UNIT-III

1. Explain about Memory Management Techniques.

Ans: Memory Management is the process of controlling and coordinating computer memory, assigning portions known as blocks to various running programs to optimize the overall performance of the system.

Memory management plays several roles in a computer system.

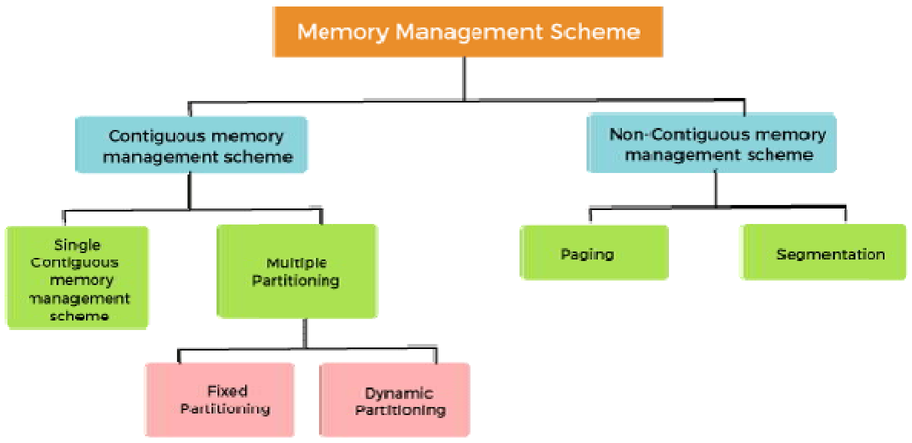
Following are the important roles in a computer system:

- Memory manager is used to keep track of the status of memory locations, whether it is free or allocated. It addresses primary memory by providing abstractions so that software perceives a large memory is allocated to it.
- Memory manager permits computers with a small amount of main memory to execute programs larger than the size or amount of available memory. It does this by moving information back and forth between primary memory and secondary memory by using the concept of swapping.
- The memory manager is responsible for protecting the memory allocated to each process from being corrupted by another process. If this is not ensured, then the system may exhibit unpredictable behavior.
- Memory managers should enable sharing of memory space between processes. Thus, two programs can reside at the same memory location although at different times.

Memory management Techniques:

The Memory management Techniques can be classified into following main categories:

- Contiguous memory management schemes
- Non-Contiguous memory management schemes


Memory Management Scheme
Contiguous memory management scheme
Non-Contiguous memory management scheme
Single Contiguous memory management scheme
Multiple Partitioning
Paging
Segmentation
Fixed Partitioning
Dynamic Partitioning

Classification of memory management schemes

Contiguous memory management schemes:

In a Contiguous memory management scheme, each program occupies a single contiguous block of storage locations, i.e., a set of memory locations with consecutive addresses.

Single contiguous memory management schemes:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition.

Advantages of Single contiguous memory management schemes:

- Simple to implement.
- Easy to manage and design.
- In a Single contiguous memory management scheme, once a process is loaded, it is given full processor's time, and no other processor will interrupt it.

Disadvantages of Single contiguous memory management schemes:

- Wastage of memory space due to unused memory as the process is unlikely to use all the available memory space.
- The CPU remains idle, waiting for the disk to load the binary image into the main memory.
- It can not be executed if the program is too large to fit the entire available main memory space.
- It does not support multiprogramming, i.e., it cannot handle multiple programs simultaneously.

Multiple Partitioning:

The single Contiguous memory management scheme is inefficient as it limits computers to execute only one program at a time resulting in wastage in memory space and CPU time. The problem of inefficient CPU use can be overcome using multiprogramming that allows more than one program to run concurrently. To switch between two processes, the operating systems need to load both processes into the main memory. The operating system needs to divide the available main memory into multiple parts to load multiple processes into the main memory. Thus multiple processes can reside in the main memory simultaneously.

The multiple partitioning schemes can be of two types:

- Fixed Partitioning
- Dynamic Partitioning

Fixed Partitioning

The main memory is divided into several fixed-sized partitions in a fixed partition memory management scheme or static partitioning. These partitions can be of the same size or different sizes. Each partition can hold a single process. The number of partitions determines the degree of multiprogramming, i.e., the maximum number of processes in memory. These partitions are made at the time of system generation and remain fixed after that.

Advantages of Fixed Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Fixed Partitioning memory management schemes:

- This scheme suffers from internal fragmentation.
- The number of partitions is specified at the time of system generation.

Dynamic Partitioning

The dynamic partitioning was designed to overcome the problems of a fixed partitioning scheme. In a dynamic partitioning scheme, each process occupies only as much memory as they require when loaded for processing. Requested processes are allocated memory until the entire physical memory is exhausted or the remaining space is insufficient to hold the requesting process. In this scheme the partitions used are of variable size, and the number of partitions is not defined at the system generation time.

Advantages of Dynamic Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Dynamic Partitioning memory management schemes:

- This scheme also suffers from internal fragmentation.
- The number of partitions is specified at the time of system segmentation.

Non-Contiguous memory management schemes:

In a Non-Contiguous memory management scheme, the program is divided into different blocks and loaded at different portions of the memory that need not necessarily be adjacent to one another. This scheme can be classified depending upon the size of blocks and whether the blocks reside in the main memory or not.

2.Explain the concept of Deadlock prevention in detail.

Ans: If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

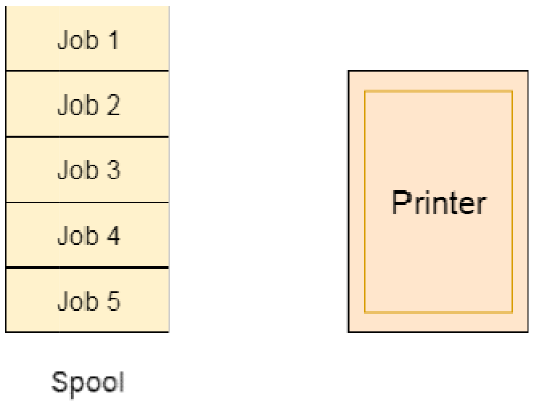
Let's see how we can prevent each of the conditions.

1. Mutual Exclusion

Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.
However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.



Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there n ay arise a race condition between the processes to get space in that spool.

We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance. Therefore, we cannot violate mutual exclusion for a process practically.

2. Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)

This can be implemented practically if a process declares all the resources initially. How ever, this sounds very practical but can't be done in the computer system because a process can't determine ne essary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.

2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

3. No Preemption

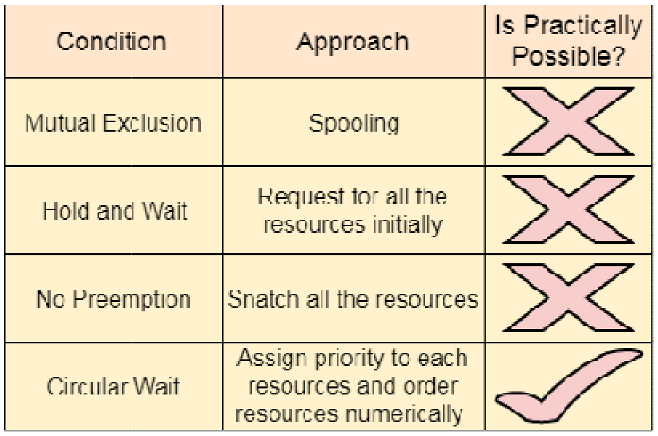
Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

4. Circular Wait

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.



Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	✗
Hold and Wait	Request for all the resources initially	✗
No Preemption	Snatch all the resources	✗
Circular Wait	Assign priority to each resources and order resources numerically	✓

Among all the methods, violating Circular wait is the only approach that can be implemented practically.

3. What is Deadlock. Explain about Characteristics of Deadlocks and explain about methods of Handling of Deadlocks?

Ans:Deadlock is a situation where a process or a set of processes is blocked, waiting for some other resource that is held by some other waiting process.

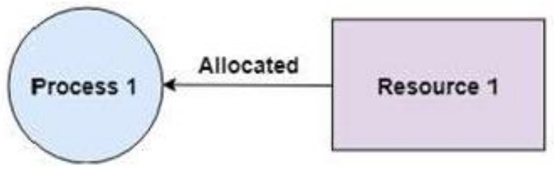
Characteristics of Deadlock:-

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive. They are given as follows −

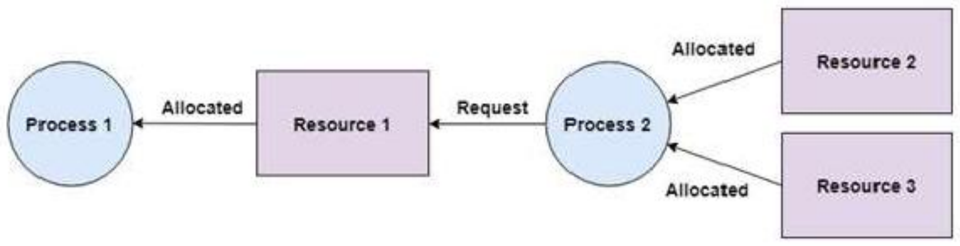
Mutual Exclusion

There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



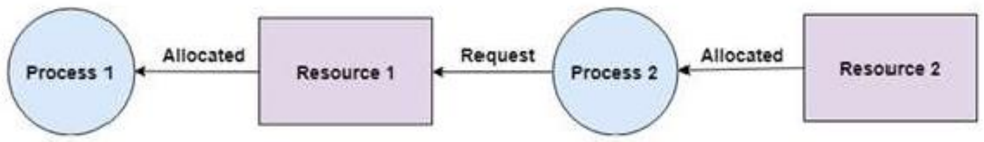
Hold and Wait

A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



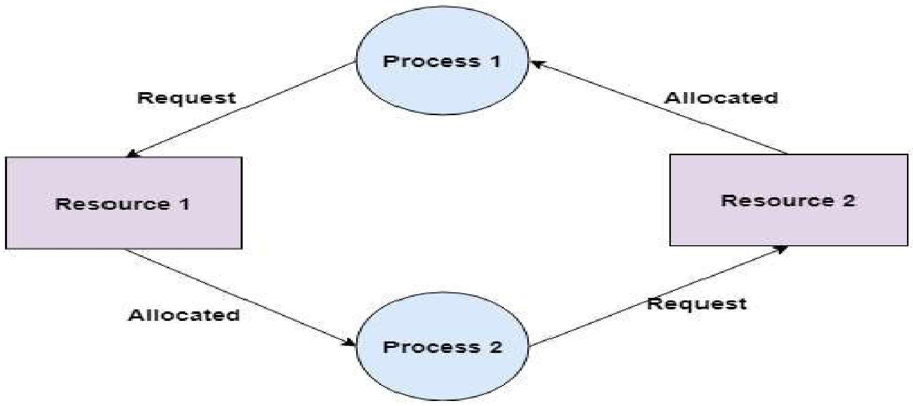
No Preemption

A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



Circular Wait

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Methods of Deadlock:-

Deadlock detection, deadlock prevention and deadlock avoidance are the main method for handling deadlocks. Details about these are given as follows −

Deadlock Detection

Deadlock can be detected by the resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be handed using the given methods −

- All the processes that are involved in the deadlock are terminated. This approach is not that useful as all the progress made by the processes is destroyed.
- Resources can be preempted from some processes and given to others until the deadlock situation is resolved.

Deadlock Prevention

It is important to prevent a deadlock before it can occur. So, the system checks each transaction before it is executed to make sure it does not lead to deadlock. If there is even a slight possibility that a transaction may lead to deadlock, it is never allowed to execute.

Some deadlock prevention schemes that use timestamps in order to make sure that a deadlock does not occur are given as follows −

Wait - Die Scheme

- In the wait - die scheme, if a transaction T1 requests for a resource that is held by transaction T2, one of the following two scenarios may occur −
    - TS(T1) < TS(T2) - If T1 is older than T2 i.e T1 came in the system earlier than T2, then it is allowed to wait for the resource which will be free when T2 has completed its execution.
    - TS(T1) > TS(T2) - If T1 is younger than T2 i.e T1 came in the system after T2, then T1 is killed. It is restarted later with the same timestamp.

Wound - Wait Scheme

- In the wound - wait scheme, if a transaction T1 requests for a resource that is held by transaction T2, one of the following two possibilities may occur −

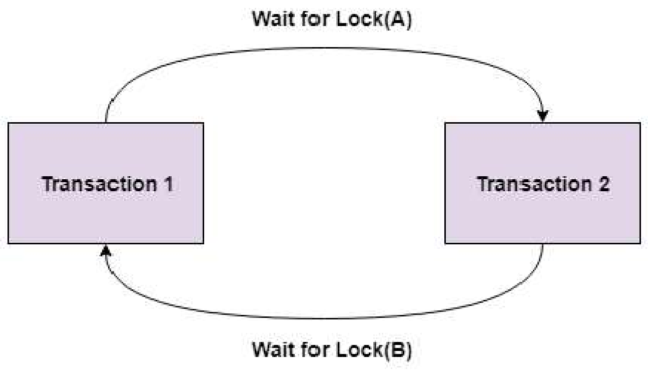
- TS(T1) < TS(T2) - If T1 is older than T2 i.e T1 came in the system earlier than T2, then it is allowed to roll back T2 or wound T2. Then T1 takes the resource and completes its execution. T2 is later restarted with the same timestamp.
- TS(T1) > TS(T2) - If T1 is younger than T2 i.e T1 came in the system after T2, then it is allowed to wait for the resource which will be free when T2 has completed its ex cution.

Deadlock Avoidance

It is better to avoid a deadlock rather than take measures after the deadlock has occurred. The wait for graph can be used for deadlock avoidance. This is however only useful for smaller databases as it can get quite complex in larger databases.

Wait for graph

The wait for graph shows the relationship between the resources and transactions. If a transaction requests a resource or if it already holds a resource, it is visible as an edge on the wait for graph. If the wait for graph contains a cycle, then there may be a deadlock in the system, otherwise not.



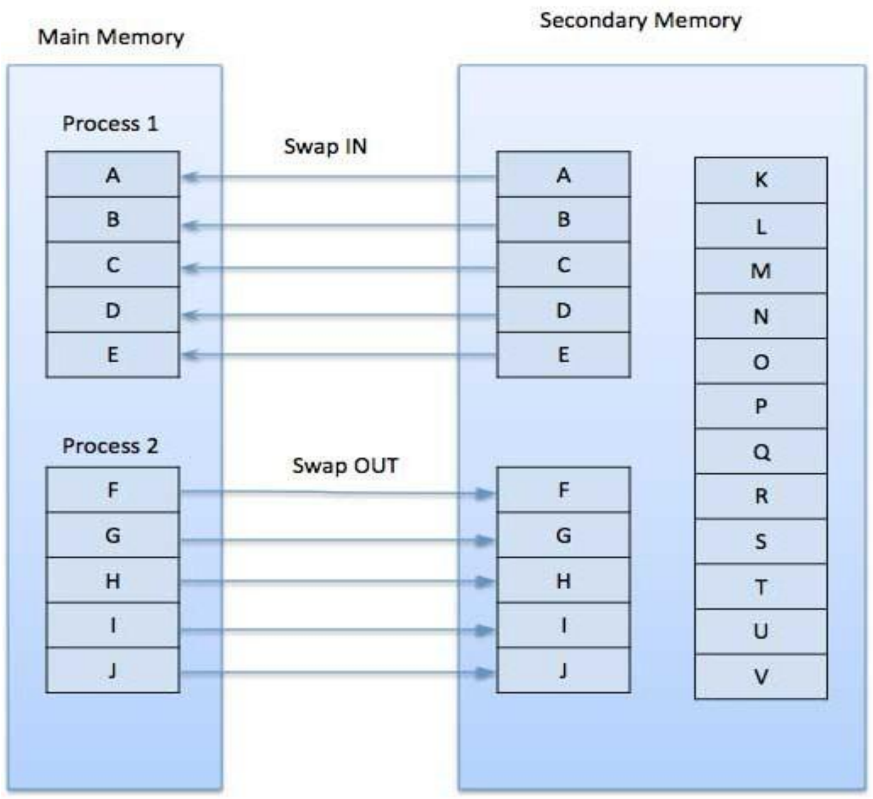
Ostrich Algorithm

The ostrich algorithm means that the deadlock is simply ignored and it is assumed that it will never occur. This is done because in some systems the cost of handling the deadlock is much higher than simply ignoring it as it occurs very rarely. So, it is simply assumed that the deadlock will never occur and the system is rebooted if it occurs by any chance.

4.Explain about Demand paging and Page Replacement algorithms?

Ans: Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging –

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

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

Page Replacement Algorithms

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

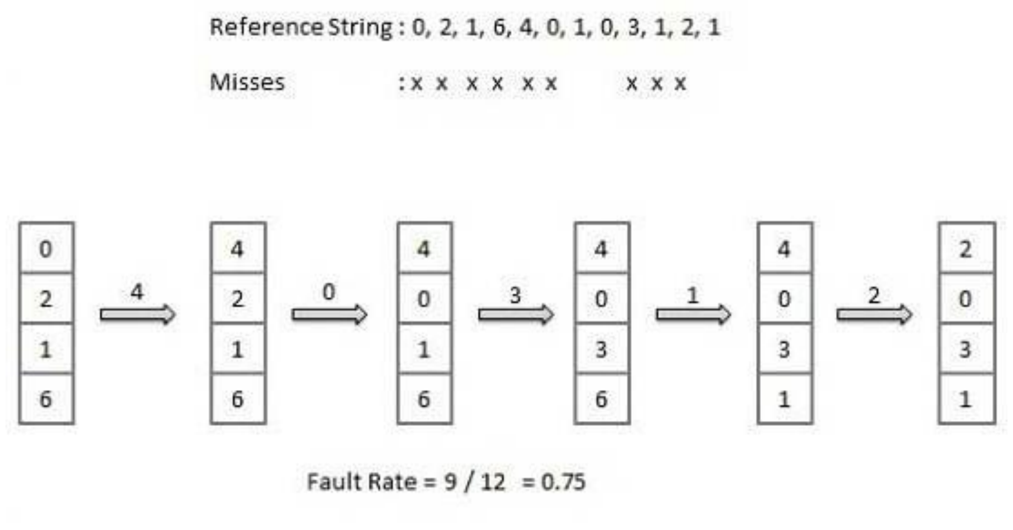
Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

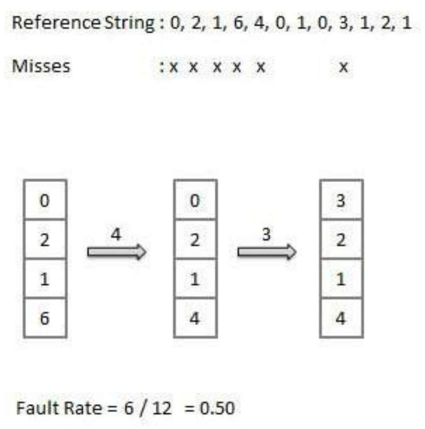
First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.


Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses :x x x x x x x x x
0 2 1 6
4
4 2 1 6
0
4 0 1 6
3
4 0 3 6
1
4 0 3 1
2
2 0 3 1
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.



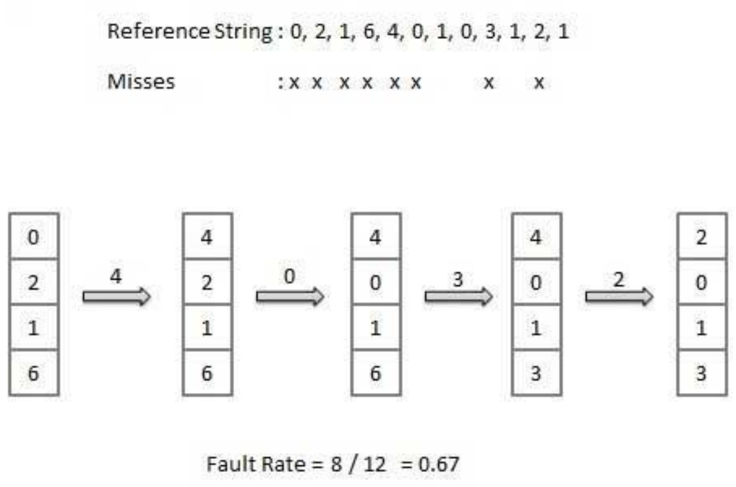
Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x

Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x

Fault Rate = 8 / 12 = 0.67

Page Buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used(LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used(MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

UNIT-IV

1. Explain

A) Overview of Mass Storage Structure?

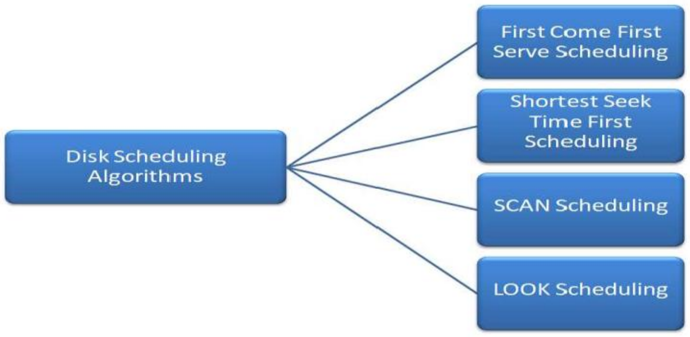
Ans:- Disks are the mainly used mass storage devices. They provide the bulk of secondary storage in operating systems today.

Disk Structure

Each modern disk contains concentric tracks and each track is divided into multiple sectors. The disks are usually arranged as a one dimensional array of blocks, where blocks are the smallest storage unit.Blocks can also be called as sectors. For each surface of the disk, there is a read/write desk available. The same tracks on all the surfaces is known as a cylinder.

Disk Scheduling

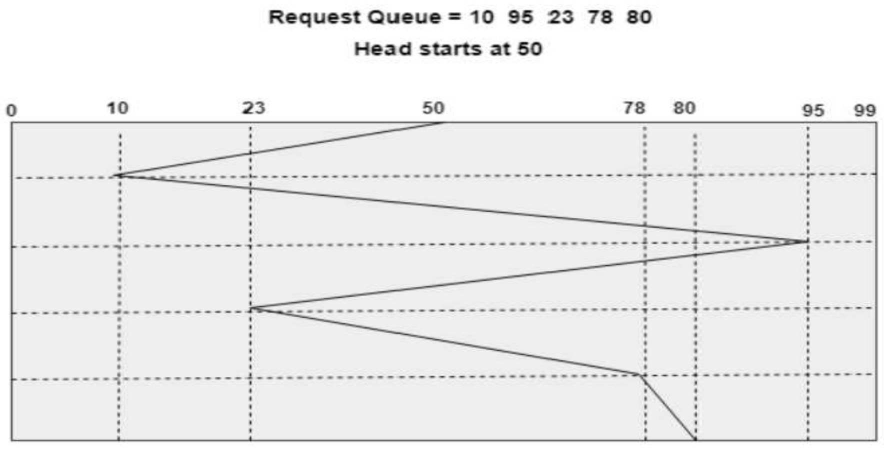
There are many disk scheduling algorithms that provide the total head movement for various requests to the disk. Here are the types of disk scheduling


Disk Scheduling Algorithms
First Come First Serve Scheduling
Shortest Seek Time First Scheduling
SCAN Scheduling
LOOK Scheduling

All these algorithms are explained using the following requests for the disk –

First Come First Serve Scheduling (FCFS)

In first come first served scheduling, the requests are serviced in their coming order. This algorithm is fair as it allows all requests a chance but it does not provide the fastest possible service. An example of FCFS scheduling is given below –

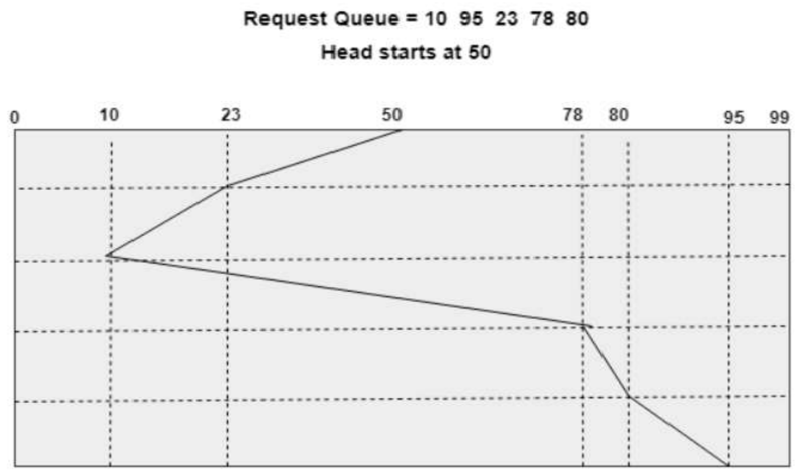


FCFS Disk Scheduling

In the above example, the requests are serviced in the order they appear i.e 10, 95, 23, 78, 80. The seek head is initially positioned at 50 and it starts from there.

Shortest Seek Time First Scheduling

The requests that are closest to the current head are served first before moving away in shortest seek time first scheduling algorithm. A problem with SSTF algorithm is that it may cause starvation for some requests. An example of Shortest Seek Time First Scheduling is given below −

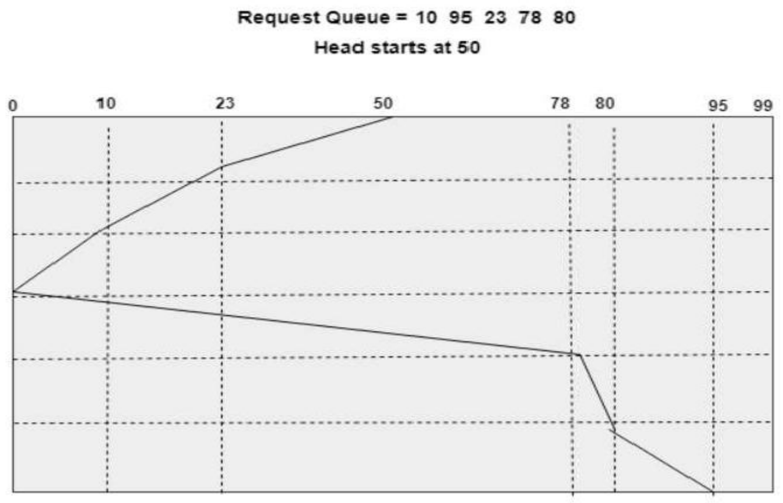


SSTF Disk Scheduling

In the above example, the requests are serviced in the order 23, 10, 78, 80, 95. The seek head is initially positioned at 50 and it starts from there. 23 is closest to 50 so it is services first. Then 10 is closer to 23 than 78 so it is services next. After this 78, 80 and 95 are serviced.

SCAN Scheduling

In this scheduling algorithm, the head moves towards one direction while servicing all the requests in that direction until it reaches the end of the disk. After that it starts moving towards the other direction. In this way, the head continuously scans back and forth across the disk. An example of SCAN scheduling is given below −

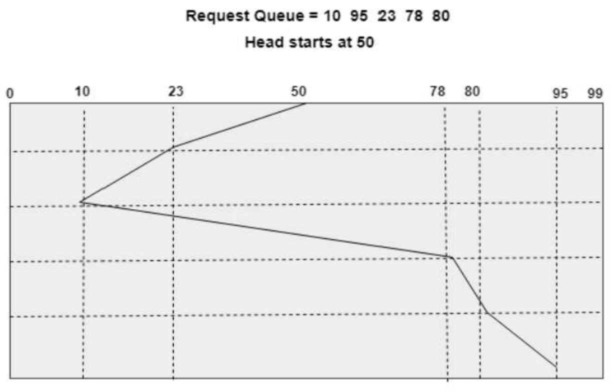


SCAN Disk Scheduling

In the above example, the requests are serviced in the order 23, 10, 78, 80, 95. The head is initially at 50 and moves towards the left while servicing requests 23 and 10. When it reaches the end of the disk, it starts moving right and services 78, 80 and 95 as they occur.

LOOK Scheduling

LOOK scheduling algorithm is similar to SCAN scheduling but is its practical version. In this algorithm, the head moves towards one direction while servicing all the requests in that direction until it reaches the last request. After that it starts moving towards the other direction. An example of LOOK scheduling is given below



LOOK Disk Scheduling

In the above example, the requests are serviced in the order 23, 10, 78, 80, 95. The head is initially at 50 and moves towards the left while servicing requests 23 and 10. When it reaches the last request on the left i.e. 10, it starts moving right and services 78, 80 and 95 as they occur.

2. Explain in detail about file allocation strategies.**Ans: File Allocation Methods**

There are different kinds of methods that are used to allocate disk space. We must select the best method for the file allocation because it will directly affect the system performance and system efficiency. With the help of the allocation method, we can utilize the disk, and also files can be accessed.
There are various types of file allocations method:

1. Contiguous allocation
2. Extents
3. Linked allocation
4. Clustering
5. FAT
6. Indexed allocation
7. Linked Indexed allocation
8. Multilevel Indexed allocation
9. Inode

There are different types of file allocation methods, but we mainly use three types of file allocation methods:

1. Contiguous allocation

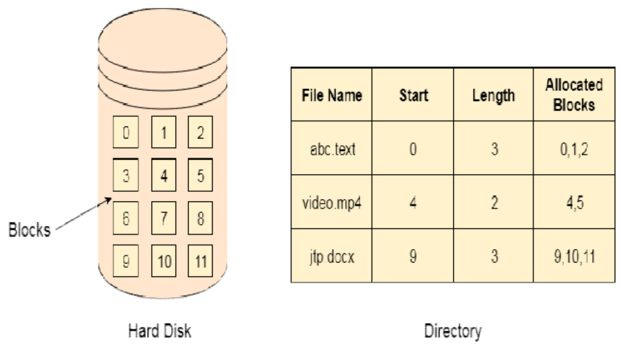
2. Linked list allocation
3. Indexed allocation

These methods provide quick access to the file blocks and also the utilization of disk space in an efficient manner.

Contiguous Allocation

If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.

In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table

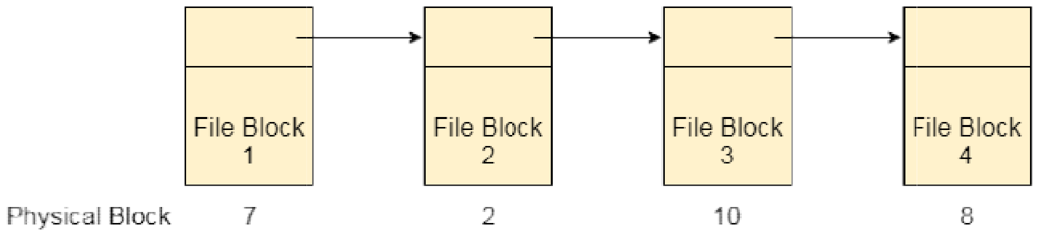


File Name	Start	Length	Allocated Blocks
abc.text	0	3	0,1,2
video.mp4	4	2	4,5
jtp.docx	9	3	9,10,11

Contiguous Allocation

Linked List Allocation

Linked List allocation solves all problens of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.

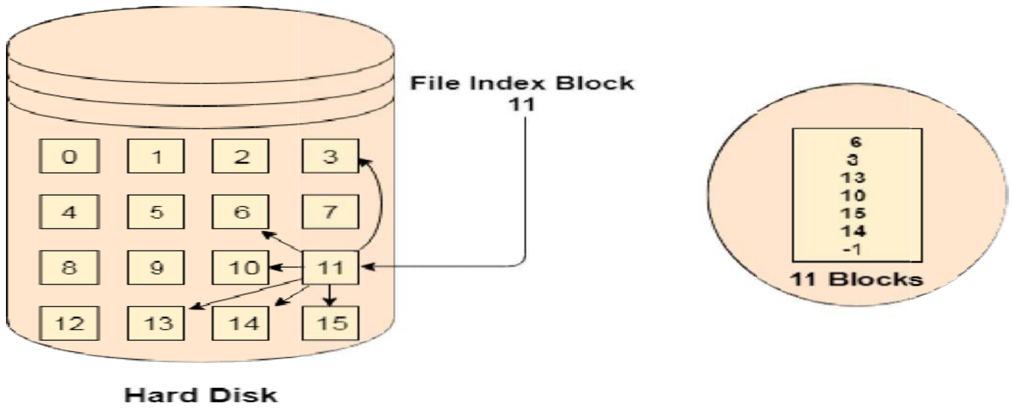


Linked List Allocation

Indexed Allocation

Indexed Allocation Scheme

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



3. Explain File access methods.

Ans: When a file is used, information is read and accessed into computer memory and there are several ways to access this information of the file. Some systems provide only one access method for files. Other systems, such as those of IBM, support many access methods, and choosing the right one for a particular application is a major design problem.

There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.

1. **Sequential Access –**
   It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.
   Read and write make up the bulk of the operation on a file. A read operation *-read next-* read the next position of the file and automatically advance a file pointer, which keeps track I/O location. Similarly, for the write*write next* append to the end of the file and advance to the newly written material.
   Key points:
   - Data is accessed one record right after another record in an order.
   - When we use read command, it move ahead pointer by one
   - When we use write command, it will allocate memory and move the pointer to the end of the file
   - Such a method is reasonable for tape.

2. **Direct Access –**
   Another method is *direct access method* also known as *relative access method*. A filed-length logical record that allows the program to read and write record rapidly. in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record.

Thus, we may read block 14 then block 59 and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.
A block number provided by the user to the operating system is normally a *relative block number*, the first relative block of the file is 0 and then 1 and so on.

3. **Index sequential method –**
   It is the other method of accessing a file which is built on the top of the sequential access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index and then by the help of pointer we access the file directly.
   Key points:
   - It is built on top of Sequential access.
   - It control the pointer by using index.

UNIT-V

1.Explain Goals and Principles of Protection?

Ans: **Protection in Operating System**

Protection is especially important in a multiuser environment when multiple users use computer resources such as CPU, memory, etc. It is the operating system's responsibility to offer a mechanism that protects each process from other processes. In a multiuser environment, all assets that require protection are classified as objects, and those that wish to access these objects are referred to as subjects. The operating system grants different 'access rights' to different subjects.

In this article, you will learn the protection in the operating system with its needs, goals, and authentication.

What is Protection in Operating System?

A mechanism that controls the access of programs, processes, or users to the resources defined by a computer system is referred to as protection. You may utilize protection as a tool for multi-programming operating systems, allowing multiple users to safely share a common logical namespace, including a directory or files.

It needs the protection of computer resources like the software, memory, processor, etc. Users should take protective measures as a helper to multiprogramming OS so that multiple users may safely use a common logical namespace like a directory or data. Protection may be achieved by maintaining confidentiality, honesty and availability in the OS. It is critical to secure the device from unauthorized access, viruses, worms, and other malware.

Need of Protection in Operating System

Various needs of protection in the operating system are as follows:

1. There may be security risks like unauthorized reading, writing, modification, or preventing the system from working effectively for authorized users.

2. It helps to ensure data security, process security, and program security against unauthorized user access or program access.
3. It is important to ensure no access rights' breaches, no viruses, no unauthorized access to the existing data.
4. Its purpose is to ensure that only the systems' policies access programs, resources, and data.

Goals of Protection in Operating System

Various goals of protection in the operating system are as follows:

1. The policies define how processes access the computer system's resources, such as the CPU, memory, software, and even the operating system. It is the responsibility of both the operating system designer and the app programmer. Although, these policies are modified at any time.
2. Protection is a technique for protecting data and processes from harmful or intentional infiltration. It contains protection policies either established by itself, set by management or imposed individually by programmers to ensure that their programs are protected to the greatest extent possible.
3. It also provides a multiprogramming OS with the security that its users expect when sharing common space such as files or directories.

Role of Protection in Operating System

Its main role is to provide a mechanism for implementing policies that define the use of resources in a computer system. Some rules are set during the system's design, while others are defined by system administrators to secure their files and programs.

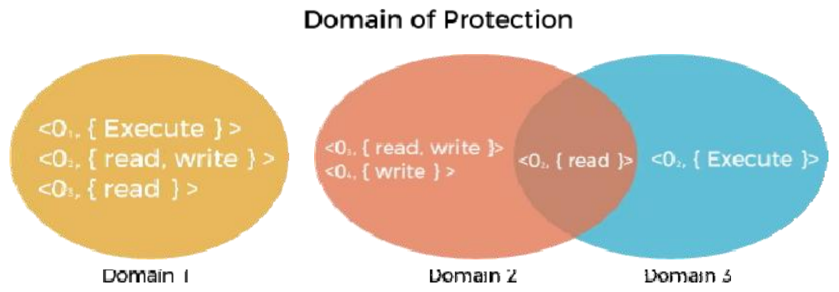
Every program has distinct policies for using resources, and these policies may change over time. Therefore, system security is not the responsibility of the system's designer, and the programmer must also design the protection technique to protect their system against infiltration.

Domain of Protection

Various domains of protection in operating system are as follows:

1. The protection policies restrict each process's access to its resource handling. A process is obligated to use only the resources necessary to fulfil its task within the time constraints and in the mode in which it is required. It is a process's protected domain.

2. Processes and objects are abstract data types in a computer system, and these objects have operations that are unique to them. A dom ain component is defined as **<object, {set of o erations on object}>**.



3. Each domain comprises a collection of objects and the operations that may be implemented on them. A domain could be made up of only one process, procedure, or user. If a domain is linked with a procedure, changing the domain would mean changing the procedure ID. Objects may share one or more common operations.

2. .Explain about the Swap space management.

wapping is a memory management technique used in multi-programming to increase the number of processes sharing the CPU. It is a technique of removing a process from the main memory and storing it into secondary memory, and then bringing it back into the main memory for continued execution. This action of moving a process out from main memory to secondary memory is called **Swap Out** and the action of moving a process out from secondary memory to main memory is called **Swap In**.

Swap-space management is a technique used by operating systems to optimize memory usage and improve system performance. Here are some advantages and disadvantages of swap-space management:

Advantages:

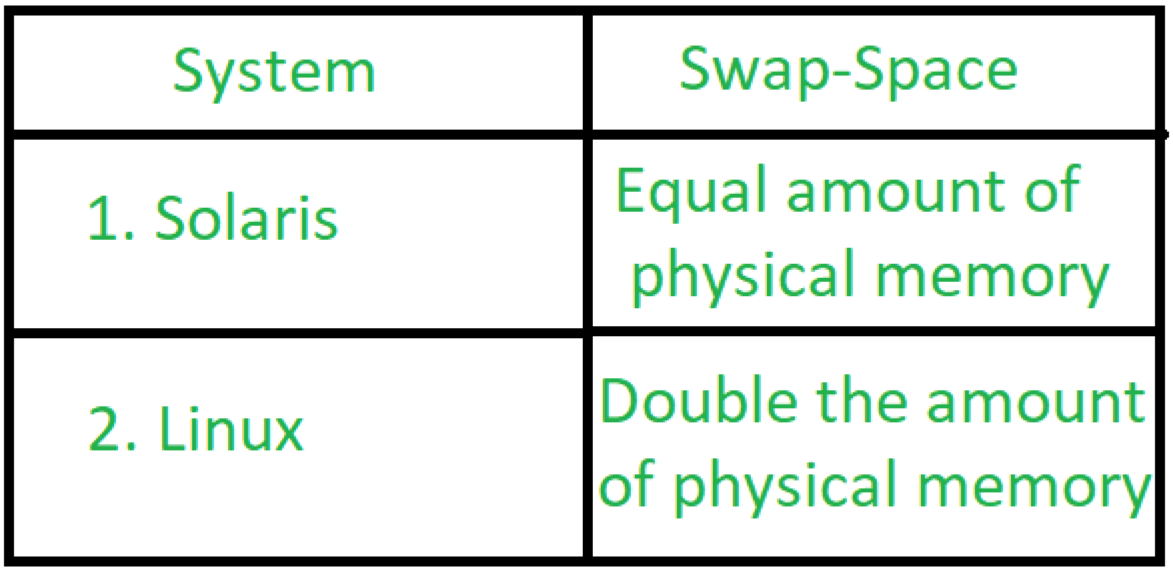
1. Increased memory capacity: Swap-space management allows the operating system to use hard disk space as virtual memory, effectively increasing the available memory capacity.
2. Improved system performance: By using virtual memory, the operating system can swap out less frequently used data from physical memory to disk, freeing up space for more frequently used data and improving system performance.
3. Flexibility: Swap-space management allows the operating system to dynamically allocate and deallocate memory as needed, depending on the demands of running applications.

Disadvantages:

Slower access times: Accessing data from disk is slower than accessing data from physical memory, which can result in slower system performance if too much swapping is required.
Increased disk usage: Swap-space management requires disk space to be reserved for use as virtual memory, which can reduce the amount of available space for other data storage purposes.
Risk of data loss: In some cases, if there is a problem with the swap file, such as a disk error or corruption, data may be lost or corrupted.
Overall, swap-space management is a useful technique for optimizing memory usage and improving system performance. However, it is important to carefully manage swap space allocation and monitor system performance to ensure that excessive swapping does not negatively impact system performance.

Swap-Space :
The area on the disk where the swapped-out processes are stored is called swap space.
Swap-Space Management :
Swap-Space management is another low-level task of the operating system. Disk space is used as an extension of main memory by the virtual memory. As we know the fact that disk access is much slower than memory access, In the swap-space management we are using disk space, so it will significantly decreases system performance. Basically, in all our systems we require the best throughput, so the goal of this swap-space implementation is to provide the virtual memory the best throughput. In these article, we are going to discuss how swap space is used, where swap space is located on disk, and how swap space is managed.
Swap-Space Use :
Swap-space is used by the different operating-system in various ways. The systems which are implementing swapping may use swap space to hold the entire process which may include image, code and data segments. Paging systems may simply store pages that have been pushed out of the main memory. The need of swap space on a system can vary from a megabytes to gigabytes but it also depends on the amount of physical memory, the virtual memory it is backing and the way in which it is using the virtual memory.
It is safer to overestimate than to underestimate the amount of swap space required, because if a system runs out of swap space it may be forced to abort the processes or may crash entirely. Overestimation wastes disk space that could otherwise be used for files, but it does not harm other.

Following table shows different system using amount of swap space:



System	Swap-Space
1. Solaris	Equal amount of physical memory
2. Linux	Double the amount of physical memory

Figure – Different systems using amount of swap-space
Explanation of above table :
Solaris, setting swap space equal to the amount by which virtual memory exceeds page-able physical memory. In the past Linux has suggested setting swap space to double the amount of physical memory. Today, this limitation is gone, and most Linux systems use considerably less swap space.
Including Linux, some operating systems; allow the use of multiple swap spaces, including both files and dedicated swap partitions. The swap spaces are placed on the disk so the load which is on the I/O by the paging and swapping will spread over the system's bandwidth.

Swap-Space Location :

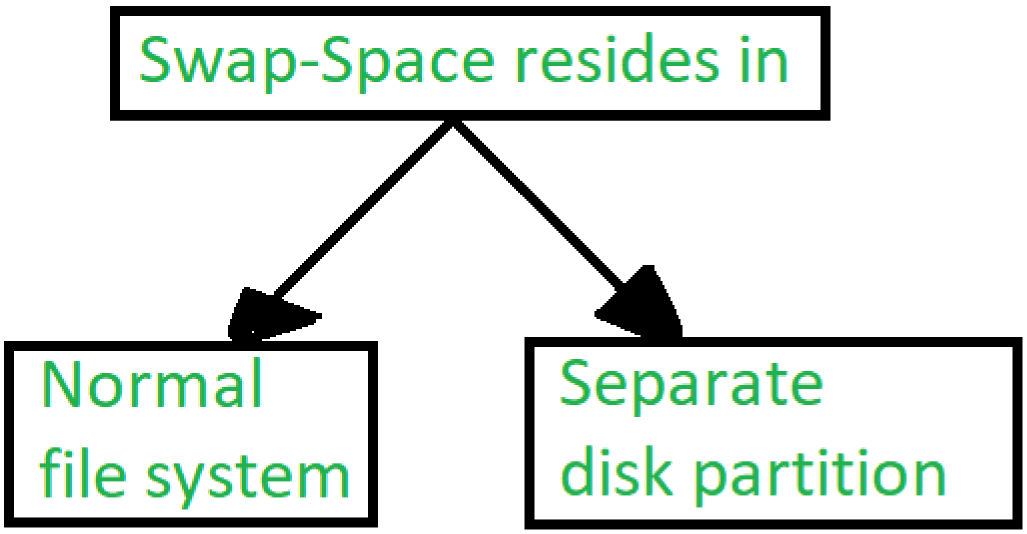


Figure – Location of swap-space

A swap space can reside in one of the two places –

1. Normal file system
2. Separate disk partition

Let, if the swap-space is simply a large file within the file system. To create it, name it and allocate its space **normal file-system** routines can be used. This approach, through easy to implement, is inefficient. Navigating the directory structures and the disk-allocation data structures takes time and extra disk access. During reading or writing of a process image, **external fragmentation** can greatly increase swapping times by forcing multiple seeks.

There is also an alternate to create the swap space which is in a separate **raw partition**. There is no presence of any file system in this place. Rather, a swap space storage manager is used to allocate and de-allocate the blocks. from the raw partition. It uses the algorithms for speed rather than storage efficiency, because we know the access time of swap space is shorter than the file system. By this **Internal fragmentation** increases, but it is acceptable, because the life span of the swap space is shorter than the files in the file system. Raw partition approach creates fixed amount of swap space in case of the **disk partitioning**.

Some operating systems are flexible and can swap both in raw partitions and in the file system space, example: **Linux**.

<u>Swap-Space Management: An Example</u> –

The traditional UNIX kernel started with an implementation of swapping that copied entire process between contiguous disk regions and memory. UNIX later evolve to a combination of swapping and paging as paging hardware became available. In Solaris, the designers changed standard UNIX methods to improve efficiency. More changes were made in later versions of Solaris, to improve the efficiency.

Linux is almost similar to Solaris system. In both the systems the swap space is used only for anonymous memory, it is that kind of memory which is not backed by any file. In the Linux system, one or more swap areas are allowed to be established. A swap area may be in either in a swap file on a regular file system or a dedicated file partition.

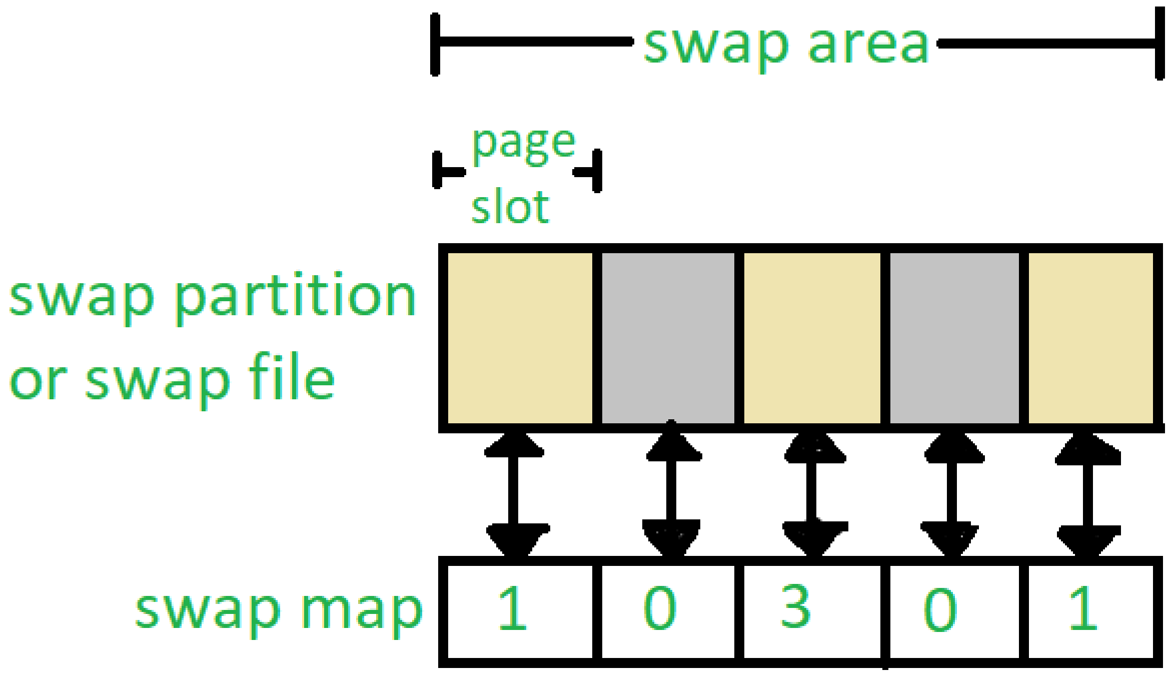


Figure – Data structure for swapping on Linux system
Each swap area consists of 4-KB **page slots**, which are used to hold the swapped pages. Associated with each swap area is a **swap-map-** an array of integers counters, each corresponding to a page slot in the swap area. If the value of the counter is 0 it means page slot is occupied by swapped page. The value of counter indicates the number of mappings to the swapped page. For example, a value 3 indicates that the swapped page is mapped to the 3 different processes.

3.Explain in detail about RAID structures and different levels.

Ans: Redundant Array of Independent Disk (RAID) combines multiple small, inexpensive disk drives into an array of disk drives which yields performance more than that of a Single Large Expensive Drive (SLED). RAID is also called Redundant Array of Inexpensive Disks.

Storing the same data in different disk increases the fault-tolerance.

The array of Mean Time Between Failure (MTBF) = MTBF of an individual drive, which is divided by the number of drives in the array. Because of this reason, the MTBF of an array of drives are too low for many application requirements.

Types of RAID

The various types of RAID are explained below −

RAID-0

RAID Level-0 is not redundant. Since no redundant information is stored, performance is very good, but the failure of any disk in the array results in data loss. A single record is divided into strips typically 512 bytes and is stored across all disks. The record can be accessed quickly by reading all disks at the same time, called as striping.

RAID-1

RAID Level-1 provides redundancy by writing all data into two or more drives. The performance is faster on reads and slower on writes compared to a single drive. If anyone drives fails, no data is lost. This method is called mirroring.

RAID-2

RAID Level-2 is used for Hamming error correction codes and is used with drives which do not have built-in error detection.

RAID-3

RAID Level-3 stripes data at a byte level across several drives, with parity stored on one drive. Byte-level stripping hardware supports efficient use.

RAID-4

RAID Level-4 that stripes data at a block level across several drives, with parity stored on one drive. Parity information allows recovery from the failure of any single drive. The performance of the level-4 array is good for reads.

Writes, however, require that parity data be updated each time. Because only one drive in the array stores redundant data. The cost per megabyte is low.

RAID-5

RAID Level-5 is similar to level 4, but distributes parity among the drives. This can speed up small writes in the multiprocessing system. The performance for reads is lower than a level-4 array. The cost per megabyte is the same as level-4.

Levels	**Summary**
RAID-0	It is the fastest and most efficient array type but offers no fault-tolerance.
RAID-1	It is the array of choice for a critical, fault tolerant environment.
RAID-2	It is used today because ECC is embedded in almost all modern disk drives.
RAID-3	It is used in single environments which access long sequential records to speed up data transfer.
RAID-4	It offers no advantages over RAID-5 and does not support multiple simultaneous write operations.
RAID-5	It is the best choice in a multi-user environment. However, at least three drives are required for the RAID-5 array.