Scheme of Evaluation

Internal Assessment Test 2 – February 2026

Sub:	**Operating Systems**							**Sub Code:**	**MMC104**
Date:	20-02-26	Duration:	90 mins	Max Marks:	50	**Sem:**	I	**Branch:**	**MCA**

Q.NO	Description	Marks Distribution	Max Marks
1	**Describe about the virtual memory management** • **Explanation of Virtual Memory** • **Draw a Virtual memory block diagram**	**7** **3**	**10**
2	**Explain TLB with neat diagram** • Explanation of TLB • Draw a TLB Architecture diagram	**7** **3**	**10**
3	**Explain demand paging? Explain how demand paging can be implemented** • Explanation of Demand Paging • Implementation of demand paging • Pros and cons of demand paging	**2** **6** **2**	**10**
4	**Consider the following reference string:** **7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1** **How many page faults would occur in case of i)FIFO ii) LRU iii)Optimal algorithm?** • Find out the solutions of page fault using FIFO,LRU and Optimal page replacement algorithm	**10**	**10**
5	**How can deadlock be prevented describe them** • Definition of deadlock prevention • Give explain about the necessary conditions of deadlock prevention	**5** **5**	**10**
6	**Explain the deadlock detection algorithm for several instance of a resource** • Definition of deadlock detection • Give explain about the steps of deadlock detection algorithm	**3** **7**	**10**
7	**Write and explain about the Banker's algorithm** • Explanation of Banker's algorithm • Steps of Banker's algorithm	**4** **6**	**10**

<table>
<tr>
<td>8</td>
<td>Consider the following

<table>
<tr><td></td><td><u>Allocation</u></td><td><u>Max</u></td><td><u>Available</u></td></tr>
<tr><td></td><td>A B C</td><td>A B C</td><td>A B C</td></tr>
<tr><td>P_0</td><td>0 1 0</td><td>7 5 3</td><td>3 3 2</td></tr>
<tr><td>P_1</td><td>2 0 0</td><td>3 2 2</td><td></td></tr>
<tr><td>P_2</td><td>3 0 2</td><td>9 0 2</td><td></td></tr>
<tr><td>P_3</td><td>2 1 1</td><td>2 2 2</td><td></td></tr>
<tr><td>P_4</td><td>0 0 2</td><td>4 3 3</td><td></td></tr>
</table>
Answer the following question: i) Construct a need matrix.
ii) Is the system in a safe state? If yes what is the safe sequence?
iii) If process P1 makes a request (1 0 2) can the request be granted?

Find out the solution of need matrix
Find out the solution of safe
sequence
</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>Explain file allocation methods with neat diagrams

Explanation of file allocation methods
Draw a diagram of each file allocation method
</td>
<td>5
5</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>Explain how free space management is done using i) Bit vector ii) Linked list iii) Grouping iv) Counting

Explanation of free space management
Explanation of each method with diagram
And also pros and cons of each method of free space management.
</td>
<td>2
6
2</td>
<td>10</td>
</tr>
</table>

Internal Assessment Test 2 – February 2026

Operating Systems							Sub Code:	MMC104
20/02/2026	Duration:	90 min's	Max Marks:	50	Sem:	I	Branch:	MCA

PART I

1. Describe about the virtual Memory management

Virtual memory is a technique that allows the execution of processes that may not be completely in memory. One major advantage of this scheme is that programs can be larger than physical memory. This technique frees programmers form the concerns the of memory-storage limitations. Virtual memory also allows processes to easily share files and address space, and it provides an efficient mechanism for process creation. It is a virtual resource of a computer. It is an illusion that a computer system processes more memory that is actually having. This illusion makes a process independent of the size of main memory. It also permits a large number of processes to share a computer system without constraining each other. Virtual memory allows execution of partially loaded processes. As a consequence, virtual address spaces of active processes in a virtual-memory system can exceed the capacity of the physical memory. This is accomplished by maintaining an image of the entire virtual-address space of a process on secondary storage, and by bringing its sections into main memory when needed. The Operating System decides which sections to bring in, when to bring them in, and where to place them. Thus, virtual-memory systems provide for automatic migration of portions of address spaces between secondary and primary storage. Virtual memory provides the illusion of a much larger memory than may actually be available, so programmers are relieved of the burden of trying to fit a program into limited memory.

The ability to execute a program that is only partially in memory would confer many benefits:

➢ A program would no longer be constrained by the amount of physical memory that is available. User would be able to write programs for an extremely large virtual-address space simplifying the programming task.

➢ Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding increase in CPU utilization and throughput, but with no increase in response time or turnaround time.

➢ Less I/O would be needed to load or swap each user program into memory, so each user program would run faster.

Thus, running a program that is not entirely in memory would benefit both the system and the user.

Virtual memory is to be provided for programmers when only a smaller physical memory is available.

Virtual memory is the separation of user logical memory from physical memory this separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available (Figure 1).

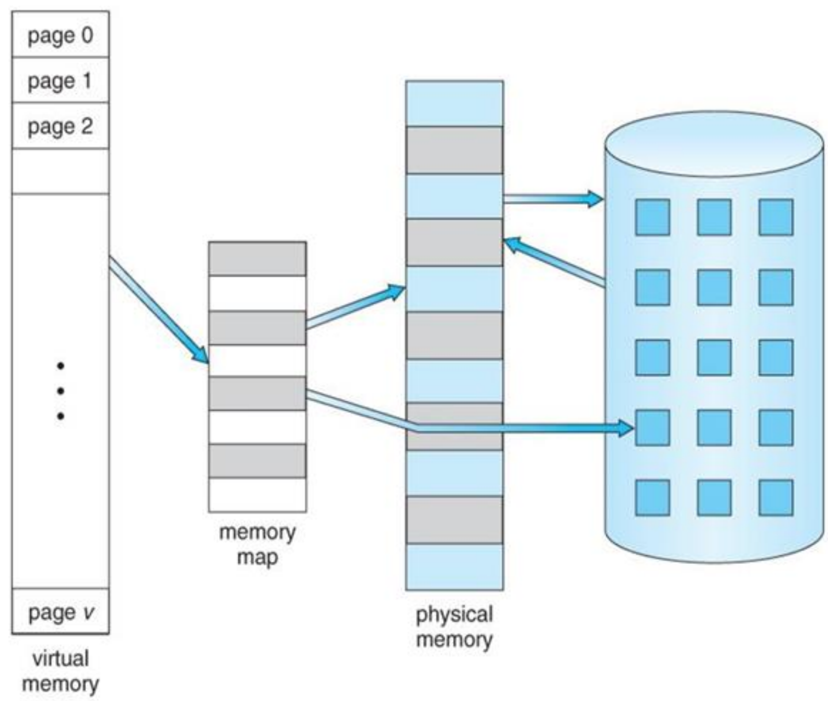


Fig 1: Virtual Memory

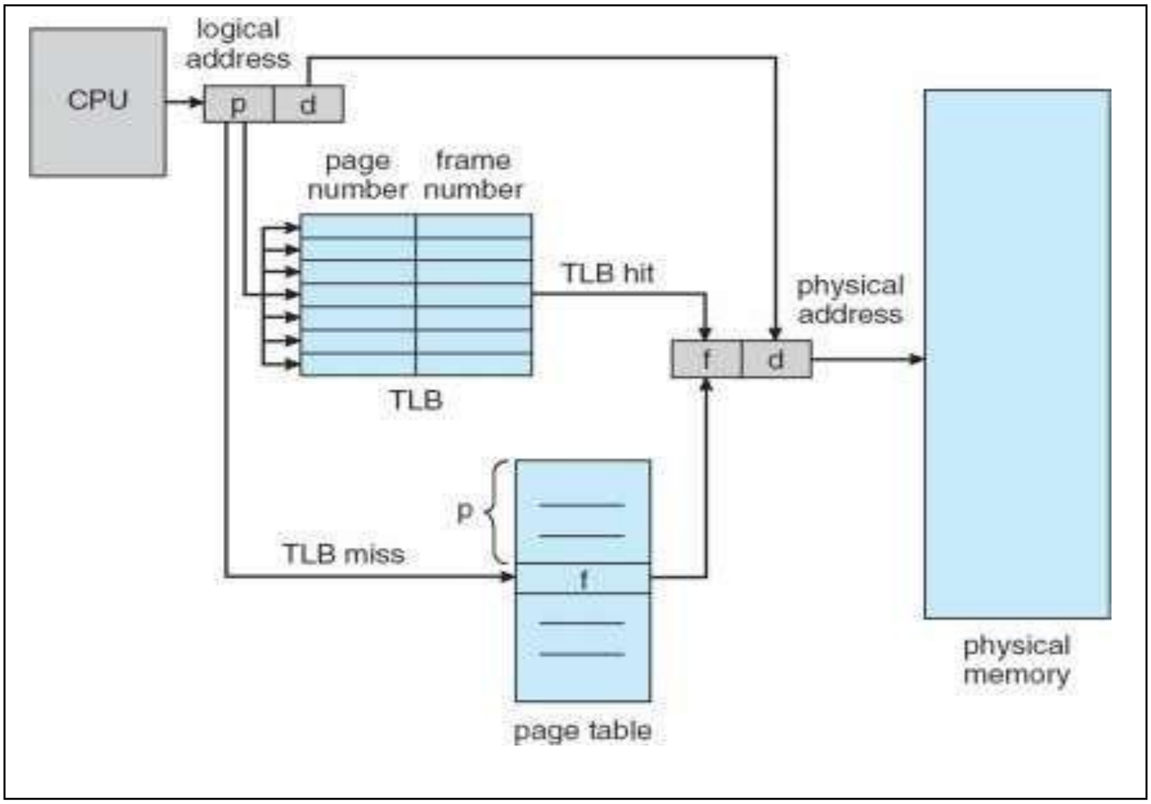
Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory. The segment replacement algorithms are very complex in use because of variable sizes of segments while page replacement algorithms are easy to use and implement.

2. Explain TLB with neat diagram

The hardware implementation of the page table can be done by using dedicated registers. But the usage of the register for the page table is satisfactory only if the page table is small.

If the page table contains a large number of entries then we can use TLB (translation Look-aside buffer), a special, small, fast look-up hardware cache.

- The TLB is an associative, high-speed memory.
- Each entry in TLB consists of two parts: a tag and a value.
- When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then the corresponding value is returned.



When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory.

If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory. In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference. If the TLB is already full of entries, the operating system must select one for replacement. Replacement policies range from least recently used (LRU) to random. Furthermore, some TLBs allow entries to be wired down, meaning that they cannot be removed from the TLB. Typically, TLB entries for kernel code are often wired down.

The percentage of times that a particular page number is found in the TLB is called the **hit ratio**. An 80-percent hit ratio means that we find the desired page number in the TLB 80 percent of the time. If it takes 20 nanoseconds to search the TLB, and 100 nanoseconds to access memory, then a mapped memory access takes 120 nanoseconds when the page number is in the TLB. If we fail to find the page number in the TLB (20 nanoseconds), then we must first access memory for the page table and frame number (100 nanoseconds), and then access the desired byte in memory (100 nanoseconds), for a total of 220 nanoseconds.

To find the effective memory-access time, we must weigh each case by its probability: (Where P is Hit ratio)

EAT(effective access time) = P x hit memory time + (1-P) x miss memory time.
= 0.80 x 120 + 0.20 x 220
= 140 nanoseconds.
In this example, we suffer a 40-percent slowdown in memory access time (from 100 to 140 ns).
For a 98-percent hit ratio, we have
EAT(effective access time)= P x hit memory time + (1-P) x miss memory time.
= 0.98 x 120 + 0.02 x 220
= 122 nanoseconds.
This increased hit rate produces only a 22-percent slowdown in access time.
Example:
What will be the EAT if hit ratio is 70%, time for TLB is 30ns and access to main memory is 90ns?
P = 70% = 70/100 = 0.7
Hit memory time = 30ns + 90ns = 120ns
Miss memory time = 30ns + 90ns + 90ns = 210ns Therefore,
EAT = P x Hit + (1-P) x Miss
= 0.7 x 120 + 0.3 x 210
=840 + 63.0
=147 ns

PART II

3. Explain demand paging? Explain how demand paging can be implemented

A demand paging is similar to a paging system with swapping (Figure 2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed. When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process; the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed. Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

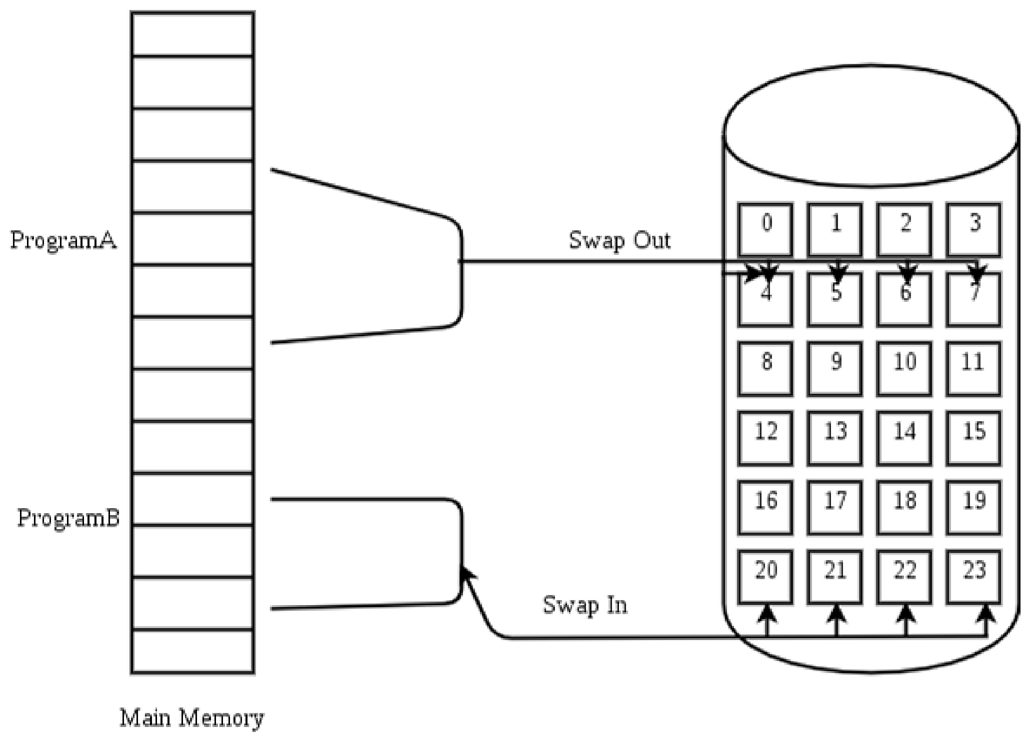


Fig 2: Transfer of a paged memory to continuous disk space

With this scheme, we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk. The valid-invalid scheme can be used for this purpose. When this bit is set to “valid”, this value indicates that the associated page is both legal and in memory. If the bit is set to “invalid”, this value indicates that the page either is not valid or is valid but is currently on the disk. The page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page that is not currently in memory is simply marked invalid, or contains the address of the page on disk. This situation is depicted in Figure 3.

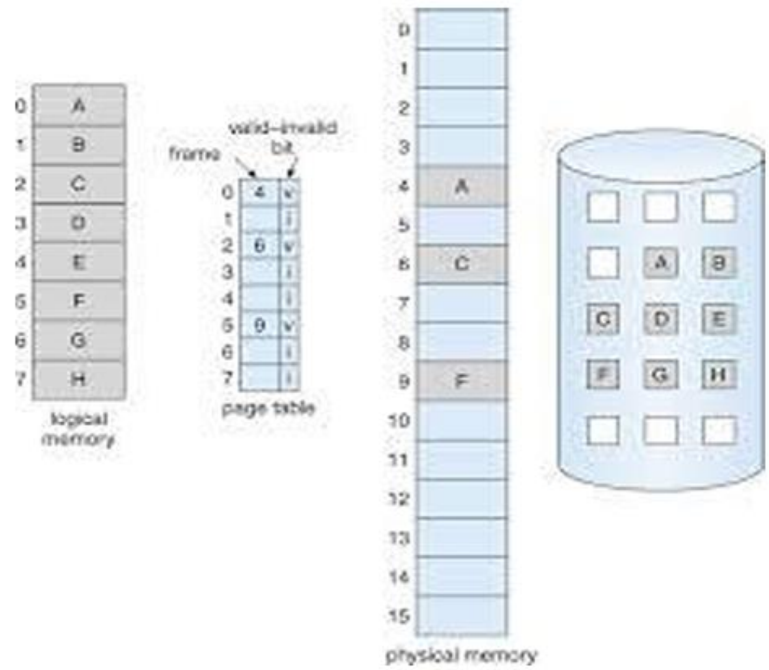


Fig 3: Page Table when some pages are not in main memory

Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory. Initially only those pages are loaded which will be required the process immediately. The pages that are not moved into the memory are marked as invalid in the page table. For an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page. When the process requires any of the pages that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

- We check an internal table (page table) for this process to determine whether the reference was valid or invalid.
- If the reference was invalid, we terminate the process, if it was valid but not yet brought in, we have to bring that from main memory.
- Now we find a free frame in memory.
- Then we read the desired page into the newly allocated frame.
  When the disk read is complete, we modify the internal table to indicate that the page is now in memory.
- We restart the instruction that was interrupted by the illegal address trap. Now the process can access the page as if it had always been in memory.

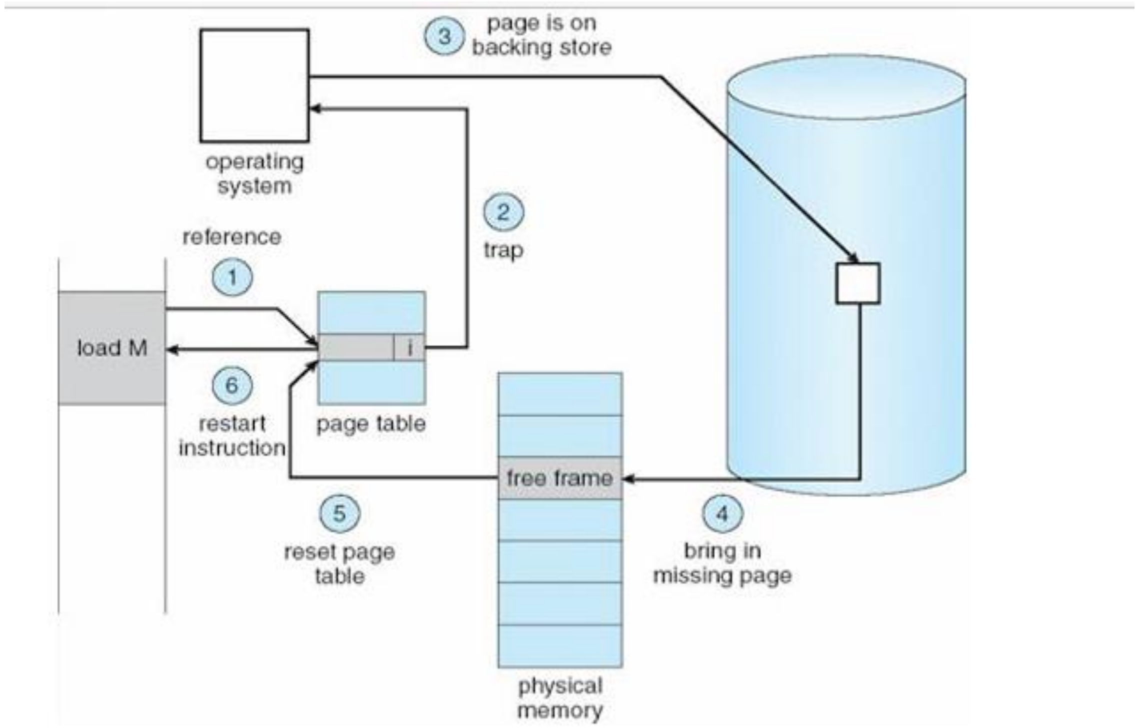


Fig 4: Steps in handling a page fault

Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory. The page replacement is used to make the frame free if they are not in used. If no frame is free then other process is called in.

Advantages of Demand Paging

➢Large virtual memory.

➢More efficient use of memory.

➢Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

Disadvantages of Demand Paging

➢Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.

➢Due to the lack of an explicit constraint on jobs address space size.

4.Consider the following reference string:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
How many page faults would occur in case of i)FIFO ii) LRU iii)Optimal algorithm?

8. Consider the following reference string:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
How many page faults would occur in case of
(i) FIFO (ii) LRU (iii) Optimal algorithm?

Answer :

Let us assume the no. of page frames = 3

(i) FIFO.

Ref / Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
F2		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
F3			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
Hit	×	×	×	×	✓	×	×	×	×	×	×	✓	✓	×	×	✓	✓	×	×	×

Total no. of pages = 20
page fault / page miss = 15
page hit = 5

Hit ratio = $\frac{\text{no. of page hit}}{\text{total no. of pages}} = \frac{5}{20} = 0.25$

Fault ratio = $\frac{\text{no. of page fault}}{\text{total no. of pages}} = \frac{15}{20} = 0.75$

CMR

(ii) LRU

Ref / Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
F2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
F3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
Hit	×	×	×	×	✓	×	✓	×	×	×	×	✓	✓	×	✓	×	✓	×	✓	✓

Total no. of pages = 20
page fault = 12
page hit = 8

Hit ratio = $\frac{8}{20} = 0.40$

Fault ratio = $\frac{12}{20} = 0.60$

(iii) optimal.

Ref / Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
F2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
F3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
Hit	×	×	×	×	✓	×	✓	×	✓	✓	×	✓	✓	×	✓	✓	✓	×	✓	✓

Total no. of pages = 20
page fault = 9
page hit = 911

Hit ratio = $\frac{911}{20} = 0.55$

Fault ratio = $\frac{9}{20} = 0.45$

PART III

5.How can deadlock be prevented describe them

Deadlock-Prevention

• Deadlockscanbeeliminatedbypreventingatleastoneofthefourrequiredconditions:
1) Mutual exclusion
2) Hold-and-wait
3) No preemption
4) Circular-wait.

Mutual Exclusion

• This condition must hold for non-sharable resources.
• For example:
A printer cannot be simultaneously shared by several processes.
• On the other hand, shared resources do not lead to deadlocks.
• For example:
Simultaneous access can be granted for read-only file.
• A process never waits for accessing a sharable resource.
• In general, we cannot prevent deadlocks by denying the mutual-exclusion condition because some resources are non-sharable by default.

Hold and Wait

• To prevent this condition:
The processes must be prevented from holding one or more resources while simultaneously waiting for one or more other resources.
• There are several solutions to this problem.
• For example:
Consider a process that
→copies the data from a tape drive to the disk
→sorts the file and →then prints the results to a printer.

Protocol-1

 Each process must be allocated with all of its resources before it begins execution.
 All the resources (tape drive, disk files and printer)are allocated to the process at the beginning.

Protocol-2

 A process must request a resource only when the process has none.

If(resources are available)
then
{
allocate resources to the process
}

else
{
If(resources are allocated to waiting process)
then
{
preempt the resources from the waiting
process allocate the resources to the
requesting-process the requesting-process
must wait
}
}
 Initially, the process is allocated with tape drive and disk file.
 The process performs the required operation and releases both tape drive and disk file.
 Then, the process is a gain allocated with disk file and the printer
 Again ,the process performs the required operation &releases both disk file and the printer.
• Disadvantagesofabove2 methods:
1) Resource utilization may below, since resources maybe allocated but unused for along period.
2) Starvation is possible.
No Preemption
• To prevent this condition: the resources must be preempted.
• There are several solutions to this problem.
Protocol-1
• If a process is holding some resources and requests another resource that cannot be immediately
allocated to it, then all resources currently being held are preempted.
• Thepreemptedresourcesareaddedtothelistofresourcesforwhichtheprocessiswaiting.
• The process will be restarted only when it regains the old resources and the new resources that it
is requesting.
Protocol-2
•
•
•

• When a process request resources, we check whether they are available or not.
• These 2 protocols may be applicable for resources whose states are easily saved and restored, such as registers and memory.
• But,these 2 protocols are generally not applicable to other devices such as printers and tape drives.
Circular-Wait
• Deadlockcanbepreventedbyusingthefollowing2protocol:
Protocol-1
 Assign numbers all resources.
 Requiretheprocessestorequestresourcesonlyinincreasing/decreasingorder.

Protocol-2
 Require that whenever a process requests a resource, it has released resources with a lower number.
• One big challenge in this scheme is determining the relative ordering of the different resources.

6. Explain the deadlock detection algorithm for several instance of a resource

Deadlock detection is a method used by the Operating System to determine whether a set of processes is in a deadlock state.
When **multiple instances of each resource type** are available, the deadlock detection algorithm is similar to the Banker's algorithm but uses a **Request matrix** instead of a Need matrix.

It periodically checks the system state to identify deadlocked processes.

Data Structures Used

Let:

- **n** = number of processes
- **m** = number of resource types

The following data structures are required:

1. **Available (m):**
   `Available[j]` = number of available instances of resource type Rj.
2. **Allocation (n × m):**
   `Allocation[i][j]` = number of instances of Rj allocated to process Pi.
3. **Request (n × m):**
   `Request[i][j]` = number of instances of Rj requested by process Pi.
4. **Finish (n):**
   Boolean array used to indicate whether a process can complete.
   Initially, `Finish[i] = false` for all processes.

Deadlock Detection Algorithm

Step 1:
Initialize:
If `Allocation[i] = 0`, then set `Finish[i] = true`;
otherwise `Finish[i] = false`.

Step 2:
Find a process Pi such that:

- `Finish[i] == false`
- `Request[i] ≤ Available`

If no such process exists, go to Step 4.

Step 3:
If such a process is found:

- `Available = Available + Allocation[i]`
- `Finish[i] = true`
- Go back to Step 2.

Step 4:
If any `Finish[i] == false`, then those processes are **deadlocked**.
If all processes are marked `true`, then **no deadlock exists**.

The deadlock detection algorithm for several instances works by simulating process completion using available resources. If some processes cannot complete after repeated checks, they are declared deadlocked. This method is used in systems where deadlock avoidance is not implemented.

PART IV

7.Write and explain about the Banker's algorithm

Bankers' Algorithm is resource allocation and deadlock avoidance algorithm which test the entire request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

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

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

Available:

 It is a 1-d array of size 'm 'indicating the number of available resources of each type.

 Available[j] = k means there are 'k' instances of resource type Rj

Max:

 It is a 2-d array of size'n*m' that defines the maximum demand of each processing system.

 Max[i,j]=k means process Pi may request atmost 'k' instances of resource type Rj.

Allocation:

 It is a 2-darrayofsize 'n*m 'that defines the number of resources of each type currently
allocated to each process.

 Allocation[i,j]=k means process Pi is currently allocated ‘k’ instances of resource type Rj
Need:
 Itisa2-darrayofsize ‘n*m’ that indicates the remaining resource need of each process.
 Need[i,j] =k means process Pi currently need ‘k’ instances of resource type Rj
 For its execution.
 Need[i,j] =Max[i,j]–Allocation[i,j]
Allocationi specifies the resources currently allocated to process Pi and Need specifies the additional resources that process Pimay still request to complete its task.

Banker’salgorithm consists of Safety algorithm and Resource request algorithm Safety
Algorithm
The algorithm for finding out whether or not a system is in a safe state can be described as follows:
1) Let Work and Finish be vectors of length ‘m’ and ‘n ’respectively.
Initialize: Work = Available
Finish[i]= false; for i=1,2,3,4....n
2) Find an i such that both
a) Finish[i]= false
b) Needi<=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]=true for all i
Then the system is in a safe state
Resource-Request Algorithm
Let Requesti be the request array for process Pi.Requesti[j]=k means process Piwants k
instances of resource type Rj.When a request for resources is made by process Pi,the following
actions are taken:
1) If Requesti<=Needi
Go to
step(2);otherwise,raiseanerrorcondition,sincetheprocesshasexceededitsmaximum claim.
2) If Requesti<=Available

Go to step(3);otherwise, Pimust wait, since the resources are not available.
3) HavethesystempretendtohaveallocatedtherequestedresourcestoprocessPiby modifying the state as follows:
Available = Available – Request i
Allocationi=Allocationi+Requesti
Needi= Needi– Requesti

8. Consider the following

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Answer the following question: i) Construct a need matrix. ii) Is the system in a safe state? If yes what is the safe sequence? iii) If process P1 makes a request (1 0 2) can the request be granted?

Solution:

- **Find out the need matrix**
- **Check if the system in a safe state with steps**
- **Find out the steps if process P1 makes a request (1 0 2) can the request be granted?**

Step 1: Construct the Need Matrix

The Need Matrix is calculated as:
Need[i][j]=Max[i][j]−Allocation[i][j]Need[i][j] = Max[i][j] - Allocation[i][j]Need[i][j]=Max[i][j]−Allocation[i][j]
Process Allocation (A B C) Max (A B C) Need (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 0 9 0 2 6 0 2
P3 2 1 1 2 2 2 0 1 1
P4 0 0 2 4 3 3 4 3 1
Step 2: Check if the System is in a Safe State
To check for a safe state, we follow these steps:
1. Initialize:
o Available resources: (A B C) = (3 3 2)
o Finish[] = {false, false, false, false, false} (Initially, no process has finished)
o Safe sequence = {} (Initially empty)
2. Find a process whose Need ≤ Available
P3: Need(0 1 1) ≤ Available(3 3 2) ✅
□ Allocate resources to P3, update Available:
New Available = (3+2, 3+1, 2+1) = (5 4 3)
□ Finish[P3] = true
□ Safe sequence = { P3 }

P1: Need(1 2 2) ≤ Available(5 4 3) ✅
□ Allocate resources to P1, update Available:
New Available = (5+2, 4+0, 3+0) = (7 4 3)
□ Finish[P1] = true
□ Safe sequence = { P3, P1 }
P4: Need(4 3 1) ≤ Available(7 4 3) ✅
□ Allocate resources to P4, update Available:
New Available = (7+0, 4+0, 3+2) = (7 4 5)
□ Finish[P4] = true
□ Safe sequence = { P3, P1, P4 }

P0: Need(7 4 3) ≤ Available(7 4 5) ✅
□ Allocate resources to P0, update Available:
New Available = (7+0, 4+1, 5+0) = (7 5 5)
□ Finish[P0] = true
□ Safe sequence = { P3, P1, P4, P0 }

P2: Need(6 0 2) ≤ Available(7 5 5) ✅
□ Allocate resources to P2, update Available:
New Available = (7+3, 5+0, 5+0) = (10 5 5)
□ Finish[P2] = true
□ Safe sequence = { P3, P1, P4, P0, P2 }

3. All processes can finish, so the system is in a SAFE STATE.
Safe Sequence: P3→P1→P4→P0→P2P3 → P1 → P4 → P0 → P2P3→P1→P4→P0→P2

Step 3: Check if P1’s Request (1 0 2) Can Be Granted
P1 is requesting (A B C) = (1 0 2).
We check if the request can be granted by verifying:
1. Request ≤ Available:
(1 0 2) ≤ (3 3 2) ?
Yes, so the request can be granted.
2. New Allocation & Available Resources after Granting P1’s Request:
New Allocation for P1 = (2+1, 0+0, 0+2) = (3 0 2)
New Available = (3-1, 3-0, 2-2) = (2 3 0)
New Need for P1 = (1-1, 2-0, 2-2) = (0 2 0)

3. Check for a Safe Sequence Again with Updated Values:
Available resources = (2 3 0)
Following a similar process as Step 2, if a safe sequence exists, the request is granted.

Solution

(i) Need matrix

Need = Max − allocation.

	Max (ABC)	Allocation (ABC)	Need (ABC)
P0	753	010	743
P1	322	200	122
P2	902	302	600
P3	222	211	011
P4	433	002	431

(ii) Let work = Available (3 3 2) + allocation

Need <= ~~work~~ available

P0 = 743 ≤ 332 false

P0 is not in safe state.

P1 = 122 ≤ 332 true

work = available + allocation (P1)

= (3 3 2) + (200)

= (5 3 2)

P1 is in safe state.

P2 = 600 ≤ 532 false

P2 is not in safe state.

P3 = 011 ≤ 532 true

work = 532 + 211

= 743

P3 is in safe state.

P4 = 431 ≤ 743 true

work = 743 + 002

= 745

P4 is in safe state.

P0 = 743 ≤ 745 true

work = 745 + 010

= 755

P0 is in safe state

P2 = 600 ≤ 755 true

work = 755 + 302

= 10 5 7

P2 is in safe state.

Safe sequence is P1 → P3 → P4 → P0 → P2

(iii) The request from P1 (1, 0, 2) can be granted immediately, as the system will remain in a safe state.

③ avail = avail − request;
alloc = alloc + request;
$need_i$ = $need_i$ − request;

④ check if new state is Safe or not (using Banker's algorithm).

Ex:-

	Alloc.	max	(work) Availa	need
P_0	010	753	332	743
P_1	200	322		~~122~~
P_2	302	902		600
P_3	211	222		011
P_4	002	433		431

→ If P_1 requests (1, 0, 2), determine if it can be granted immediately.

$P_1 \rightarrow R(1, 0, 2)$

need(P_1) = max − Alloc
= 322 − 200
= 122

need (P_1) = max − Alloc
= 322 − 200
= 122.

① Request ≤ need
102 ≤ 122

→ (if not less, stop → say that further cannot solve)

② Request ≤ avail
102 ≤ 332

③ avail = avail − request
⇒ 332 − 102 = 230
alloc = alloc + request.
⇒ 200 + 102 = 302
need = need − request
⇒ 122 − 102 = 020

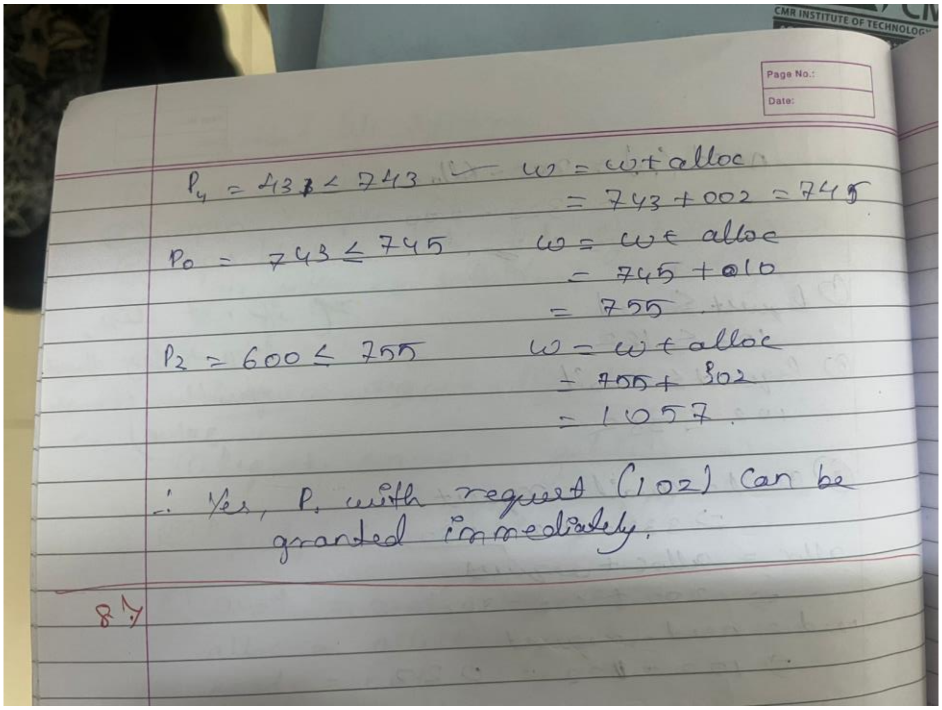
New Table:-

	Alloc.	max	Avail	Need.
P_0	010	753	230	743
P_1	302	322		020
P_2	302	902		600
P_3	211	222		011
P_4	002	433		431

Need ≤ avail

P_0 = 743 ≤ 230 ×
P_1 = 020 ≤ 230   w = w + alloc
= 230 + 302 = 532

P_2 = 600 ≤ ~~230~~ 532 ×
P_3 = 011 ≤ ~~230~~ 532 ✓
= w = w + alloc = 743
= 532 + 211



$P_4 = 431 \leq 743$ → $w = w + alloc$
$= 743 + 002 = 745$

$P_0 = 743 \leq 745$ $w = w + alloc$
$= 745 + 010$
$= 755$

$P_2 = 600 \leq 755$ $w = w + alloc$
$= 755 + 302$
$= 1057$

∴ Yes, P_1 with request (102) can be granted immediately.

8/

Final Answer
1. Need Matrix:
P0: (7 4 3)
P1: (1 2 2)
P2: (6 0 2)
P3: (0 1 1)
P4: (4 3 1)
2. Safe State:✅Yes
Safe Sequence:{ P3, P1, P4, P0, P2 }
3. Can P1's Request (1 0 2) Be Granted?✅Yes

PART V

9. Explain file allocation methods with neat diagrams

Files are usually stored on secondary storage devices such as disk. These files are then called back when needed. As part of their implementation, files must be stored in the hard disk. This has to be done in such a way that the disk space is utilized effectively and the files can be accessed quickly.

There are three methods of file allocation:

Contiguous allocation

Linked allocation

Indexed allocation

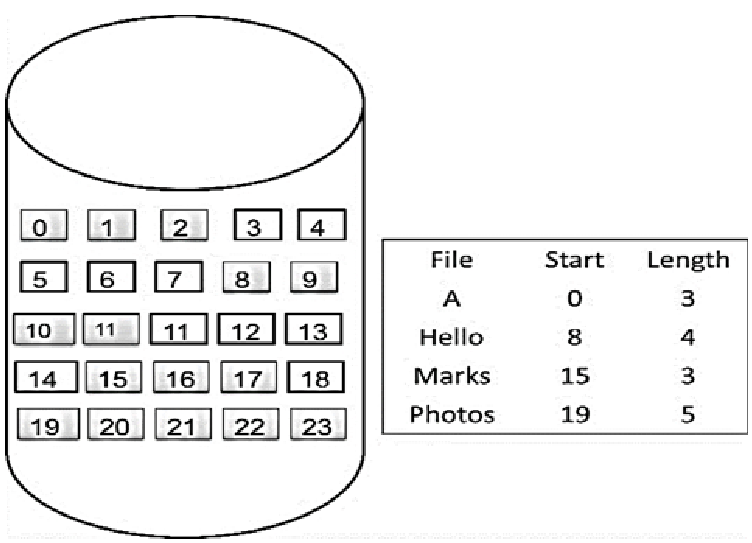
Contiguous Allocation:

a. Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk. The word contiguous means continuous

b. Because of contiguous allocation, there is minimal or no disk-head movement which reading/writing the blocks of the file.

c. The seek time is minimal over here. Consequently, access time of a file and the I/O performance is greatly improved.

d. To access a file, there we only need to know the starting location and length of the file which are stored in the directory as shown in the figure.



It has certain problems associated with it:

- Finding the space for a new file (usually done with First fit and Best Fit Algorithms)
- Problem of external fragmentation. It occurs whenever free space is broken into tiny chunks.
- Compaction (which is a solution for fragmentation) can take up lot of time and may need system to be down, wherein normal operation will not be permitted.
- Determining space to be allocated for file, especially if it needs to grow. (For e.g. the size of a Word file grows on increasing as new data is typed in)

Advantage:

• Contiguous allocation is easy to implement.

Disadvantage:

• It can be considered as a form of dynamic memory allocation, and external fragmentation may occur and compaction may be needed.

• It is difficult to estimate the file size. The size of a file may grow at run time and may be larger than the specified number of allocated blocks. In this case, the OS must move the blocks in order to provide mode space. In some systems, this is simply an error.

2. Linked Allocation:

With the linked allocation approach, disk blocks of a file are chained together with a linked-list. The directory entry of a file contains a pointer to the first block and a pointer to the last block.

To create a file, we create a new directory entry and the pointers are initialized to nil. When a write occurs, a new disk block is allocated and chained to the end of the list.

a. This method solves the problems associated with contiguous allocation.

b. Here the blocks of a single file can be scattered anywhere on the disk. The reason because the entire file is implemented as a Linked List.

c. Here every file is an element of Linked List (similar to concept of Linked List in Data Structures).

d. The directory maintained by the OS contains a pointer to the first and the last blocks of a file.

e. Each block of a file contains a pointer to the next block after it in the list.

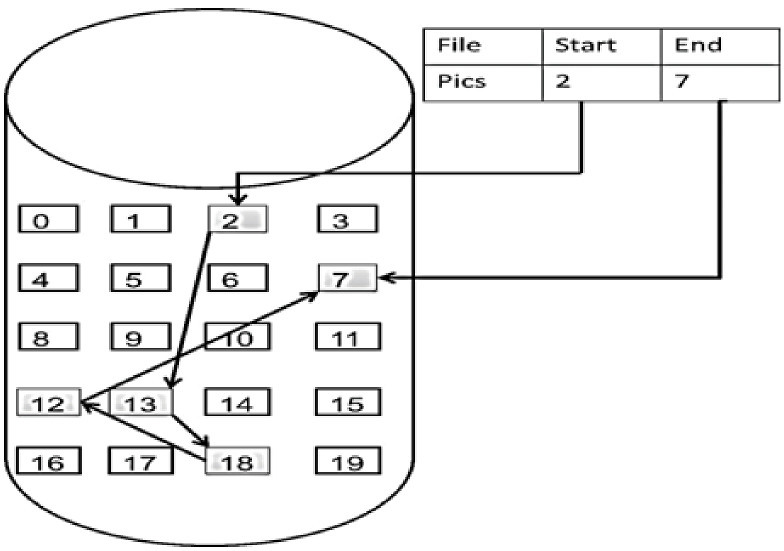
f. For creating a new file, we need to just create a new entry in the directory and not to search for sufficient space as in contiguous.

g. The free space management system allocates space to a block for writing and is then appended to the end of the List.

h. To Read a file, we need to follow the pointers from each block.

i. Advantages include no external fragmentation, size of file need not be declared at start, a file can grow as long as free blocks are available on disk.

j. Disadvantages: It works perfectly for Sequential access only, space needs to be allocated in block for pointers, error in pointer links can lead to Invalid read.



3. Indexed allocation:

Indexed Allocation With the contiguous allocation method, a user must indicate the file size before creating the file.Then, the operating system searches the disk to find contiguous disk blocks for the file.

The directory entry is easy. It contains the initial disk address of this file and the number of disk blocks.Therefore, if the initial address is b and the number of blocks is n, the file will occupy blocks b, b+1, b+2, …,b+n-1.

Each file has an index block that is an array of disk block addresses. The i-th entry in the index block points to the i-th block of the file. A file's directory entry contains a pointer to its index.

Hence, the index block of an indexed allocation plays the same role as the page table. Index allocation supports both sequential and direct access without external fragmentation.

a. In indexed allocation method, all the pointers (pointing to the next block in the Linked list) are gathered together into one location known as Index Block.

b. In the earlier method (i.e. Linked Allocation) the pointers along with the blocks were scattered across the disk and needed to be retrieved in order by visiting each block for access the file.

c. This problem gets eliminated here.

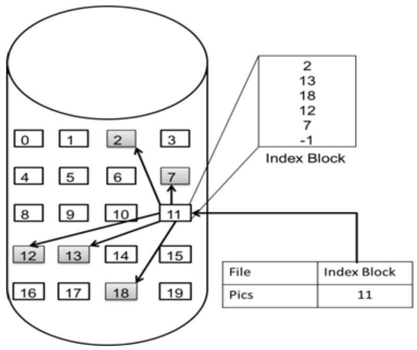
d. Each file has an index block of its own, which is an array of disk-block addresses.

e. The kth entry of the index-block is a pointer to the kth block of the file.

f. When a file is created initially, all pointers in the index block are set to null value. As new blocks are written, the pointers are modified accordingly.

g. Indexed allocation supports direct access and does not suffer from any external fragmentation.

h. Indexed allocation suffers from the problem of wasted space. E.g. if a file is made up of two blocks only, then a huge amount of space will be wasted.



10.Explain how free space management is done using i) Bit vector ii) Linked list iii) Grouping iv) Counting

The methods of free space management, such as Bit Vector, Linked List, Grouping, and Counting, are used to keep track of available and allocated blocks of free space in a storage system. Let's explore each method:

Bit Vector: The Bit Vector method represents the storage space as a sequence of bits, where each bit corresponds to a fixed-size block of storage. The bit value of 0 indicates that the block is free, while a value of 1 indicates that the block is allocated. The Bit Vector can be implemented as an array of bits or a bitmap.

• Pros: The Bit Vector method is compact and efficient in terms of space. It allows for quick checking of free or allocated blocks by directly accessing the corresponding bit.

• Cons: The Bit Vector method can suffer from wastage of space when the block size is small or not a power of 2. It also requires extra operations to find consecutive free blocks.

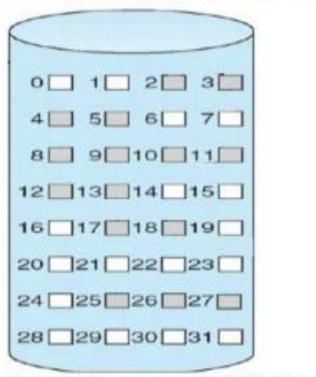


Figure 1: Bit vector method disk block

Linked List:
The Linked List method maintains a linked list of free blocks, where each block contains a pointer to the next free block. The list starts with a pointer to the first free block, and the last block in the list has a null pointer. When a block is allocated, it is removed from the free list, and when a block is deallocated, it is added back to the free list.
• Pros: The Linked List method is flexible and efficient when dealing with variable-sized blocks. It allows for efficient coalescing of adjacent free blocks.
• Cons: The Linked List method requires additional memory for maintaining the pointers, leading to overhead. Searching for a free block may involve traversing the list, resulting in increased access time.

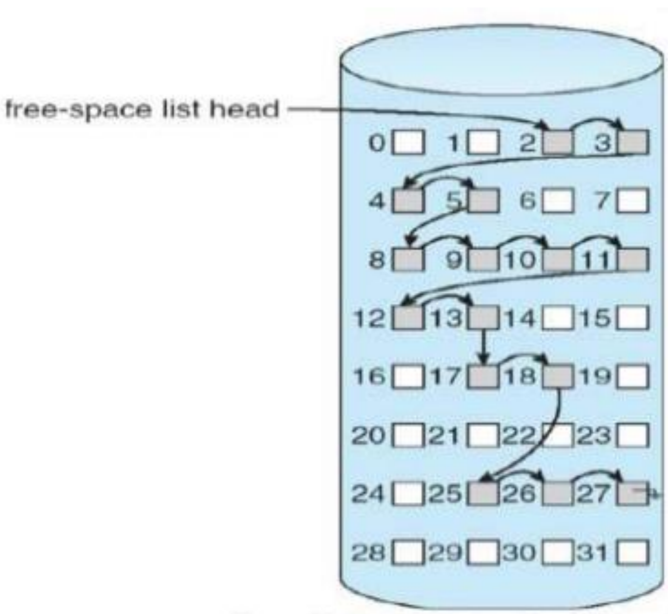


Figure 2: Linked List method of free disk blocks

Grouping: The Grouping method organizes blocks into fixed-size groups or clusters. Each group consists of a fixed number of blocks, and a bitmap or other data structure is used to track

the allocation status of each group. If a group has any free blocks, it is marked as available, and if all blocks in a group are allocated, it is marked as full.
• Pros: The Grouping method reduces the overhead of managing individual blocks by grouping them together. It provides efficient allocation and deallocation operations at the group level.
• Cons: The Grouping method may result in internal fragmentation if the block size is smaller than the group size. It requires extra operations to find a specific free block within a group.

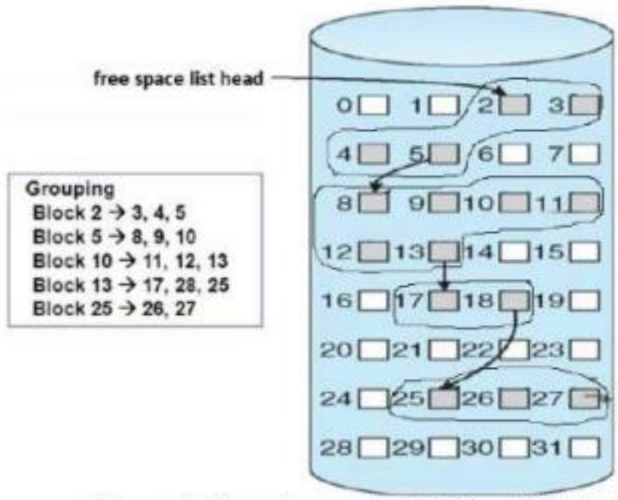


Figure 3: Grouping method of free disk blocks

Counting:
The Counting method keeps a count of the number of free blocks in the storage system. A counter is maintained that is incremented when a block is deallocated and decremented when a block is allocated. The count represents the number of available free blocks.
• Pros: The Counting method provides a simple and efficient way to track the number of free blocks. It allows for quick checking of available space without the need for additional data structures.
• Cons: The Counting method does not provide direct information about the location or size of individual free blocks. It may require additional operations to locate and allocate specific blocks.