Unit – IV

Deadlocks and Mass-storage structure

Objectives:

- Students will be able to know the problems of deadlock and study the various avoidance mechanisms

Syllabus: Deadlocks and Mass-storage structure

Deadlocks-

- System model
- Deadlock characterization: Necessary conditions, Resource-Allocation Graph
- Methods for handling deadlocks:
  - deadlock- prevention
  - Avoidance: Safe state, Resource-Allocation-Graph, Banker's Algorithm
  - Detection: single instance of each resource type, several instances of a resource type
  - Recovery
    - process termination
    - resource pre-emption

Mass-storage structure- Overview (Magnetic disks, Magnetic tapes), Disk Scheduling (FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK Scheduling), Disk Management (Disk Formatting, Boot blocks, Bad blocks).

Outcomes:

Students will be able to

- Develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks
- Present a number of different methods for preventing or avoiding deadlocks in a computer system.

Learning Material

4.1. System Model:

Definition of Deadlock:

- In a multiprogramming environment several processors may compete for a finite number of resources.
- A process requests resources and are not available at that time.
- So the process enters into waiting state sometimes the waiting process can never change its state.
- This situation is called a deadlock.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

- **Request:**
  - The process requests the resource.
  - If the request cannot be granted immediately. Then the requesting process must wait until it can acquire the resource.
- **Use:**
  - The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- **Release:** The process releases the resource.
- The resources may be either physical resources (for example, printers, tape drives, memory space, and CPU cycles) or logical resources (for example, files, semaphores, and monitors).

Deadlock Examples:

Example 1:

- Consider a system with three CD RW drives.
- Suppose each of three processes holds one of these CD RW drives.
- If each process now requests another drive, the three processes will be in a deadlocked state.

Example 2:

- Consider a system with one printer and one DVD drive.
- Suppose that process P_i is holding the DVD and process P_j is holding the printer.
- If P_i requests the printer and P_j requests the DVD drive, a deadlock occurs.

4.2. Deadlock Characterization

4.2.1. Necessary Conditions:

There are four conditions that are necessary for the occurrence of a deadlock:

1. **Mutual Exclusion:**
   - At least one resource must be held in a non-sharable mode;
   - If any other process requests this resource, then that process must wait for the resource to be released.
2. **Hold and Wait:**
   - A process must be simultaneously holding at least one resource and waiting for at least one resource that is currently being held by some other process.
3. **No preemption:**
   - Once a process is holding a resource (i.e. once its request has been granted), then that resource cannot be taken away from that process until the process voluntarily releases it.
4. **Circular Wait:**
   - A set of processes { P0, P1, P2, . . ., PN } must exist such that every P[i] is waiting for P[(i + 1) % (N + 1)].

4.2.2. Resource-Allocation Graph:

- In some cases deadlocks can be understood more clearly through the use of Resource Allocation Graphs.
- RAG contains the following properties:
  - A set of resource categories, { R1, R2, R3, . . ., RN }, which appear as square nodes on the graph.

- Dots inside the resource nodes indicate specific instances of the resource. (E.g. two dots might represent two laser printers.)
- A set of processes, { P1, P2, P3, . . ., PN }
- **Request Edge:**
  - A set of directed arcs from Pi to Rj, indicating that process Pi has requested Rj, and is currently waiting for that resource to become available.
- **Assignment Edge:**
  - A set of directed arcs from Rj to Pi indicating that resource Rj has been allocated to process Pi, and that Pi is currently holding resource Rj.
  - Note that a request edge can be converted into an assignment edge by reversing the direction of the arc when the request is granted.

Note:

- If a resource allocation graph contains no cycles, then the system is not deadlocked.
- If a resource allocation graph does contain **cycle**s AND each resource category contains only a **single instance**, then a **deadlock exists**.
- If a resource category contains **more than one instance**, then the presence of a **cycle** in the resource allocation graph indicates the possibility of a **deadlock**, **but does not guarantee** one.

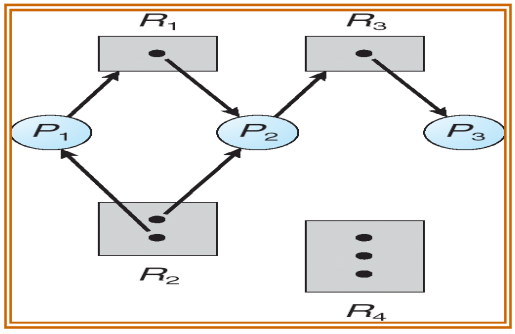


Fig: Resource allocation graph

- The content of above resource-allocation graph is represented as follows:
  - The sets *P,* R and *E:*
    - *P* == {P1, P2, P3}
    - R== {R1, R2, R3}
    - *E* == {P_1→ R_1, p_2→ R_3, R_1→ p_2, R_2→p_2, R_2→P_1, R_3→ P_3}
  - Resource instances:
    - One instance of resource type R_1
    - Two instances of resource type R_2
    - One instance of resource type R_3
    - Three instances of resource typeR_4
  - Process states:
    - Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1.
    - Process P_2 is holding an instance of R_1 and an instance of R_2 and is waiting for an instance of R_3.
    - Process P_3 is holding an instance of R_3 .

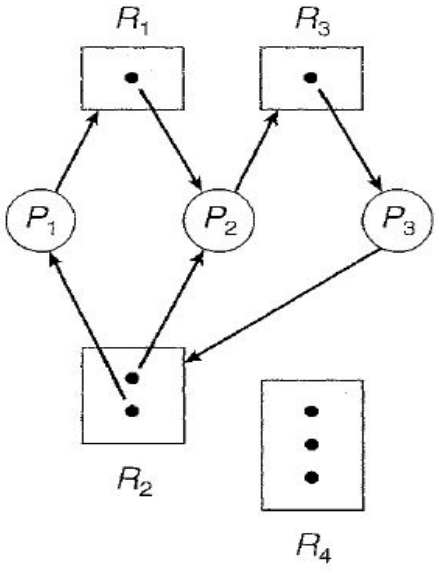


Fig: Resource allocation graph with a deadlock

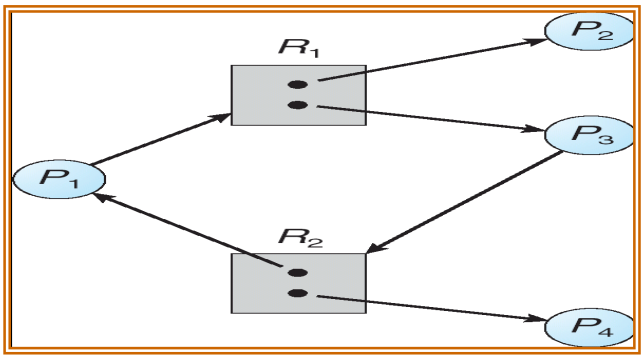


Fig: Resource allocation graph with a cycle but no deadlock

- In this example, we also have a cycle However, there is no deadlock.
- Observe that process *P4* may release its instance of resource type R2.
- That resource can then be allocated to P3, breaking the cycle.
    - *If a resource-allocation graph does not have a cycle, then the system is not in a deadlocked state. If there is a cycle, then the system may or may not be in a deadlocked state*

4.3. Methods for Handling Deadlocks

Generally, the deadlock problem can be handled in one of three ways:

- We can use a protocol to prevent or avoid deadlocks, ensuring that the system will *never* enter a deadlocked state.
- We can allow the system to enter a deadlocked state, detect it, and recover.
- We can ignore the problem altogether and pretend that deadlocks never occur in the system.

4.3.1. Deadlock- Prevention:

- By ensuring that at least one of these conditions cannot hold, we can *prevent* the occurrence of a deadlock.

4.3.1.1. Mutual Exclusion:

- The mutual-exclusion condition must hold for non-sharable resources. For ex: a printer cannot be simultaneously shared by several processes.
- Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock.

- Read-only files are a good example of a sharable resource. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file. A process never needs to wait for a sharable resource.
- We cannot prevent deadlocks by denying the mutual exclusion condition, because some resources are strictly sharable.

4.3.1.2. Hold and Wait

- To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.
  - **One protocol** that can be used requires each process to request and be allocated all its resources before it begins execution.
  - **An alternative protocol** allows a process to request resources only when it has none.
    - A process may request some resources and use them.
    - Before it can request any additional resources it must release all the resources that it is currently allocated.

Example:

- To illustrate the difference between these two protocols:
- We consider a process that copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer.

<u>According to Protocol 1:</u>

- If all resources must be requested at the beginning of the process, then the process must initially request the DVD drive, disk file, and printer.
- It will hold the printer for its entire execution, even though it needs the printer only at the end.

<u>According to Protocol 2:</u>

- The process to request initially only the DVD drive and disk file.
- It copies from the DVD drive to the disk and then releases both the DVD drive and the disk file.

- The process must then again request the disk file and the printer.
- After copying the disk file to the printer, it releases these two resources and terminates.

Both these protocols have two main disadvantages:

- First, resource utilization may be low, since resources may be allocated but unused for a long period.
- Second, starvation is possible.
  - A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

4.3.1.3. No Pre-emption

- Pre-emption of process resource allocations can prevent this condition of deadlocks, when it is possible.
- Approach 1:
  - If a process is forced to wait when requesting a new resource, then all other resources previously held by this process are implicitly released, forcing this process to reacquire the old resources along with the new resources in a single request, similar to the previous discussion.
- Approach 2:
  - When a resource is requested and it is not available, then the system looks to see what other processes currently have those resources *and* are blocked itself and waiting for some other resource.
  - If such a process is found, then some of their resources may get pre empted and added to the list of resources for which the process is waiting.
  - Either of these approaches may be applicable for resources whose states are easily saved and restored, such as registers and memory, but are generally not applicable to other devices such as printers and tape drives.

4.3.1.4. Circular Wait

- One way to avoid circular wait is to number all resources, and to require that processes request resources only in strictly increasing (or decreasing) order.
- In other words, in order to request resource Rj, a process must first release all Ri such that i >=j.
- One big challenge in this scheme is determining the relative ordering of the different resources.

4.3.2. Deadlock Avoidance:

- The most useful model requires that each process declare the maximum number of each type that it need.
- We can construct an algorithm that ensure the system will never enter into deadlock by giving the priori information about the maximum number of resources of each type that may requested for each process.
- A deadlock avoidance algorithm dynamically check the resource allocation state to ensure that the system never enter into a deadlock.
- The resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

4.3.2.1. Safe State

- A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock.
- A sequence of processes $<p_1,p_2,.....p_n>$ is a safe sequence for all the current allocation state if , for each p_i, the resource that pi can still request can be satisfied by the current available resource plus the resources held by all p_j, with j<i.
- In this situation the resources that process pi needs are not immediately available, and then p_i can wait until all p_j have finished,

p_i can obtain all of its needed resources, completed its designated task, and return it's all allocated resources, and terminates.

- When p_i terminates, p_{i+1} can obtain its needed resources, and so on. If no such sequence exists, then system state is said to be unsafe.

Fig: safe, unsafe, and deadlock state space

4.3.2.2. Resource-Allocation Graph Algorithm:

- In addition to request and assignment edges, the other edge is claim edge.

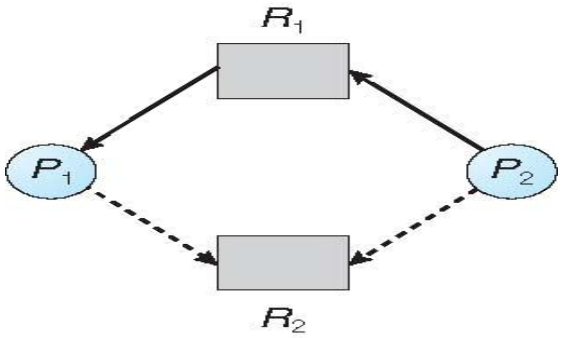


Fig: Resource-allocation graph for deadlock avoidance

- A claim edge P_i->R_j is represents that process pi may request R_j at some time in the future. This claim edge is converted to request edge.
- Similarly, when R_j is released by P_i, the assignment edge R_j->P_i is reconverted to a claim edge P_i->R_j.

- Here note that the resources must be claimed a prior in the system. That is, before process pi starts executing, all its claim edges must appear in resource allocation graph.
- We can relax this condition by allowing a claim edge P_i->R_j to be added to the graph only if all the edges associated with process pi are claim edges.
- Suppose P_i request resource R_j. The request can be granted only if converting the edge P_i->R_j to an assignment edge R_j->P_i does not result in the formation of a cycle in the resource allocation graph.
- If no cycle exists, then the resource allocation will leave the system safely. If cycle is found the system leads to unsafe state.
- From the below graph, P_2 requests R_2. Although R_2 is currently free, we cannot allocate it to P_2. Since this action will create a cycle in the graph.
- A cycle that indicates the system is in an unsafe state. If P_1 request R_2, and P_2 request R_1, then a deadlock will occur.

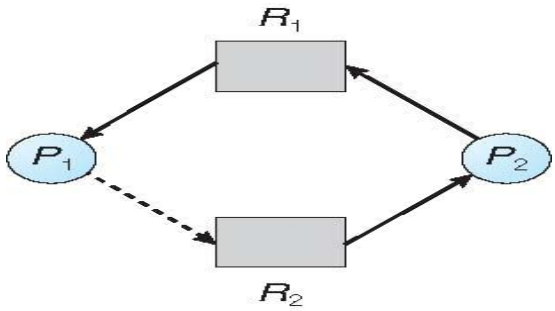


Fig: An unsafe state in a resource-allocation graph

Note: The resource-allocation-graph algorithm is not applicable to a resource allocation system with multiple instances of each resource type.

4.3.2.3. Banker's Algorithm:

- The name was chosen because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.
- When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need.
- This number may not exceed the total number of resources in the system.
- System has M resources and N processes

Data structures:

- **Available:**
  - A vector of length *m* indicates the number of available resources of each type.
  - If *Available[j]* equals *k,* then *k* instances of resource type *Ri* are available.
- **Max:**
  - An *n* x *m* matrix defines the maximum demand of each process.
  - If *Max[i] [j]* equals *k,* then process P_i may request at most *k* instances of resource type R_j.
- **Allocation:**
  - An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.
  - If *Allocation[i][j]* equals k, then process P_i is currently allocated k instances of resource type R_j.
- **Need:**
  - An *n* x *m* matrix indicates the remaining resource need of each process.
  - If *Need[i][j]* equals *k,* then process P_i may need *k* more instances of resource type R_j to complete its task. Note that *Need[i][j]* equals *Max[i][j]- Allocation [i][j].*

4.3.2.3.1. Safety Algorithm

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

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize *Work*= *Available* and *Finish[i]* =*false* for $i = 0, 1, \ldots, n - 1$.
2. Find an index *i* such that both

   a. *Finish[i]* ==*false*

   b. $Need_i$ <= *Work*

   If no such *i* exists, go to step 4.

3. Work = Work + Allocationi

   Finish[i] = *true*

   Go to step 2.

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

This algorithm may require an order of *m* x n^2 operations to determine whether a state is safe.

4.3.2.3.2. Resource-Request Algorithm:

- This algorithm for determining whether requests can be safely granted.
- Let $Request_i$ be the request vector for process P_i.
- If $Request_i$ *[j]==k,* then 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 $Request_i$ <=$Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If $Request_i$ <=*Available,* go to 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 P_i by modifying the state as follows:

$Available = Available - Request_i;$

$Allocation_i = Allocation_i + Request_i;$

$Need_i = Need_i - Request_i;$

- If the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources.
- However, if the new state is unsafe, then P_i must wait for $Request_i$, and the old resource-allocation state is restored.

4.3.2.3.3. An Illustrative Example:

- To illustrate the use of the banker's algorithm, consider a system with
  - Five processes *Po* through *P4*
  - Three resource types *A, B,* and C.
  - Resource type *A* has 10 instances, resource type *B* has 5 instances, and resource type C has 7 instances.
  - Suppose that, at time *T0* , the following snapshot of the system has been taken:

AVAILABLE: 3 3 2			
Process	**Allocation**	**Max**	**Need**
	A B C	**A B C**	**A B C**
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

  - The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

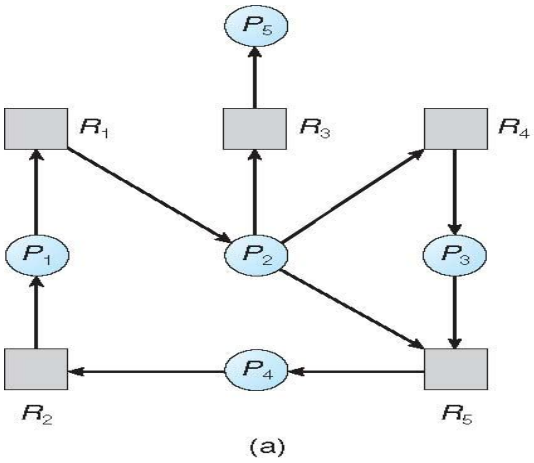
4.3.3. Deadlock Detection:

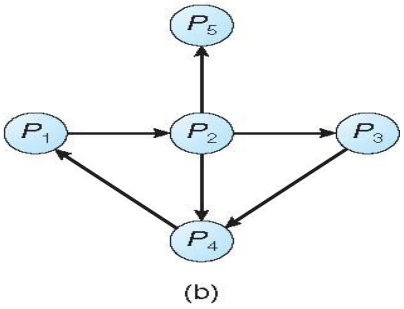
- If the system does not employ either deadlock prevention or deadlock avoidance algorithms then Deadlock situation may occur.
- In this situation, the system must provide:

- An algorithm checks the state of the system to determine whether a deadlock has occurred.
- An algorithm to recover from deadlock.

4.3.3.1. Single Instance of Each Resource Type

- If all resources are only one single instance, then we define a deadlock detection algorithm that uses wait-for graph.
- We obtain this graph by removing nodes of type resource and collapsing the appropriate edges.
- An edge from pi to P_j in a wait for graph implies that process P_i is waiting for process P_j to release a resource that P_i needs.
- An edge P_i->P_j exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges P_i->R_q and R_q->P_j for some resource R_q.





Resource- Allocation Graph **Corresponding Wait-for Graph**

- A deadlock exists in the system if and only if the wait-for graph contains a cycle.
- To detect deadlocks, the system needs to maintain the wait-for graph and periodically to invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

4.3.3.2. Several Instances of a Resource Type

- The wait-for graph is not applicable for multiple instances of each resource type.
- The deadlock detection algorithm is applicable for multiple instances of a resource type.
- This algorithm uses several data structures that are similar to banker's algorithm

- **Available:** it indicates the number of available resources of each type.
- **Allocation:** it defines the number of resources of each type currently allocated to each process.
- **Request:** it tells the current request of each process.

Detection Algorithm:

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

Example:

- To illustrate this algorithm, we consider a system with five processes *Po* through *P4* and three resource types *A, B,* and C.
- Resource type *A* has seven instances, resource type *B* has two instances, and resource type C has six instances.
- Suppose that, at time *T0,* we have the following resource-allocation

state:

	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	

- Sequence <P_0, P_2, P_3, P_1, P_4> will result in Finish[i] = true for all i
- If P_2 requests an additional instance of type C

	Request
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

State of system:

- Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests.
- Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

4.3.3.3. Detection-Algorithm Usage:

We invoke detection algorithms based on two factors:

1) How often is a deadlock likely to occur?
2) How many processes will be affected by deadlock when it happens?

- If the deadlock occurs frequently, then the detection algorithm should be invoked frequently.
- Resource allocated to deadlocked processes will be idle until the deadlock can be broken.
- Deadlocks occur only when some process makes a request that can't be granted every time.

- We could invoke detection algorithm every time a request for allocation cannot be granted immediately.
- In this case we can identify the process not only the set of processes that is deadlocked but also the specific process that caused the deadlock.
- Invoking deadlock detection algorithm for every request may incur a overhead.
- The alternative is invoking the algorithm at less frequent intervals. For example, once per hour, or whenever CPU utilization drops below 40 percentage.

4.3.4. Recovery from Deadlock:

There are three basic approaches to recovery from deadlock:

1. Inform the system operator, and allow him/her to take manual intervention.
2. Terminate one or more processes involved in the deadlock
3. Preempt resources

4.3.4.1. Process Termination

- Two basic approaches, both of which recover resources allocated to terminate processes:

Abort all deadlocked processes: Terminate all processes involved in the deadlock. This definitely solves the deadlock, but at the expense of terminating more processes than would be absolutely necessary.

Abort one process at a time until the deadlock cycle is eliminated: Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.

- In the second case there are many factors that can go into deciding which processes to terminate next:

1. Process priorities.
2. How long the process has been running, and how close it is to finishing.
3. How many and what type of resources is the process holding.

4. How many more resources does the process need to complete.
5. How many processes will need to be terminated?
6. Whether the process is interactive or batch.
7. Whether or not the process has made non restorable changes to any resource.

4.3.4.2. Resource Pre emption:

- When preempting resources to relieve deadlock,

  There are three important issues to be addressed:

Selecting a victim deciding

- Which resources to pre-empt from which processes involves many of the same decision criteria outlined above.

Rollback

- After preemption of resources from a process, what should be done with that process?
- It is missing some needed resource. So, it cannot continue its execution normally.
- We must roll back the process to some safe state and restart it from that state.
- Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only solution is total roll back.
- Abort the process and then restart it, although it is effective to rollback as necessary to break the deadlock.

Starvation

- How do you guarantee that a process won't starve because its resources are constantly being pre empted?
- In a system if selecting a victim is based on cost factor, it may happen that the same process is always picked as a victim.
- We must ensure that a process can be picked as a victim only a finite number of times.
- Solution: include number of rollbacks in cost factor

PART II: Mass-Storage Structure

4.4. Overview of Mass-Storage Structure

4.4.1. Magnetic Disks

Traditional magnetic disks have the following basic structure:

- It provides the bulk of secondary storage.
- Disks are relatively simple
- Each disk platter has a flat circular shape like CD
- Platter diameter ranges from 1.8 to 5.25
- The two surfaces of a platter are covered with magnetic material.
- We store information by recording magnetically on the platters.
- A read-write head flies just above each surface of every platter.
- Heads are attached to a disk arm that moves all the heads as a unit.
- The surface of a platter is logically divided into circular tracks
- Tracks are subdivided into sectors.
- The set of arcs that are at one arm position makes up a cylinder.
- The storage capacity of common disk drives is measured in giga bytes.
- When disk is in use a drive motor spins at high speed
- Drives rotate at 60 to 250 times per second
- **Transfer rate** is rate at which data flow between drive and computer
- **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
- **Head crash** results from disk head making contact with the disk surface.
- A head crash cannot be repaired; the entire disk must be replaced.
- Disks can be removable
- Drive attached to computer via **I/O bus**
- Busses vary, including
    - **EIDE(Enhanced integrated drive electronics)**,
    - **ATA(Advanced technology attachment)**,
    - **SATA(Serial ATA)**,
    - **USB**,

- **Fiber Channel**,
- **SCSI.**

- **Data transfers on a bus are carried out by special electronic processors called controllers.**
- **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array.
- To perform disk I/O operation,
  - Computer → sends command to → host controller → sends that command via messages to → disk controller →Operates the disk drive hardware to complete the command

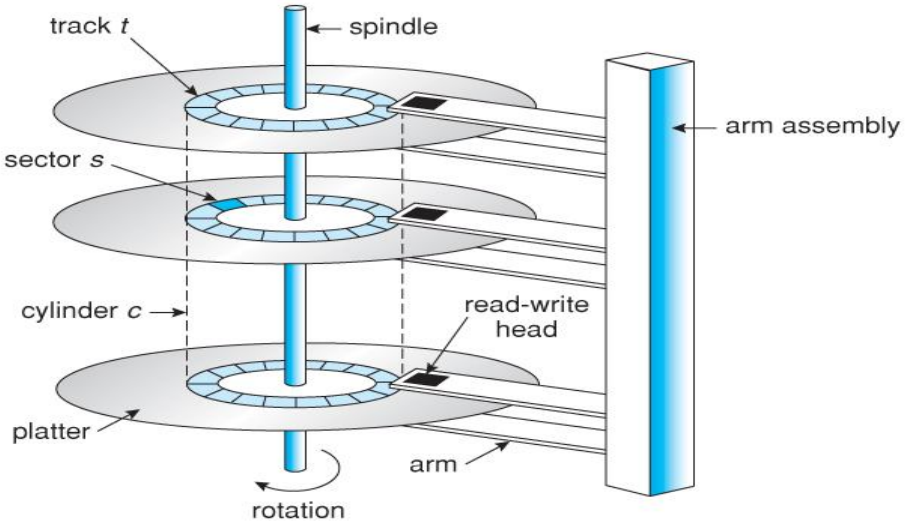


Figure 1.1 - Moving-head disk mechanism

4.4.2. Magnetic Tapes

- Magnetic tapes were once used for common secondary storage before the days of hard disk drives.
- These are relatively permanent and holds large quantities of data
- Today these are used primarily for backup, storage of infrequently-used data and acts as a medium for transferring information from one system to another.
- Access time is slow compared to main memory and magnetic disk

- Accessing a particular spot on a magnetic tape can be slow, but once reading or writing commences, access speeds are comparable to disk drives.
- Capacities of tape drives can range from 20 to 200 GB, and compression can double that capacity.

4.5. Disk Scheduling

- One of the responsibilities of operating system is to use the hardware efficiently.
- For the disk drive meeting this responsibility is having fast access time and large disk bandwidth.
- Access time= Seek time +Rotational Latency
- Disk Bandwidth=Total number of bits transferred, divided by the total time between the first request for service and the completion of the last transfer.
- We can improve both the access time and the bandwidth by scheduling the servicing of disk I/O requests in a good manner.
- Several algorithms exist to schedule the servicing of disk I/O requests
- We illustrate scheduling algorithms with a request queue (0-19

  Set of requests: 98, 183, 37, 122, 14, 124, 65, 67

  Head pointer 53

4.5.1. FCFS Scheduling

- ***First-Come First-Serve*** is simple and intrinsically fair, but not very efficient.
- Consider in the following sequence the wild swing from cylinder 122 to 14 and then back to 124:

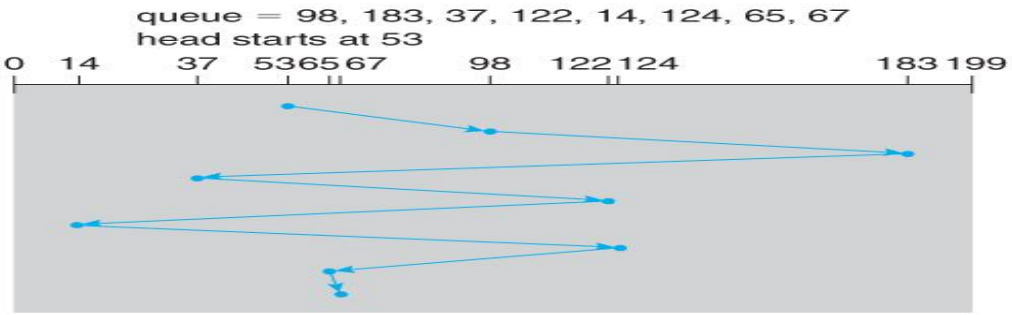


Figure: FCFS disk scheduling.

4.5.2. SSTF Scheduling

- ***Shortest Seek Time First*** scheduling is more efficient, but may lead to starvation if a constant stream of requests arrives for the same general area of the disk.
- SSTF reduces the total head movement to 236 cylinders, down from 640 required for the same set of requests under FCFS.

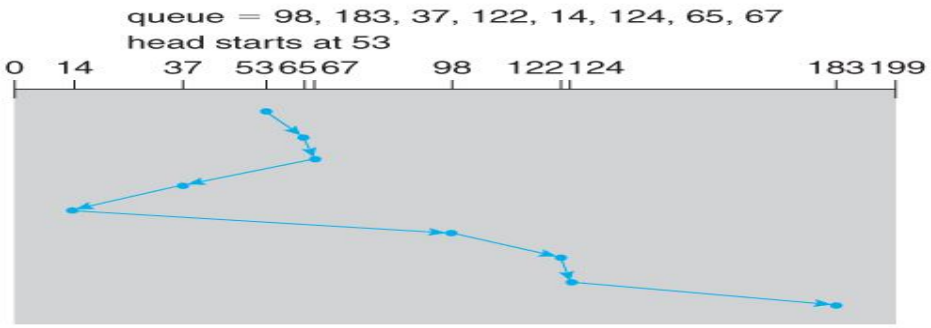


Figure: SSTF disk scheduling.

4.5.3. SCAN Scheduling

- The ***SCAN*** algorithm also known as the ***elevator*** algorithm
- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

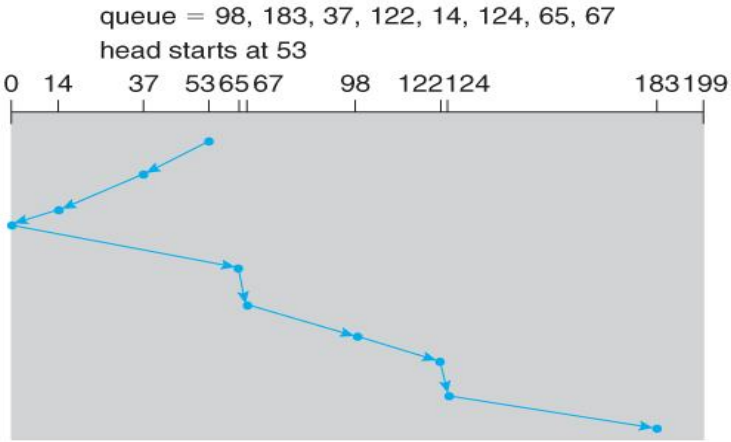


Figure: SCAN disk scheduling.

4.5.4. C-SCAN Scheduling

- The *Circular-SCAN* algorithm improves upon SCAN by treating all requests in a circular queue fashion - Once the head reaches the end of the disk, it returns to the other end without processing any requests, and then starts again from the beginning of the disk:

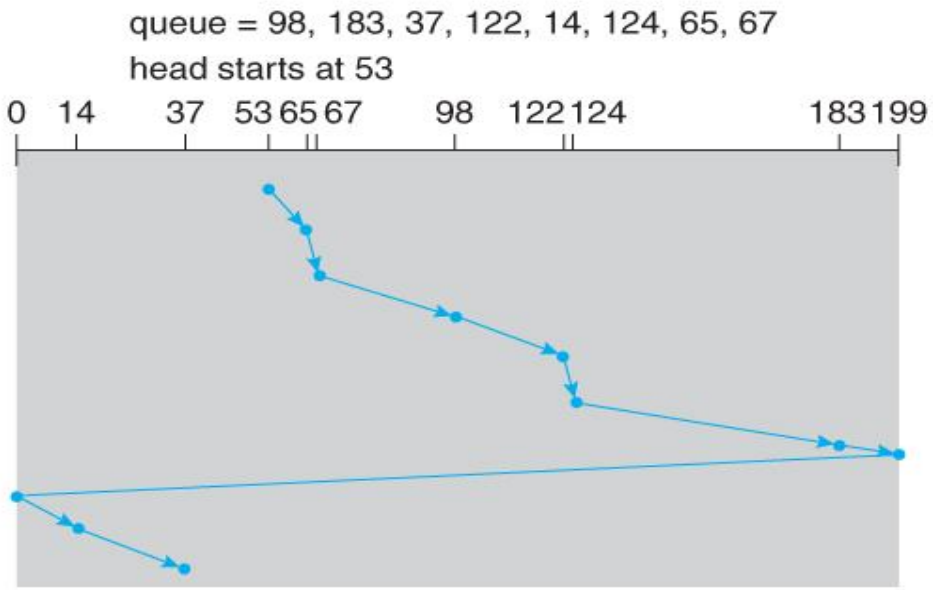


Figure: C-SCAN disk scheduling.

4.5.5. LOOK Scheduling

- ***LOOK*** scheduling improves upon SCAN by looking ahead at the queue of pending requests, and not moving the heads any farther towards the end of the disk than is necessary. The following diagram illustrates the circular form of LOOK:

4.5.6. C-LOOK Scheduling:

- LOOK a version of SCAN, C-LOOK a version of C-SCAN.
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk

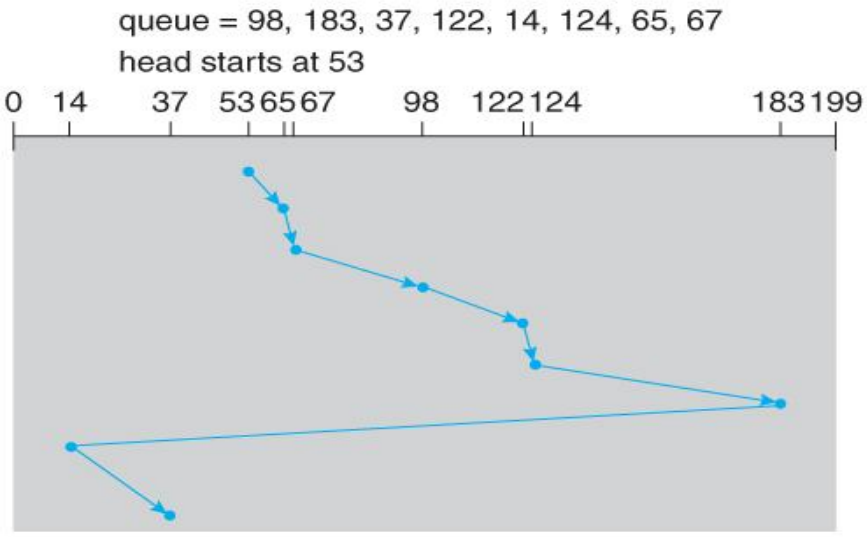


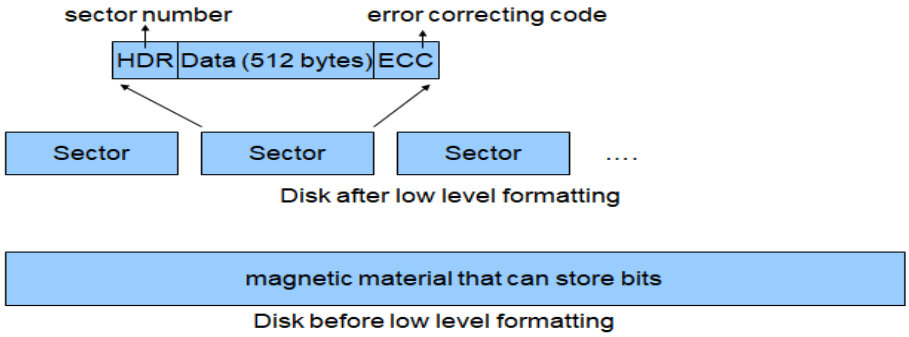
Figure: C-LOOK disk scheduling.

4.6. Disk Management:

4.6.1. Disk Formatting:

- Before a disk can be used, it has to be *low-level formatted*, which means laying down all of the headers and trailers marking the beginning and ends of each sector.
- **Low-level formatting**, or **physical formatting** — Dividing a new disk into sectors that the disk controller can read and write
  - This formatting fills the disk with a special data structure for each sector
  - Each sector can hold header information + data + Trailer
  - Header and trailer contain information used by the disk controller
    - Sector number
    - ECC (Error correcting code)

Low Level Formatting



- Controller –Write a sector -Update ECC with a value calculated based on data.
- Read a sector- recalculate ECC-compare with the stored value
- If stored value mismatch with recalculated value –data is corrupted in sector-Bad sector
- ECC- contains enough information if a few number of bits are corrupted.
- Controller can identify which bits are changed and calculate their correct values.
- The controller automatically does ECC processing whenever a sector is read or written.

4.6.2. Boot Blocks:

- Computer ROM contains a *bootstrap* program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it.
- The first sector on the hard drive is known as the *Master Boot Record, MBR,* and contains a very small amount of code in addition to the *partition table.*
- The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the *active* or *boot* partition.

- The boot program then looks to the active partition to find an operating system, possibly loading up a slightly larger / more advanced boot program along the way.
- In a *dual-boot* (or larger multi-boot) system, the user may be given a choice of which operating system to boot, with a default action to be taken in the event of no response within some time frame.
- Once the kernel is found by the boot program, it is loaded into memory and then control is transferred over to the OS.
- The kernel will normally continue the boot process by initializing all important kernel data structures, launching important system services (e.g. network daemons, sched, init, etc.), and finally providing one or more login prompts.
- Boot options at this stage may include *single-user* a.k.a. *maintenance* or *safe* modes, in which very few system services are started - These modes are designed for system administrators to repair problems or otherwise maintain the system.

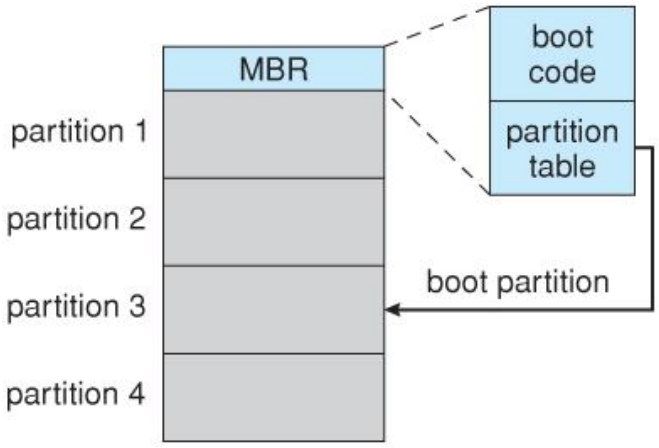


Figure: Booting from disk in Windows 2000.

4.6.3. Bad Blocks:

- No disk can be manufactured to 100% perfection, and all physical objects wear out over time.

- For these reasons all disks are shipped with a few bad blocks, and additional blocks can be expected to go bad slowly over time.
- If a large number of blocks go bad then the entire disk will need to be replaced, but a few here and there can be handled through other means.
- In the old days, bad blocks had to be checked for manually.
- Formatting of the disk or running certain disk-analysis tools would identify bad blocks, and attempt to read the data off of them one last time through repeated tries.
- Then the bad blocks would be mapped out and taken out of future service.
- Sometimes the data could be recovered, and sometimes it was lost forever.
- Modern disk controllers make much better use of the error-correcting codes, so that bad blocks can be detected earlier and the data usually recovered.
- Most disks normally keep a few spare sectors on each cylinder, as well as at least one spare cylinder.
- ***Sector sparing***: Whenever possible a bad sector will be mapped to another sector on the same cylinder, or at least a cylinder as close as possible.
- ***Sector slipping*** may also be performed, in which all sectors between the bad sector and the replacement sector are moved down by one, so that the linear progression of sector numbers can be maintained.
- If the data on a bad block cannot be recovered, then a ***hard error*** has occurred, which requires replacing the file(s) from backups, or rebuilding them from scratch.

****

UNIT-IV

Assignment-Cum-Tutorial Questions

SECTION-A

I. Objective Questions

1. A direct edge P_i---->R_j is called a_____ [ ]
   A) Assignment edge C) Request edge
   B) Claim edge D) Release edge
2. A direct edge R_j----> P_i is called a_____ [ ]
   A) Assignment edge C) Request edge
   B) Claim edge D) Release edge
3. Deadlocks can be described in terms of a directed graph called a_____
   A) Directed Acyclic Graph [ ]
   B) Resource allocation graph
   C) Resource request graph
   D) Resource release graph
4. If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred.
   [T/F]
5. If each resource type has exactly several instances, then a cycle does not imply that a deadlock has occurred.
   [T/F]
6. The surface of a platter is logically divided into circular_____[ ]
   A) Sectors B) Tracks C) platters D) surfaces
7. C-SCAN refers to_____ [ ]
   A) Coding SCAN C) Ceil SCAN
   B) Circular SCAN D) City SCAN
8. SCAN algorithm is also called as_____ [ ]
   A) Circular SCAN B) elevator C) LOOK D)
   B) C-LOOK
9. The time to move from the disk arm to the desired cylinder is called_____

A) Rotational latency [ ]

B) Seek time

C) Transfer rate

D) Random-access time

10. The time for the desired sector to rotate to the disk head is called_____.

A) Rotational latency [ ]

B) Seek time

C) Transfer rate

D) Random-access time

11. Which one of the following statement about WAIT-FOR graph is true? [ ]

A) An edge Pi->Pj exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges Pi->Rq and Rq->Pj for some resource Rq.

B) An edge Pi->Rj exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges Pi->Rq and Rq->Pj for some resource Rq.

C) An edge Pi->Pj exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges Pi-> Pj and Rq->Pj for someresource Rq.

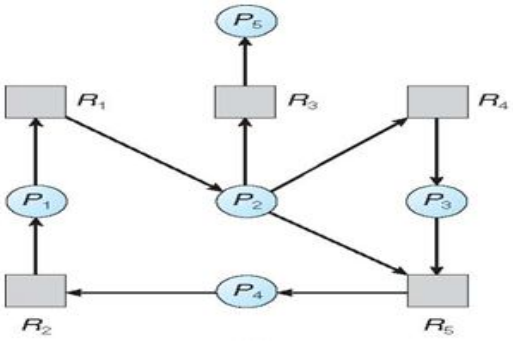
D) An edge Pi->Pj exists in a wait for graph if and only if the corresponding resource allocation graph contains two edges P_i->R_q and P_i ->P_j for some resource R_q.

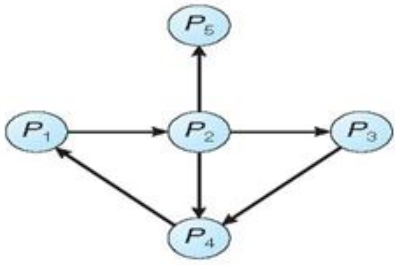
12. Which of the following approaches are used to recover from dead lock

A) Process termination C)Resource preemption

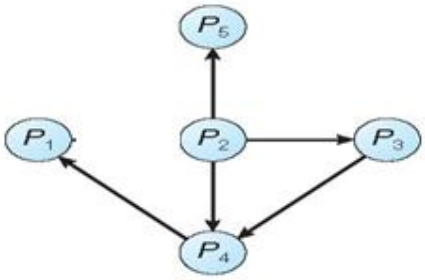
B) Both of the above methods D) None of the above [ ]

13. Which one of the following wait-for graph is equivalent to the given Resource Allocation graph? [ ]

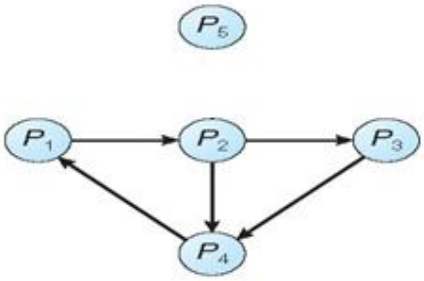




A)



B)



C)

D) No wait-for graph for the given RAG

14. **Consider a system having 'm' resources of the same type. These resources are shared by 3 processes A, B, C, which have peak time demands of 3, 4, 6 respectively. The minimum value of 'm' that ensures that deadlock will never occur is [ ]**

A) 11 B) 12 C) 13 D) 14

15. Which algorithm of disk scheduling selects the request with the least seek time from the current head positions? [ ]
    A) SSTF scheduling C)FCFS scheduling
    B) SCAN scheduling D) LOOK scheduling
16. The circular wait condition can be prevented by [ ]
    A) Defining a linear ordering of resource types C) Using thread
    B) Using pipes D) All of the mentioned
17. For non sharable resources like a printer, mutual exclusion [ ]
    A) Must exist C) Must not exist
    B) May exist D) None of these
18. The disadvantage of a process being allocated all its resources before beginning its execution is : [ ]
    A) Low CPU utilization C) Low resource utilization
    B) Very high resource utilization D) None of these
19. To ensure no preemption, if a process is holding some resources and requests another resource that cannot be immediately allocated to it : [ ]
    A) Then the process waits for the resources be allocated to it
    B) The process keeps sending requests until the resource is allocated to it
    C) The process resumes execution without the resource being allocated to it
    D) Then all resources currently being held are preempted
20. A system has 12 magnetic tape drives and 3 processes : P0, P1, and P2. Process P0 requires 10 tape drives, P1 requires 4 and P2 requires 9 tape drives. [ ]

Process	Maximum needs	Currently allocated
P0	10	5
P1	4	2
P2	9	2

Which of the following sequence is a safe sequence?

A) P0, P1, P2 C) P1, P2, P0

B) P2, P0, P1 D) P1, P0, P2

21. The content of the matrix Need is : [ ]

A) Allocation – Available C) Max – Available

B) Max – Allocation D) Allocation – Max

22. An edge from process Pi to Pj in a wait for graph indicates that :

A) Pi is waiting for Pj to release a resource that Pi needs. [ ]

B) Pj is waiting for Pi to release a resource that Pj needs.

C) Pi is waiting for Pj to leave the system.

D) Pj is waiting for Pi to leave the system.

23. A computer system has 6 tape drives, with 'n' processes competing for them. Each process may need 3 tape drives. The maximum value of 'n' for which the system is guaranteed to be deadlock free is : [ ]

A) 2 B) 3 C) 4 D) 1

24. A system has 3 processes sharing 4 resources. If each process needs a maximum of 2 units then, deadlock : [ ]

A) Can never occur. C) any occur.

B) Has to occur. D) None of these.

SECTION-B

Descriptive Questions

1. Define deadlock and classify the necessary conditions for deadlock?
2. List and explain different methods used for handling deadlocks?
3. Describe in detail about BANKER'S algorithm?
4. With a neat sketch explain the overview of mass storage structure.
5. Differentiate SCAN, C-SCAN and LOOK, C-LOOK disk scheduling algorithms with an example?
6. What is sector sparing? Explain how it is useful in identifying bad blocks in mass storage?
7. Demonstrate in detail about swap-space management?

Problems:

1. Consider the snapshot of a system processes p1, p2, p3, p4, p5,
   Resources A, B, C, D
   Allocation[0 0 1 2, 1 0 0 0, 1 3 5 4, 0 6 3 2, 0 0 1 4]
   Max[0 0 1 2, 1 7 5 0, 2 3 5 6, 0 6 5 2, 0 6 5 6]
   Available[1 5 2 0] .
   i. What will be the content of the Need matrix?
   ii.Is the system in safe state? If Yes, then what is the safe sequence?

2. Consider the following and find out the possible resource allocation sequence with the help of deadlock detection algorithm processes p0, p1, p2, p3, p4, Resources A, B, C
   Allocation [0 1 0, 2 0 0 , 3 0 3, 2 1 1, 0 0 2]
   Max[0 0 0, 2 0 2, 0 0 0, 1 0 0, 0 0 2]
   Available[0 0 0].
   i. What will be the content of the Need matrix?
   ii.Is the system in safe state? If Yes, then what is the safe sequence?

3. A computer system uses the Banker's Algorithm to deal with deadlocks. Its current state is shown in the table below, where P0, P1, P2 are processes, and R0, R1, R2 are resources types.

	Maximum Need			**Current Allocation**			**Available**		
	R0	**R1**	**R2**		**R0**	**R1**	**R2**	**R0**	**R1**	**R2**
P0	4	1	2	**P0**	1	0	2	2	2	0
P1	1	5	1	**P1**	0	3	1			
P2	1	2	3	**P2**	1	0	2			

i. Show that the system can be in safe state?

ii. What will the system do on a request by process P0 for one unit of resource type R1?

4. Four resources ABCD. A has 6 instances, B has 3 instances, C has instances and D has 2 instances.

Process	Allocation ABCD	Max ABCD
P1	3011	4111
P2	0100	0212
P3	1110	4210
P4	1101	1101
P5	0000	2110

i. Is the current state safe?

ii. If P5 requests for (1,0,1,0), can this be granted?

5. Why disk scheduling is needed? Schedule the given requests **98, 183, 37, 122, 14, 124, 65, 67, 10, 150** with the following disk scheduling algorithms and calculate seek time?

   a. FCFS disk scheduling

   b. SSTF disk scheduling

   c. SCAN disk scheduling

   d. C-SCAN disk scheduling

   e. LOOK disk scheduling

   f. C-LOOK disk scheduling

SECTION-C

Previous GATE/NET questions

1. A system contains three programs and each requires three tape units for its operation. The minimum number of tape units which the

system must have such that deadlocks never arise is_____

GATE-CS-2014 [ ]

A) 6 B) 7 C) 8 D) 9

2. A system has 6 identical resources and N processes competing for them. Each process can request atmost 2 resources. Which one of the following values of N could lead to a deadlock? **GATE-CS-2015**

[ ]

A) 1 B) 2 C) 3 D) 4

3. Considering a system with five processes P0 through P4 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 t0 following snapshot of the system has been taken: **GATE-CS-2014**

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	

i. What will be the content of the Need matrix?
ii. Is the system in safe state? If Yes, then what is the safe sequence?

4. An operating system uses the Banker's algorithm for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes P0, P1, and P2. The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a safe state. Consider the following independent requests for additional resources in the current state:

REQ1: P0 requests 0 units of X, 0 units of Y and 2 units of Z

REQ2: P1 requests 2 units of X, 0 units of Y and 0 units of Z

Which one of the following is TRUE? **GATE-CS-2014** [ ]

A) Only REQ1 can be permitted.

B) Only REQ2 can be permitted.

C) Both REQ1 and REQ2 can be permitted.

D) Neither REQ1 nor REQ2 can be permitted

5. Which of the following is NOT a valid deadlock prevention scheme?

GATE CS 2000 [ ]

A) Release all resources before requesting a new resource

B) Number the resources uniquely and never request a lower numbered resource than the last one requested.

C) Never request a resource after releasing any resource

D) Request and all required resources be allocated before execution