{0,1,0}{\mathrm{Allocation}_0}$
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 $\overset{6,0,0}{\mathrm{Need}_2} < \overset{7,5,5}{\mathrm{Work}}$
So P₂ must be kept in safe sequence

Step 3
Work = $\overset{7,5,5}{\mathrm{Work}} + \overset{3,0,2}{\mathrm{Allocation}_2}$
Work = | 10 | 5 | 7 | (A B C)

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

Step 4
Finish [i] = true for $0 \le i \le n$
Hence the system is in Safe state

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

Hence, the new system is in safe State, so Request can be granted immediately for process **P1**

d. What will happen if process P4 requests <3, 3, 0 > for extra resources?

i. Request$_i$ <= Need$_i$ **[<3,3,0> <= <4, 3, 1>] Goto Step 2**

ii. Request$_i$ <= Available$_i$ **[<3,3,0> <= <3, 3, 2>] Goto Step 2**

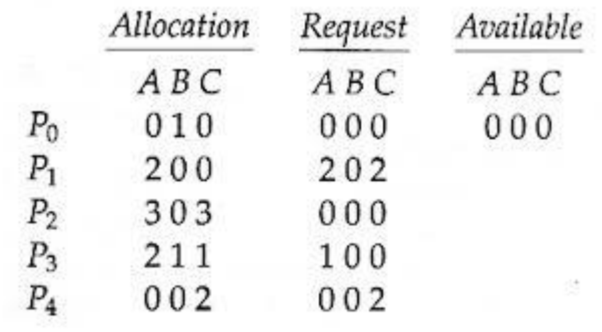
iii.

```
A. Available = Available - Request(i)

B. Allocation(i) = Allocation(i) + Request(i)

C. Need(i) = Need(i) - Request(i)
```

Example: For the Following snapshot :



	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

a. What is the Maximum Need matrix?

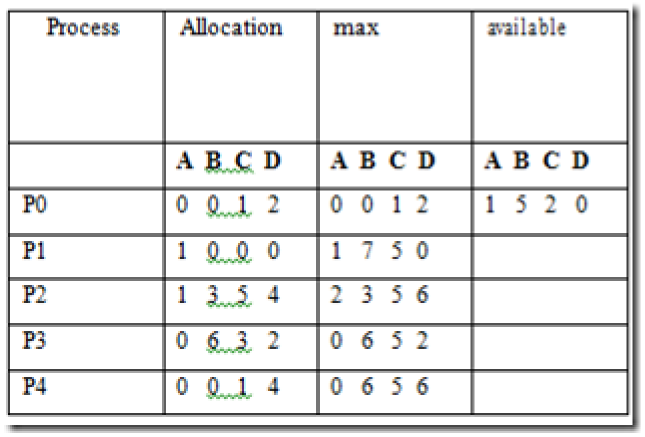
b. Is Safe sequence exist? What is the safe sequence?

c. Suppose P2 make an additional request for an instance C. Will the extra request granted immediately? P2→ <0,0,1>

Ans- a:

	A	**B**	**C**
P0	0	1	0
P1	4	0	2
P2	3	0	3
P3	3	1	1
P4	0	0	4

Question 3:



Process	Allocation	max	available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

a. What is the Need matrix?

b. Is Safe sequence exist? What is the safe sequence?

c. Suppose P1 make an additional request for <0,4,2,0>. Can the extra request granted immediately?

What is a Live lock?

There is a variant of deadlock called livelock. This is a situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work. This is similar to deadlock in that no progress is made but differs in that neither process is blocked or waiting for anything.
A human example of livelock would be two people who meet face-to-face in a corridor and each moves aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time.

Handling Deadlock
The above points focus on preventing deadlocks. But what to do once a deadlock has occured. Following three strategies can be used to remove deadlock after its occurrence.

1. **Preemption**

   We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.

2. **Rollback**

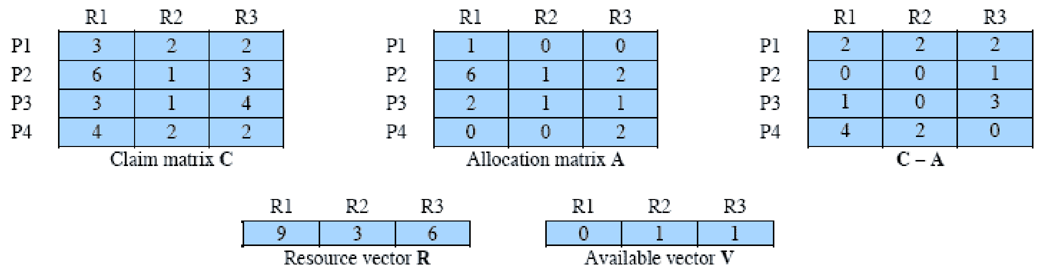
In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

3. **Kill one or more processes**

   This is the simplest way, but it works.

PROBLEMS:

1. **Determination Safe State using Banker's Algorithm**
   a. A system consisting of four processes and three resources.
   b. Allocations are made to processors
   c. *Is this a safe state?*



	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

(a) Initial state

Hint: Claim Matrix (Maximum Need), Resource Vector R (Total Number of Instances of Resource type)