**OS**

**UNIT V**

**I/O and File Management**

**I/O Management**

/O Requests are managed by Device Drivers in collaboration with some system programs inside the I/O device. The requests are served by OS using three simple segments : I/O Traffic Controller : Keeps track of the status of all devices, control units, and communication channels

File management is defined as the process of manipulating files in computer system, it management includes the process of creating, modifying and deleting the files

The primary role of the operating system in computer Input / Output is to manage and organize I/O operations and all I/O devices. In this chapter, you will learn about the various uses of input output devices concerning the operating system.

**File Management system and its function**

What is a file management system? File management is the process of administering a system that correctly handles digital data. Therefore, an effective file management system improves the overall function of a business workflow. It also organizes important data and provides a searchable database for quick retrieval.

**I/O Devices**

I/O devices are the pieces of hardware used by a human (or other system) to communicate with a computer. For instance, a keyboard or computer mouse is an input device for a computer, while monitors and printers are output devices.

**Examples of input/output devices**

* CD-R/RW, DVD, and Blu-ray drive.
* Digital camera.
* Fax machine.
* Floppy diskette drive.
* Hard drives.
* Modem.
* NIC (network interface card)
* SD Card.
* [Sound card](https://www.computerhope.com/jargon/s/souncard.htm)
* [Touch screen](https://www.computerhope.com/jargon/t/toucscre.htm)
* [USB flash drive](https://www.computerhope.com/jargon/j/jumpdriv.htm)

**What are scheduling policies?**

**Disk scheduling**

* Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling. Disk scheduling is important because: Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
* Two or more request may be far from each other so can result in greater disk arm movement.
* Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let’s have a quick look at some of the important terms:

* Seek Time:Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
* Rotational Latency: Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
* Transfer Time: Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
* Disk Access Time: Disk Access Time is:

Disk Access Time = Seek Time +

Rotational Latency +

Transfer Time

* Disk Response Time: Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time*is the response time of the all requests. *Variance Response Time*is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

Why do we need disk scheduling algorithms?

Disk Scheduling Algorithms are needed because a process can make multiple I/O requests and multiple processes run at the same time. The requests made by a process may be located at different sectors on different tracks. Due to this, the seek time may increase more.

The main aim of disk scheduling algorithms is to reduce or minimize the seek time for a set of requests.

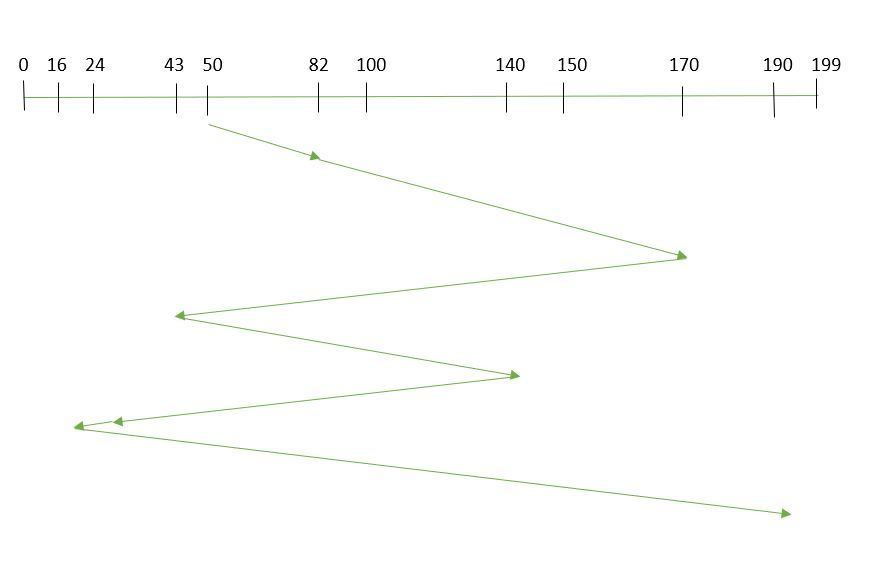
A scheduling policy is a set of rules and objectives that guides the schedule optimizer in its decisions. Use scheduling policies to promote or de-emphasize factors like business priorities, travel time, and customer preferences.

**Disk Scheduling Algorithms**

1. **FCFS**: FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)  
And current position of Read/Write head is : 50 



Advantages: 

* Every request gets a fair chance
* No indefinite postponement

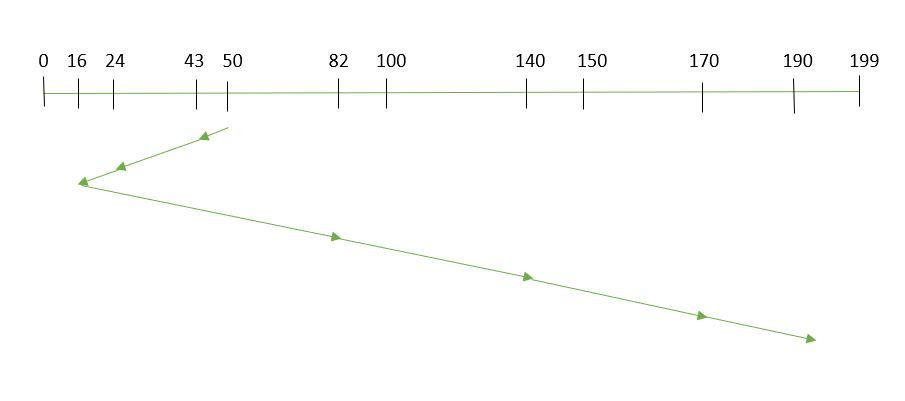
Disadvantages: 

* Does not try to optimize seek time
* May not provide the best possible service

1. **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system. Let us understand this with the help of an example.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)  
And current position of Read/Write head is : 50 



So, total seek time:

=(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170)   
=208

Advantages: 

* Average Response Time decreases
* Throughput increases

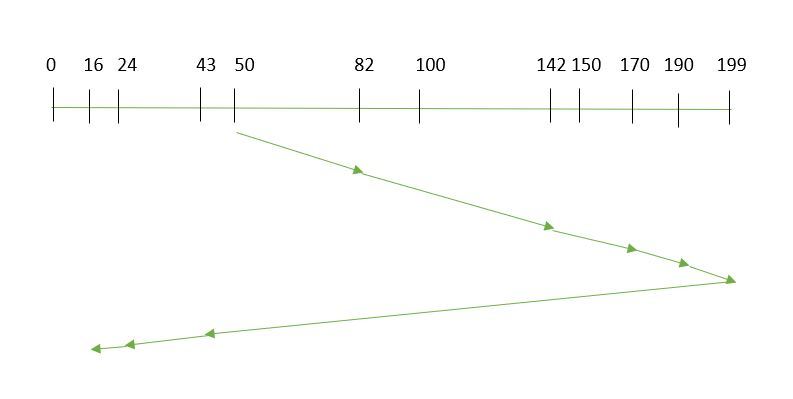
Disadvantages: 

* Overhead to calculate seek time in advance
* Can cause Starvation for a request if it has higher seek time as compared to incoming requests
* High variance of response time as SSTF favours only some requests

1. **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”. 



Therefore, the seek time is calculated as:

=(199-50)+(199-16)   
=332

Advantages: 

* High throughput
* Low variance of response time
* Average response time

Disadvantages: 

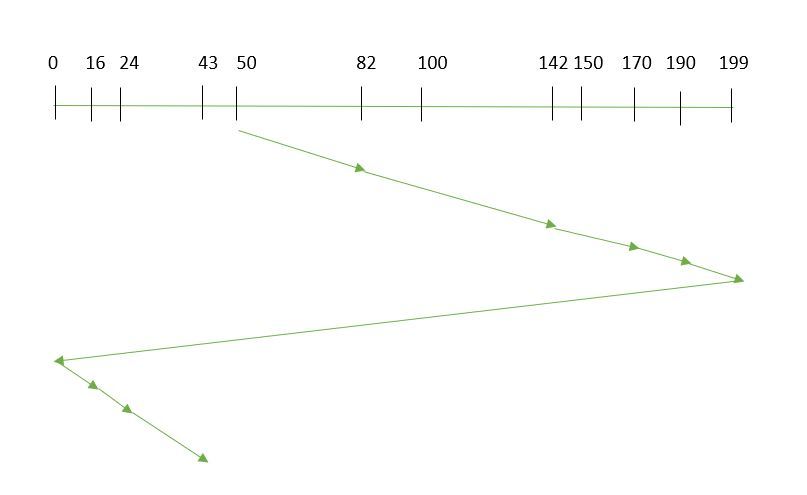
* Long waiting time for requests for locations just visited by disk arm

1. **CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

These situations are avoided in *CSCAN*algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”. 



Seek time is calculated as:

=(199-50)+(199-0)+(43-0)   
=391

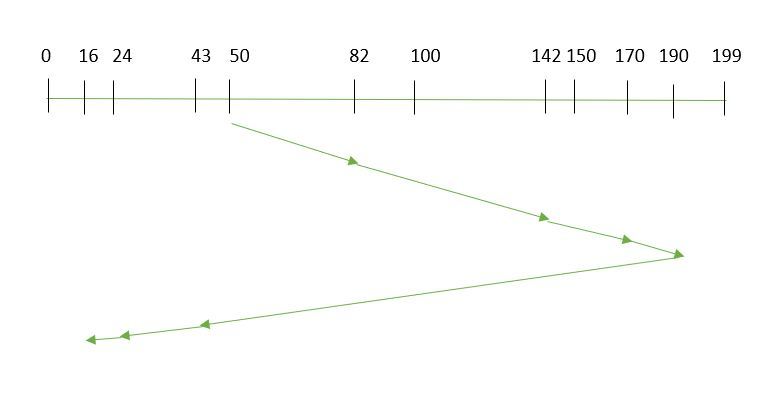
Advantages: 

* Provides more uniform wait time compared to SCAN

1. LOOK: It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”. 



So, the seek time is calculated as:

=(190-50)+(190-16)   
=314

**FILE MANAGEMENT**

A file management system is used for file maintenance (or management) operations. It is is a type of software that manages data files in a computer system. A file management system has limited capabilities and is designed to manage individual or group files, such as special office documents and records.

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*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.

1. **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. 

1. **Index sequential method –**

It is the other method of accessing a file that 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.

# **Structures of Directory in Operating System**

A **directory** is a container that is used to contain folders and files. It organizes files and folders in a hierarchical manner.

https://media.geeksforgeeks.org/wp-content/uploads/111-11.png

There are several logical structures of a directory, these are given below.

* **Single-level directory –**   
  The single-level directory is the simplest directory structure. In it, all files are contained in the same directory which makes it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have a unique name. if two users call their dataset test, then the unique name rule violated.

https://media.geeksforgeeks.org/wp-content/uploads/222-13.png

**Advantages:**

* Since it is a single directory, so its implementation is very easy.
* If the files are smaller in size, searching will become faster.
* The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

**Disadvantages:**

* There may chance of name collision because two files can have the same name.
* Searching will become time taking if the directory is large.
* This can not group the same type of files together.
* **Two-level directory –**   
  As we have seen, a single level directory often leads to confusion of files names among different users. the solution to this problem is to create a separate directory for each user.

In the two-level directory structure, each user has their own *user files directory (UFD)*. The UFDs have similar structures, but each lists only the files of a single user. system’s *master file directory (MFD)* is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.

https://media.geeksforgeeks.org/wp-content/uploads/222-2-1.png

**Advantages:**

* We can give full path like /User-name/directory-name/.
* Different users can have the same directory as well as the file name.
* Searching of files becomes easier due to pathname and user-grouping.

**Disadvantages:**

* A user is not allowed to share files with other users.
* Still, it not very scalable, two files of the same type cannot be grouped together in the same user.
* **Tree-structured directory –**   
  Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.   
  This generalization allows the user to create their own subdirectories and to organize their files accordingly.

https://media.geeksforgeeks.org/wp-content/uploads/222-3-1.png

A tree structure is the most common directory structure. The tree has a root directory, and every file in the system has a unique path.

**Advantages:**

* Very general, since full pathname can be given.
* Very scalable, the probability of name collision is less.
* Searching becomes very easy, we can use both absolute paths as well as relative.

**Disadvantages:**

* Every file does not fit into the hierarchical model, files may be saved into multiple directories.
* We cannot share files.
* It is inefficient, because accessing a file may go under multiple directories

# **System Protection in Operating System**

Protection refers to a mechanism which controls the access of programs, processes, or users to the resources defined by a computer system. We can take protection as a helper to multi programming operating system, so that many users might safely share a common logical name space such as directory or files.

**Need of Protection:**

* To prevent the access of unauthorized users and
* To ensure that each active programs or processes in the system uses resources only as the stated policy,
* To improve reliability by detecting latent errors.

**Role of Protection:**

The role of protection is to provide a mechanism that implement policies which defines the uses of resources in the computer system. Some policies are defined at the time of design of the system, some are designed by management of the system and some are defined by the users of the system to protect their own files and programs.

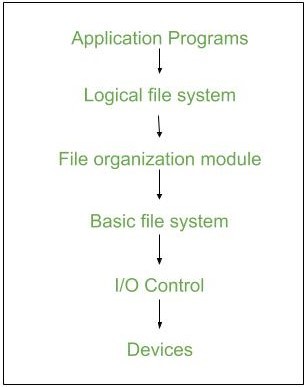
Every application has different policies for use of the resources and they may change over time so protection of the system is not only concern of the designer of the operating system. Application programmer should also design the protection mechanism to protect their system against misuse.

Policy is different from mechanism. Mechanisms determine how something will be done and policies determine what will be done. Policies are changed over time and place to place. Separation of mechanism and policy is important for the flexibility of the system.

# **File System Implementation in Operating System**

A file is a collection of related information. The file system resides on secondary storage and provides efficient and convenient access to the disk by allowing data to be stored, located, and retrieved.

**File system organized in many layers :**



* **I/O Control level –**

Device drivers acts as interface between devices and Os, they help to transfer data between disk and main memory. It takes block number a input and as output it gives low level hardware specific instruction./li>

* **Basic file system –**

It Issues general commands to device driver to read and write physical blocks on disk. It manages the memory buffers and caches. A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.

* **File organization Module –**

It has information about files, location of files and their logical and physical blocks. Physical blocks do not match with logical numbers of logical block numbered from 0 to N. It also has a free space which tracks unallocated blocks.

* **Logical file system –**

It manages metadata information about a file i.e includes all details about a file except the actual contents of file. It also maintains via file control blocks. File control block (FCB) has information about a file – owner, size, permissions, location of file contents.

**Advantages :**

1. Duplication of code is minimized.
2. Each file system can have its own logical file system.

**Disadvantages :**

If we access many files at same time then it results in low performance.

We can implement file system by using two types data structures.

# **File Allocation Methods**

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

* Contiguous Allocation
* Linked Allocation
* Indexed Allocation

The main idea behind these methods is to provide:

* Efficient disk space utilization.
* Fast access to the file blocks.

All the three methods have their own advantages and disadvantages as discussed below:

**1. Contiguous Allocation**

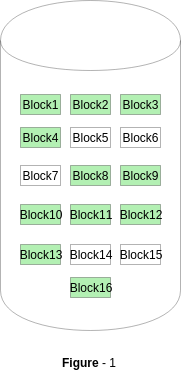
In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.  
The directory entry for a file with contiguous allocation contains

* Address of starting block
* Length of the allocated portion.

# **Free space management in Operating System**

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. Bitmap or Bit vector –A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: 0 indicates that the block is allocated and 1 indicates a free block.The given instance of disk blocks on the disk in Figure 1 (where green blocks are allocated) can be represented by a bitmap of 16 bits as:**0000111000000110**.



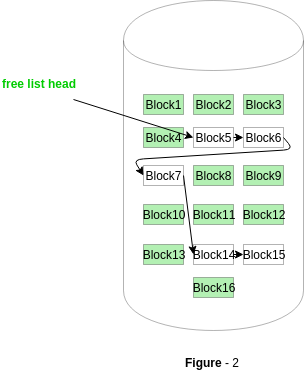
**Advantages –**

* + Simple to understand.
  + Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

The block number can be calculated as:  
*(number of bits per word) \*(number of 0-values words) + offset of bit first bit 1 in the non-zero word*.

For the Figure-1, we scan the bitmap sequentially for the first non-zero word.  
The first group of 8 bits (00001110) constitute a non-zero word since all bits are not 0. After the non-0 word is found, we look for the first 1 bit. This is the 5th bit of the non-zero word. So, offset = 5.  
Therefore, the first free block number = 8\*0+5 = 5.

1. **Linked List –**  
   In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.



In Figure-2, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list.  
A drawback of this method is the I/O required for free space list traversal.

1. **Grouping –**  
   This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks.  
   An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.
2. **Counting –**  
   This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block.  
   Every entry in the list would contain:
   * Address of first free disk block
   * A number n