Deadlock

- **Basic Terminology & Definitions**
  Deadlock, livelock, starvation, resource allocation graph
- **Conditions To Deadlock, Approaches To Deadlock**
  Mutual exclusion, hold and wait, no preemption, circular wait, deadlock prevention, avoidance, detection and recovery, Ostrich Algorithm
- **Deadlock Prevention**
  Solutions and demerits of the solutions for the conditions Mutual exclusion, hold and wait, no preemption, circular wait
- **Deadlock Avoidance**
  Process initiation denial, resource allocation denial, banker's algorithm, safe-unsafe state
- **Deadlock Detection And Recovery**
  Detection algorithm, resource category, solutions to every category, integrated deadlock strategy
- **Dining philosophers problem**
- **Solved problems**
  Problems on process initiation denial, resource allocation denial, detection algorithm, determine if the states are safe or unsafe.
- **Multiple Choice Questions**

Basic concepts and Terminologies

Q. Define Deadlock, resource, system state, safe state, unsafe state, consumable nonconsumable resources, livelock,
Ans.
Deadlock: Deadlock is defined as a situation wherein a set of two or more processes are waiting for the resources which are currently held by other members, which in turn are waiting for the resources in possession of the some other processes in the same set, thus none of them can proceed further.

In other words, the deadlock occurs if two processes need the same two resources to continue and each has ownership of one. Unless some action is taken, each process will wait indefinitely for the missing resource. The deadlock leads to permanent blockage of all the deadlocked processes.

Ans. Resource: A **resource**, or **system resource**, is any physical or virtual component of limited availability within a computer system. Every device connected to a computer system is a resource.

The basic resources of computing machine are: CPU, memory, files and I/O devices. Every internal system component is a resource. Virtual system resources include files, network connections and memory areas.

Ans. System State: The state of the system at any point of time is defined as the allocation of resources to processes at that particular instance.

Ans. Safe State: the system state in that does not lead to deadlock is termed as safe state/

Ans. Unsafe state: The system state that leads to deadlock is called unsafe state.

Ans. Reusable and consumable resources:

non-consumable resources: resources that are made available again after use.

Examples of reusable resources are:

- processors,
- I/O channels,
- Main and secondary memory, disk space
- Devices, and
- Data structures such as files, databases, and semaphores.
- Glasses, space in street!

Consumable resources: one task creates the resource, another can use it once only.

Examples of consumable resources are:

- Interrupts,
- Signals,
- Messages, and
- Information in I/O buffers.
- Spoken words

The resources in any computing system are broadly classified into two categories as, reusable and consumable resources. The consumable resources are created and destroyed. Also there is no limit on consumable resources of a particular type.

Ans. live lock : Live lock can be defined as a condition in which one 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 nor waiting for anything.

The system can come out of livelock if their relative pace of execution is changed while in execution.

Q. List three examples of deadlocks that are not related to a computer system environment.

Ans. Deadlock is not only associated with computer systems, but they are possible with general life scenarios, too. Some such examples are listed below.

- A scenario when two cars are crossing a single-lane bridge from opposite directions.
- A scenario where a person is going down a ladder while another person is climbing up the ladder at the same time.
- A scenario when two trains are traveling toward each other on the same track.

2. Process resource representation methodologies

Q. Comment on the resource state modeling methods.
Ans. Operating systems needs to analyze the process-resource states to determine if it is in deadlocked state or not. Two kinds of models are used to represent the resource allocation state of system as : **Graph model** and **Matrix model**.
Graph model:

- This model can depict the allocation state of a restricted class of system in which any process can request and use exactly one resource unit of each resource class.
- Uses a simple graph algorithm to determine if the circular wait condition is attained by the processes.

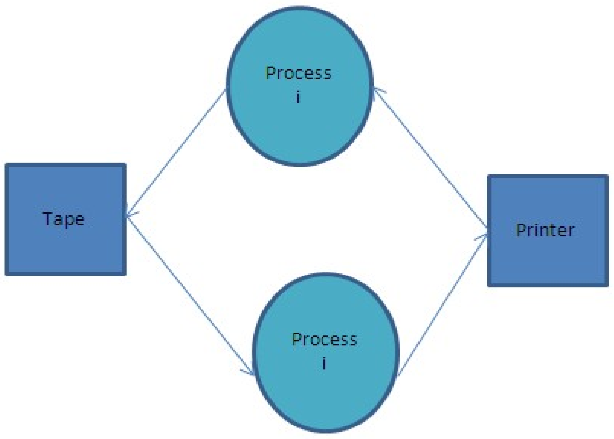


Figure: Resource allocation graph for two process and two resources.

Matrix model:

- This model has the advantage of generality.
- It can model allocation state in a system that permits a process to request and acquire any number of units of resource class and instances.

$$\begin{array}{c|ccccc|}
 & R1 & R2 & \dots & \dots & Rm \\
\hline
P1 & A_{11} & A_{12} & \dots & \dots & A_{1m} \\
P2 & A_{21} & A_{22} & \dots & \dots & A_{2m} \\
\dots & \dots & \dots & \dots & \dots & \dots \\
\dots & \dots & \dots & \dots & \dots & \dots \\
Pn & A_{n1} & A_{n2} & \dots & \dots & A_{nm} \\
\hline
\end{array}$$

Allocation Matrix of n resources for m types of resource classes

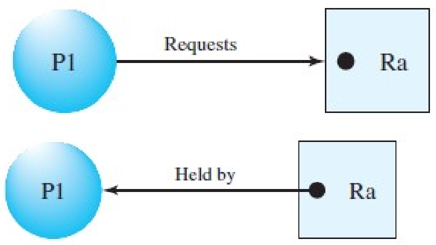
Q. Explain the concept of resource allocation graphs in detail.
Ans. Resource allocation graphs
The resource allocation to processes is the root cause for the deadlocks to occur. So, the deadlock management requires some tools to depict the resource allocation scenario at any instance. Resource allocation graphs characterize the exactly the same thing.

The resource allocation graph is basically a digraph in which each process and resource is represented by a node. Within a resource node, a dot is shown for each instance of that resource. All the directed edges from process to resource indicate the resources that have been requested but not yet granted. A graph edge directed from a reusable resource node dot to a process indicates a request that has been granted; that is, the process has been assigned one unit of that resource. A graph edge directed from a consumable resource node dot to a process indicates that the process is the producer of that resource.

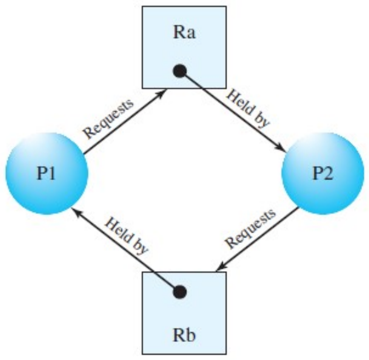
Resource Graph Model:
The resource allocation graphs(RAG) contain two kinds of nodes: circles and squares. Circles represent processes and squares are used to model resources.

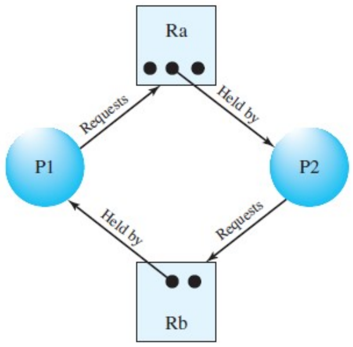
Process Resources

The RAGs represent two kinds of actions:
a. Process needs a resource
b. Resource is allocated to a process



a. A requested resource b. An allocated resource





c. Circular wait d. No deadlock

Why the system resources and their allotment to processes are mostly represented with matrix rather than with resource allocation graphs?

Ans. The system has multiple resources. Also there is no control over number of processes those can exist in system. These processes request multiple copies of each resource which becomes difficult to model with resource allocation graph. So, the matrix representation wins over the same.

$$\begin{array}{c|ccccc} & R1 & R2 & \dots & \dots & Rm \\ \hline P1 & C_{11} & C_{12} & \dots & \dots & C_{1m} \\ P2 & C_{21} & C_{22} & \dots & \dots & C_{2m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ Pn & C_{n1} & C_{n2} & \dots & \dots & C_{nm} \end{array}$$

Figure: Resource Claimed Matrix for n Processes and m Types Of Resources

$$\begin{array}{c|ccccc} & R1 & R2 & \dots & \dots & Rm \\ \hline P1 & A_{11} & A_{12} & \dots & \dots & A_{1m} \\ P2 & A_{21} & A_{22} & \dots & \dots & A_{2m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ Pn & A_{n1} & A_{n2} & \dots & \dots & A_{nm} \end{array}$$

Figure: Resource Allocation Matrix for n Processes and m Types Of Resources

3. Conditions to Deadlock, Approaches to Deadlock

Q. State the conditions for deadlock.

A deadlock has four potential reasons to occur. Out of them, the first three do not guarantee a deadlock, but when any one of them is combined with the fourth one, the deadlock occurs for sure. These four conditions can be listed as: Mutual Exclusion, Hold and wait, No preemption and Circular wait.

- Mutual exclusion:
- Hold and wait
- No preemption
- Circular wait

Q. What are the strategies to handle deadlock?
Ans. The different strategies to handle deadlock are: deadlock prevention, deadlock avoidance and deadlock detection and recovery.
Deadlock prevention: This is name of the design policy of operating system to get rid of deadlocks. It suggests how resource requests are made and how they are serviced.

Deadlock prevention is constraing how resource request can be submitted and how they can be handled by system. The foremost goal is to ensure that deadlock occurring conditions do not hold.
Deadlock Avoidance: This concept allows checks the possibility of deadlock on dynamic basis and then decides if resource granting will be safe or not.

The deadlock avoidance considers every resource request in run time and decides if it should grant the resources. This concept also requires all the processes to submit their total resource requirements in advance. Deadlock avoidance method allows more concurrency.
Deadlock detection and recovery: The detection technique checks if the system is in deadlock. If the answer is a "yes", then gives solution to recover from the same. This concept has nothing to do with resource granting. The recovery method depends on process and resource characteristics those have contributed to deadlock.

4. Deadlock prevention policy

Q. What is deadlock prevention policy of OS?
Ans. The deadlock prevention policy is to design system in such a way that excludes all the possibilities of deadlock occurrences. These prevention methods can be categorized into two categories: indirect method and direct method.
The indirect method tries to prevent the occurrence of one of the conditions i.e. mutual exclusion, hold and wait and no preemption. The direct method directly works on the circular wait to get prevented.

Mutual Exclusion prevention policy: Actually, this particular condition cannot be disallowed. If any resource requires mutual access then mutual exclusion must be supported by OS.

Hold and Wait prevention policy: The hold and wait condition can be prevented by requiring that a process should request all its required resources at once and blocking the process until all resources can be granted simultaneously. This approach takes much time for a process to get started and thus results in more response time, waiting time and obviously turnaround time. Also, this approach keeps most of the resources with the process for its lifetime and they may get underutilized.

No preemption prevention policy: This situation can be handled in many ways. One approach is, when a process holding certain resources and is denied further ones, then the process must release its original resources. This process when scheduled next time, can request for these resources afresh. Another approach is, when a process holding certain resources and requests for some more ones which are in possession with other process, the OS must preempt second process and make it to release all its possessions.
This method is useful only when it is applied to resources whose state can be easily saved and restored later on.

Circular wait prevention policy: In this case, all the resources in system are listed and are given some liner order numbers. If a process is allocated resources of type R, then it may subsequently request only those resources of types following R in the ordering.
i.e. Then resource Ri precedes Rj in the ordering if $i < j$. Now suppose that two processes, A and B, are deadlocked because A has acquired Ri and requested Rj, and B has acquired Rj and requested Ri. This condition would be impossible here because it implies $i < j$ and $j < i$.

This approach to deal with circular-wait prevention may be inefficient, slowing down processes and denying resource access unnecessarily.

These prevention policies leads to inefficient use of resources and overall inefficient execution of processes.

5. Deadlock avoidance policy

Q. What is deadlock avoidance policy?

Ans: **Deadlock** avoidance allows the first three conditions but ensures that deadlock point is never checked. With avoidance, a decision is dynamically made whether the next resource allocation, if granted, can lead to deadlock. This decision is based on two policies as, **Process Initiation Denial** and **Resource Allocation Denial.**

A. Process Initiation Denial: Do not start a process if its demands might lead to deadlock.

- This approach enlists all system resources in resource vector R.
- The processes involved state all their resource requirements beforehand.
- With every resource allocation resource available vector A is computed, which initially equals the resource vector R.
- The process requirements if are less than or equal to as mentioned in available vector A, then the request is granted and process is initiated.
- If the process requirements are greater than those mentioned in available vector A, then the process is denied its initiation.

This strategy is hardly optimal, because it assumes the worst: all processes will make their maximum claims together.

B. Resource allocation denial: Do not grant an incremental resource request to a process if it might lead to deadlock.

This approach, also called as Banker's algorithm, always try to keep the system is safe state. The state concept is defined with reference to a system of fixed number of resources and a fixed number of processes as follows,

State: the State of the system depicts the current allocation of resources to the processes.

Safe state: Safe state is characterized as there exist at least one sequence of resource allocation that does not result in deadlock.

Unsafe state: the state which is not safe, i.e. which does not assure even one sequence that doesn't end up in deadlock.

Here, there is no restriction on process initiation. Instead, processes are allowed to get created. All the processes give their overall resource requirements and their requirements in initial state. At every resource allocation request, the OS evaluates a hypothetical situation 'what if the required resources are granted, the process in question completes and returns back the possessions, will there be at least one sequence in which all the processes in set can get executed?" i.e. if the resource granting will result in a safe state? If yes, the allocation is completed otherwise it gets denied.

Q. Explain the process denial policy of deadlock avoidance in detail.

Process Initiation Denial is one the deadlock prevention policies. It is based on the assumption that any process requesting its initiation should claim all of its resources at the same time.
To implement this policy, the system maintains two vectors and two matrices as,

Vector R = all resources indicating various resource types and their instances.

$$R = (R_1, R_2, \ldots, R_m)$$

Vector V = the types and instances of available resources at any instance of time.

$$V = (V_1, V_2, \ldots, V_m)$$

Matrix C = resources claimed by processes.(C_{ij} = requirement of process i for resource j, with one row dedicated to each process)

$$\begin{array}{c|ccccc} & R1 & R2 & \ldots & \ldots & Rm \\ \hline P1 & C_{11} & C_{12} & \ldots & \ldots & C_{1m} \\ P2 & C_{21} & C_{22} & \ldots & \ldots & C_{2m} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ Pn & C_{n1} & C_{n2} & \ldots & \ldots & C_{nm} \end{array}$$

Claimed Matrix C

Matrix A= a matrix of resources allocated to processes. (A_{ij}= current allocation to process i of resource j)

$$\begin{array}{c|ccccc} & R1 & R2 & \ldots & \ldots & Rm \\ \hline P1 & A_{11} & A_{12} & \ldots & \ldots & A_{1m} \\ P2 & A_{21} & A_{22} & \ldots & \ldots & A_{2m} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ Pn & A_{n1} & A_{n2} & \ldots & \ldots & A_{nm} \end{array}$$

Allocation Matrix A

All these above data structures are bound in following relationships,

1. All resources are either allocated or available.

   $$R_j = V_j + \sum_{i=1 \text{ to } n} A_{ij} \text{ for all } j$$

2. No process can claim more than the total amount of resources in the system.

   $$C_{ij} <= R_j \text{ for all } i, j$$

3. No process is allocated more resources of any type than the process originally claimed to need.

   $$A_{ij} <= C_{ij} \text{ for all } I,j$$

When allowing any (n+1)th process to initiate while already n (n>=0) processes exist in system , the following condition is checked.

$$\boxed{R_{ij} >= C_{(n+1)j} + \sum_{i=1 \text{ to } n} C_{ij} \text{ for all } j}$$

That means, a process is initiated only if the maximum claims so far plus the new request can be met. This approach is hardly optimal as its based on the worst possible assumption: all processes will make their maximum claims together.

Q. Give an example of process initiation denial approach of deadlock management.
Ans.
Consider a system with given resource vector R=

6	5	9

Assume process P1,P2,P3,P4 join the system with some claims for resources in the same order as they are written.

C=

4	3	3
7	2	4
2	2	5
0	0	1

Initially the available vector is same as the resource vector.

So V=

7	5	9

Every process received is checked against the available vector one by one in their order of arrival.
.

Process	Request	Available resources Vk=Vk-Cik	Is Request < Available?	Process initiated?
P1	{4,3,3}	{7,5,9}	Yes	Yes
P2	{7,2,4}	{3,2,6}	No	No
P3	{2,2,5}	{3,2,6}	Yes	Yes
P4	{0,0,1}	{1,0,1}	Yes	Yes

Thus in the above example only P1,P3, and P4 are allowed to initiate while P2 is denied its initiation.

Q. Explain in detail the resource allocation denial policy of deadlock avoidance.
The deadlock possibility can be avoided by not awarding the resources whenever they may have potential of leading to deadlock. This strategy is most popularly known as **Banker's algorithm**. For a system of fixed number of resources and a fixed number of processes, the OS defines state as the current resource allocation to processes. This way, a safe state is considered the one in which there exists at least one sequence of process execution. Thus, any allocation leading to unsafe state is avoided.

Data structures used in computation:
Vector R = all resources indicating various resource types and their instances.

$R= (R_1,R_2, . . . , R_m)$

Vector V = the types and instances of available resources at any instance of time.

$V= (V_1,V_2, . . . ,V_m)$

Matrix C = resources claimed by processes.($C_{ij =}$ requirement of process i for resource j, with one row dedicated to each process)

C=

C_{11}	C_{12}	…	…	C_{1m}
C_{21}	C_{22}	…	…	C_{2m}
…	…	…	…	…
…	…	…	…	…
C_{n1}	C_{n2}	…	…	C_{nm}

Matrix A= a matrix of resources allocated to processes. ($A_{ij=}$ current allocation to process i of resource j)

A=

A_{11}	A_{12}	…	…	A_{1m}
A_{21}	A_{22}	…	…	A_{2m}
…	…	…	…	…
…	…	…	…	…
A_{n1}	A_{n2}	…	…	A_{nm}

Matrix M = a matrix of more resources required which is computed as,

M = C-A

Given all the above values the stafe status can be calculated as,

1. For any process i=1 to n, compare its associated row of matrix M with vector V.
2. If that Mij is less than or equal to Aj for all j, this process i can be assumed to be completed and thereby its resources can be given back to system.
   Thus, after hypothetical successful completion of process i the available vector A is updated as,
   V= V+ Aij where Aij indicates the resources j allocated earlier to process i.
   Also update Aij=0 for all js as the process holds no more resources.
3. Repeat step 1 and 2 until all processes are considered.

If all the processes in the set have the allocation matrix entries as 0s, the system is in safe sate, and sequence in which the process were selected can be termed as one of the possible execution sequence without any deadlock.

This approach seems simple and good, but it has certain preconditions X as,

1. All the resource requirements must be stated in advance.
2. The order in which processes get executed should not be constrained by synchronization or any that kind of mechanism.
3. The system must have fixed number of resources to allocate
4. The processes participating here are not allowed to exit holding the resources

Banker's Algorithm makes sure that only processes that will run to completion are scheduled to run. However, if there are deadlocked processes, the will remain deadlocked. **Banker's Algorithm does not eliminate a deadlock.**

Q. Explain the resource allocation denial concept with an example.
Ans. Resource allocation denial or Banker's algorithm
Example: consider a set of four processes p1,p2,p3,p4 and three resource types as R1,R2 and R3 with their values as,

Resource vector R(R1,R2,R3)=

13	5	9

Note: In the following matrices rows indicate process and columns represent the resources.

Claimed matrix C =

4	3	3
7	2	4
4	2	5
5	3	3

Allocation Matrix A =

2	0	0
7	2	3
3	2	2
1	1	3

More requirement M =

2	2	2
0	0	1
1	0	3
4	2	0

Available vector V (R1,R2,R3)=

0	0	1

Solution: From the given processes P1,P2,P3 and p4, only p2 is the process whose corresponding row in matrix M satisfies the criteria of being less than values in vector V.
So, **first P2 may run to completion.** Then the above tables will have values as,

1. **After P2 runs to completion**

4	3	3
0	0	0
4	2	5
5	3	3

Claimed Matrix C

2	0	0
0	0	0
3	2	2
1	1	3

Allocation Matrix
A

2	2	2

0	0	0
1	0	3
4	2	0

7	2	4

Available vector V

More Requirements M

Now, seeing the entries in M and available vector V, **any process** can be awarded with the resources and they can run to completion.

1. After P1 runs to completion

0	0	0
0	0	0
4	2	5
5	3	3

Claimed Matrix C

0	0	0
0	0	0
3	2	2
1	1	3

Allocation Matrix

A

Available vector V=

9	2	4

More Requirements M=

0	0	0
0	0	0
1	0	3
4	2	0

2. After P3 runs to completion

0	0	0
0	0	0
0	0	0
5	3	3

Claimed Matrix C

0	0	0
0	0	0
0	0	0
1	1	3

Allocation Matrix

A

0	0	0
0	0	0
0	0	0
4	2	0

Available vector V

12	4	6

More Requirements M

3. After P4 runs to completion

0	0	0
0	0	0
0	0	0
0	0	0

0	0	0
0	0	0
0	0	0
0	0	0

Claimed Matrix C Allocation Matrix A

0	0	0
0	0	0
0	0	0
0	0	0

More Requirements M

Available vector V

13	5	9

As the resultant computation is giving the process execution sequence as **P2-P1-P3-P4**, the system is in **Safe state.**

Q. Give one example of potential unsafe state that can be avoided with deadlock avoidance (Banker's algorithm).
Ans.

consider a set of four processes p1,p2,p3,p4 and three resource types as R1,R2 and R3 with their values as,

Resource vector R(R1,R2,R3)=

13	5	9

Note: In the following matrices rows indicate process and columns represent the resources.

If process P2 from earlier example requests one more resource of type R3, and if it gets granted by system, then it enters in unsafe state.

Claimed matrix C =

4	3	3
7	2	**5**
4	2	5
5	3	3

Allocation Matrix A =

2	0	0
7	2	**3**
3	2	2
1	1	3

More requirement M =

2	2	2
0	0	2
1	0	3
4	2	0

Available vector V (R1,R2, R3)=

0	0	1

In the above scenario, none of the row the More Requirement Matrix M has requirement that can be satisfied by available vector V. so, **this is an unsafe state.** To overcome this, the requests given in claimed matrix should be served in such a way that there exists one possible sequence of execution.

6. Deadlock Detection And Recovery

Q. How does deadlock detection and recovery work?

This is the most liberal policy amongst all. This approach does not limit resource access or restricts process actions. Instead, the requested resources are granted whenever possible and OS performs periodic or aperiodic checks to detect if deadlock has occurred and it is recovered accordingly.

The frequency of deadlock detection is one of the mentioned below:

- On some regular interval
- On each resource request

The first policy reduces OS overhead but may lead the system into a serious deadlock state while the later one results in early detection.

One the deadlock is detected, every kind of resource that has caused the deadlock, has a different deadlock recovery strategy.

Q. Explain in detail the deadlock detection algorithm.
Ans.

This particular approach has the least restrictions on the processes and resources. It lets the processes claim and get resources, all resources need not be claimed in beginning and system runs checks if system is in deadlock state and recovers from the same accordingly.

The detection and recovery algorithm:
Inputs:

1. Resource Vector R
2. Available vector V (optional)
3. Claimed matrix C
4. Allocation matrix A

Steps:

1. Compute the available vector V if not given.
2. Compute the more requirement matrix M= C-A
3. Mark each process that has a row in the Allocation matrix of all zeros.
4. Find an index I such that process i is not marked and its corresponding ith row of M is less than or equal to Available vector V.i.e., Mik <= Vk for all 1<=k<=m where m is total number of resources. If no such row is found, terminate the algorithm.
5. If such a row is found, mark process I and add the corresponding row of the allocation matrix to available vector V. i.e. Vk=Vk+Aik for all 1<=k<=m. return to step 4.

A deadlock is detected if there are any unmarked processes at the end of the algorithm. It says that every unmarked process is deadlocked. this algorithm does not guarantee to prevent deadlock; that will depend on the order in which future requests are granted. All that it does is it determines if deadlock currently exists accordingly calls the recovery procedure.

Q. What are the general recovery policies of system?
Ans: one of the following options can be taken whenever system detects that it is in a deadlock state.

1. Abort all deadlocked processes (one of the most common solution adopted in OS!!)
2. Rollback each deadlocked process to some previously defined checkpoint and restart them (original deadlock may reoccur)
3. Successively abort deadlock processes until deadlock no longer exists (each time we need to invoke the deadlock detection algorithm)\
4. Successively preempt some resources from processes and give them to other processes until deadlock no longer exists (a process that has a resource preempted must be rolled back prior to its acquisition)

For the options offered in 3 &4, the criteria to choose a process is one or more of the following ones.

- least amount of CPU time consumed so far
- least total resources allocated so far
- least amount of "work" produced so far...

The following table gives example of three types of the resources device, file and memory being requested, acquired by a process and released by the same.

Request a resource	**Acquire/use a resource**	**Release a resource**
Request a device	Read from/write to a device	Release a device
Open a file	Read/write a file	Close a file
allocate memory	Use the memory	Free memory

Q. what is integrated deadlock strategy?

There are several ways to address the problem of deadlock in an operating system.

- Just ignore it and hope it doesn't happen
- Detection and recovery - if it happens, take action
- Dynamic avoidance by careful resource allocation. Check to see if a resource can be granted, and if granting it will cause deadlock, don't grant it.
- Prevention - change the rules

Actually, every deadlock handling strategy has some merits and demerits associated with them. So instead of employing any one the strategies, it's better to follow different strategy in different situations and with different kinds of resources.

The general integrated deadlock strategy is:

1. Group the resources into number of different resource categories.
2. Use the linear ordering strategy defined for deadlock prevention of circular wait to prevent deadlocks between resource classes.
3. Within a resource class, use the algorithm that is most appropriate for that class.

Some example strategies within a resource class are as listed below,

1. Resource type: **Swappable space**
   Recovery strategy: **Deadlock prevention** as the one used with hold-and-wait prevention solution and **deadlock avoidance**
2. Resource type: **Process resources**
   Recovery strategy**: deadlock avoidance** with resource allocation denial solution type and **Deadlock prevention** by means of resource ordering within the resource class.
3. Resource type: **Main memory**
   Recovery strategy: **Deadlock prevention** solution used for no-preemption situation.
4. Resource type: **Internal resources**
   Recovery strategy: **Deadlock prevention** by means of resource ordering can be used.

7. Deadlocks in UNIX

Q. Comment on deadlocks in Unix.

Like any other OS, Unix also employs multiple policies to handle the deadlock issue. They can be enumerated as,

- Ostrich approach-It simply ignores the possibility of deadlocks involving user processes.
- Deadlock prevention through resource ordering strategy is used for the processes that are executing the kernel code as a result of interrupt or system calls. Data structures in kernel are locked and released in a standard order. However, not all kernel functionalities can lock the data structures so deadlocks are possible.
- One approach to deal with issue above is, the process issuing request avoids getting blocked on its lock. Instead it tries to get the next resource of the same type if possible. This strategy avoids deadlocks by avoiding circular waits, especially with buffer memory allocations.
- For the processes working with file systems, in case they need access to two different directories, both directory locks are not set at the same time. It first locks the first directory, updates it in desired manner and releases the lock and then accesses another directory and updates it as well. Thus it gets only one lock at a time thus avoiding hold and wait condition.

Dining philosophers problem

Q. Dining philosopher's problem

Ans. The dining philosophers problem is a ``classical'' synchronization problem. Taken at face value, it is a pretty meaningless problem, but it is typical of many synchronization problems that one can see when allocating resources in operating systems.

Problem definition: There are 5 philosophers sitting at a round table. Between each adjacent pair of philosophers is a fork. So they have only five fork. Each philosopher does two things: think and eat. The philosopher thinks for a while, and then stops thinking and becomes hungry. When the philosopher becomes hungry, she cannot eat until she owns the both the forks on left and right. When the philosopher is done eating she puts down the forks and begins thinking again.

The challenge in the dining philosophers problem is to design a protocol so that the philosophers do not deadlock (i.e. every philosopher has a fork), and so that no philosopher starves (i.e. when a philosopher is hungry, she eventually gets the chopsticks). Additionally, our protocol should try to be as efficient as possible -- in other words, one should try to minimize the time that philosophers spent waiting to eat.

- Solution 1: Resource ordering

This approach establishes the convention that all resources will be requested in order, and released in reverse order, and that no two resources unrelated by order will ever be used by a single unit of work at the same time. Here, the resources are the forks. They are numbered one through five. All the philosopher's have to pick up the lower numbered fork first and then the higher numbered one. Then, she will always put down the higher numbered fork first, followed by the lower numbered fork. In this case, if four of the five philosophers simultaneously pick up their lower-numbered fork, only the highest numbered fork will remain on the table, so the fifth philosopher will not be able to pick up any fork. Moreover, only one philosopher will have access to that highest-numbered fork, so he will be able to eat using two forks. When he finishes using the forks, he will put down the highest-numbered fork first, followed by the lower-numbered fork, freeing another philosopher to grab the latter and begin eating.

This solution to the problem is the one originally proposed by Dijkstra. And it is not always practical, especially when the list of required resources is not completely known in advance.

- Solution 2: monitor

Philosophers can eat if neither of their neighbors is eating. This is comparable to a system where philosophers that cannot get the second fork must put down the first fork before they try again. (no hold and wait condition)

This solution uses a single mutual exclusion lock. This lock is not associated with the forks but with the decision procedures that can change the states of the philosophers. This is ensured by the monitor. The procedures **test**, **pickup** and **putdown** are local to the monitor and share a mutual exclusion lock. The philosophers wanting to eat do not hold a fork. When the monitor allows a philosopher who wants to eat to continue, the philosopher will reacquire the first fork before picking up the now available second fork. When done eating, the philosopher will signal to the monitor that both forks are now available.

To also guarantee that no philosopher starves, one could keep track of the number of times a hungry philosopher cannot eat when his neighbors put down their forks. If this number exceeds some limit, the state of the philosopher could change to Starving, and the decision procedure to pick up forks could be augmented to require that none of the neighbors are starving.

A philosopher, who cannot pick up forks because a neighbor is starving, is effectively waiting for the neighbor's neighbor to finish eating. This additional dependency reduces concurrency. Raising the threshold for transition to the Starving state reduces this effect.

Multiple choice questions.

Q. Let S and Q be two semaphores initialized to 1, where P0 and P1 processes the following statements wait(S);wait(Q); ---; signal(S);signal(Q) and wait(Q); wait(S);---;signal(Q);signal(S); respectively. The above situation depicts a _________ .
1 Semaphore
2 Deadlock
3 Signal
4 Interrupt
Right Ans) 2
Q. _________ is the situation in which a process is waiting on another process, which is also waiting on another process , which is waiting on the first process. None of the processes involved in this circular wait are making progress.
1 Deadlock
2 Starvation
3 Dormant
4 None of the above
Right Ans) 1

Q. The Banker's algorithm is used
1 to prevent deadlock in operating systems
2 to detect deadlock in operating systems
3 to rectify a deadlocked state
4 none of the above
Right Ans) 1

Q. In one of the deadlock prevention methods, impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration. This overcomes the _______________ condition of deadlock
1 Mutual exclusion
2 Hold and Wait
3 Circular Wait
4 No Preemption
Right Ans) 3

Q. Resource locking ________.
1 Allows multiple tasks to simultaneously use resource
2 Forces only one task to use any resource at any time
3 Can easily cause a dead lock condition
4 Is not used for disk drives
Right Ans) 2

Q. A set of resources' allocations such that the system can allocate resources to each process in some order, and still avoid a deadlock is called ________.
1 Unsafe state

2 Safe state
3 Starvation
4 Greeedy allocation
Right Ans) 2

Q. A process said to be in ___________ state if it was waiting for an event that will never occur.
1 Safe
2 Unsafe
3 Starvation
4 Dead lock
Ans) 4

Q. Consider a system with m resources of same type being shared by n processes. Resources can be requested and released by processes only on at a
time. The system is deadlock free if and only if
1. The sum of all max needs is < m+n
2. The sum of all max needs is > m+n
3. Both of above
4. None

Ans: 1

Q. Consider a system consisting of 4 resources of same type that are share by 3 processes each of which needs at most two resources. Show that the system is deadlock free
Ans:
If the system is deadlocked, it implies that each process is holding one resource and is waiting for one more. Since there are 3 processes and 4 resources, one process must be able to obtain two resources. This process requires no more resources and therefore it will return its resources when done.

Q. A system has four processes P1 through P4 and two resource types R1 and R2. It has 2 units of R1 and 3 units of R2.
Given that: P1 requests 2 units of R2 and 1 unit of R1, P2 holds 2 units of R1 and 1 unit of R2, P3 holds 1 unit of R2, P4 requests 1 unit of R1
Show the resource graph for this state of the system. Is the system in deadlock, and if so, which processes are involved?

	R1	R2
P1	1	2
P2	0	0
P3	0	0

Ans:

P4	1	0

C

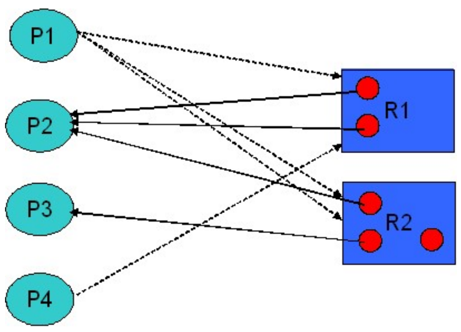
Given:

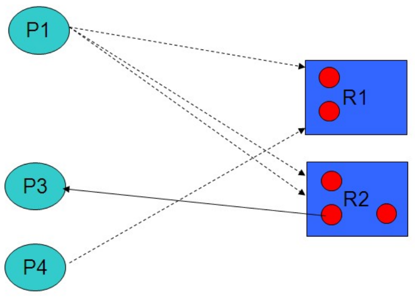
R1	R2
2	3

R

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

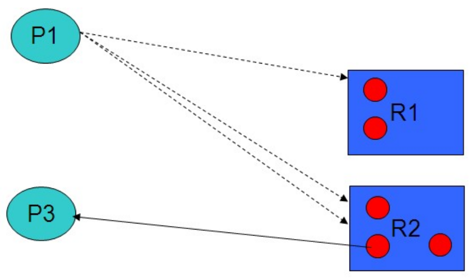
A

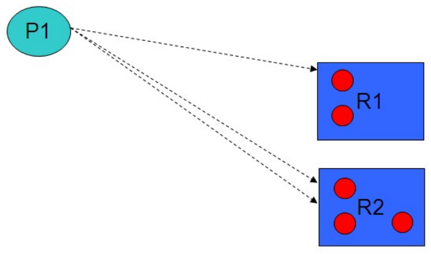




Step 1: Initial stage
(dashed lines indicate requests)

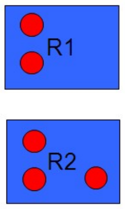
Step 2: Process P2 completes





Step 3: Process P4 completes

Step 4: Process P3 completes



Step 5: Process P1 completes

There are many sequences in which the processes could be realize, but process P2 must complete before process P1 as it hold the resources requested by former.

Q. Given 5 total units of the resource of the same type, tell whether the following system is in a safe or unsafe state.

R=5,

	P1	P2	P3	P4
C=	2	3	4	5

	P1	P2	P3	P4
A=	1	1	2	0

Ans. safe state

Q. Given a total of 5 units of resource type 1 and 4 units of resource type 2, tell whether the following system is in a safe or unsafe state. Show your work.

	R1	R2
P1	2	3
P2	3	2
P3	4	4

Claimed Matrix C

	R1	R2
P1	1	1
P2	1	1
P3	2	1

Allocation Matrix A

Ans: unsafe

Process	**Allocation**	**Claimed**	**Available**

Q. Given a total of 10 units of a resource type, and given the safe state shown below, should process 2 be granted a request of 2 additional resources? Show your work.

	P1	P2	P3	P4	P5
C=	5	6	6	2	4

	P1	P2	P3	P4	P5
A=	2	1	2	1	1

Ans. Unsafe state

Q.
Consider the snapshot of a system.

P0	{0,1,0}	{7,5,3}	{3,3,2}
P1	{2,0,0}	{3,2,2}	
P2	{3,0,2}	{9,0,2}	
P3	{2,1,1}	{2,2,2}	
P4	{0,0,2}	{4,3,3}	

Answer the following question using the banker's Algorithm
a. What is the content of matrix need?
b. Is the system in safe state?
c. If a request from process P1 arrives for (1,0,2) can the request be granted immediately?

Q. Consider a system with three processes and three types of resources with the following data.

$$R = \begin{array}{ccc} A & B & C \\ \hline 3 & 2 & 1 \end{array}$$

Resource vector

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

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

Claim

Matrix Allocation Matrix

Run the deadlock detection algorithm on the above example and check is the system in a deadlock? If yes, then what are the process(es) that need to be pre-empted and in what order, to ensure that deadlock is overcome?