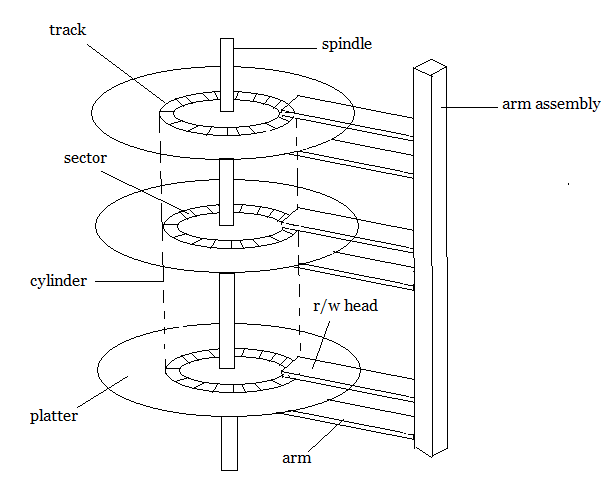
**Secondary Storage Structure**

Secondary storage devices are those devices whose memory is non volatile, meaning; the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

* Secondary storage is also called auxiliary storage.
* Secondary storage is less expensive when compared to primary memory like RAMs.
* The speed of the secondary storage is also lesser than that of primary storage.
* Hence, the data which is less frequently accessed is kept in the secondary storage.
* A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.

**Magnetic Disk Structure**

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.



**Structure of a magnetic disk**

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

* **Transfer rate:** This is the rate at which the data moves from disk to the computer.
* **Random access time:** It is the sum of the seek time and rotational latency.

**Seek time** is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

**Disk Scheduling Algorithms**

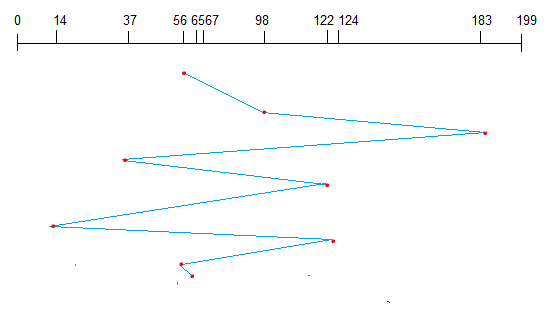
On a typical multiprogramming system, there will usually be multiple disk access requests at any point of time. So those requests must be scheduled to achieve good efficiency. Disk scheduling is similar to process scheduling. Some of the disks scheduling algorithms are described below.

**First Come First Serve:**

This algorithm performs requests in the same order asked by the system. Let's take an example where the queue has the following requests with cylinder numbers as follows:

**98, 183, 37, 122, 14, 124, 65, 67**

Assume the head is initially at cylinder **56**. The head moves in the given order in the queue i.e., **56→98→183→...→67**.

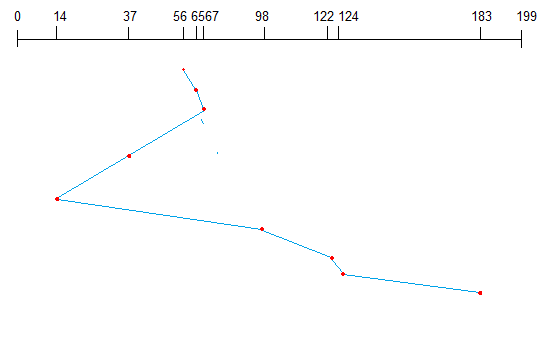


**Shortest Seek Time First (SSTF):**

Here the position which is closest to the current head position is chosen first. Consider the previous example where disk queue looks like,

**98, 183, 37, 122, 14, 124, 65, 67**

Assume the head is initially at cylinder **56**. The next closest cylinder to **56** is **65**, and then the next nearest one is **67**, then **37**, **14**, so on.

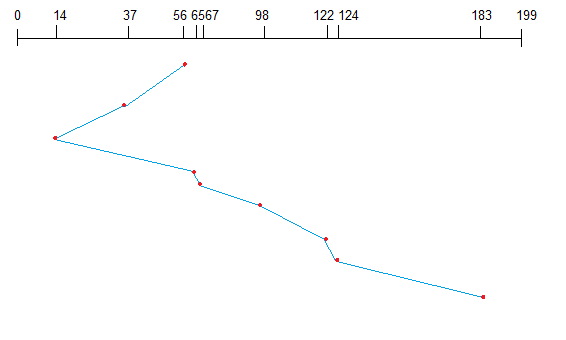


**SCAN algorithm:**

This algorithm is also called the elevator algorithm because of it's behavior. Here, first the head moves in a direction (say backward) and covers all the requests in the path. Then it moves in the opposite direction and covers the remaining requests in the path. This behavior is similar to that of an elevator. Let's take the previous example,

**98, 183, 37, 122, 14, 124, 65, 67**

Assume the head is initially at cylinder **56**. The head moves in backward direction and accesses **37**and **14**. Then it goes in the opposite direction and accesses the cylinders as they come in the path.



# RAID

RAID or “Redundant Arrays of Inexpensive Disks” is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both. The term was coined by David Patterson, Garth A. Gibson, and Randy Katz at the University of California, Berkeley in 1987.

**Why data redundancy?**

Data redundancy, although taking up extra space, adds to disk reliability. This means, in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation. On the other hand, if the data is spread across just multiple disks without the RAID technique, the loss of a single disk can affect the entire data.

**Key evaluation points for a RAID System**

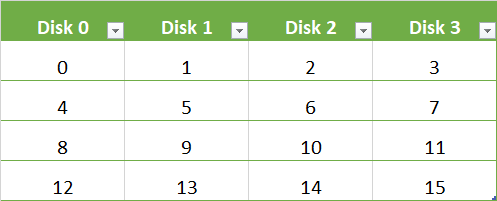
* **Reliability:**How many disk faults can the system tolerate?
* **Availability:** What fraction of the total session time is a system in uptime mode, i.e. how available is the system for actual use?
* **Performance:**How good is the response time? How high is the throughput (rate of processing work)? Note that performance contains a lot of parameters and not just the two.
* **Capacity:** Given a set of N disks each with B blocks, how much useful capacity is available to the user?

RAID is very transparent to the underlying system. This means, to the host system, it appears as a single big disk presenting itself as a linear array of blocks. This allows older technologies to be replaced by RAID without making too many changes in the existing code.

**Different RAID levels**

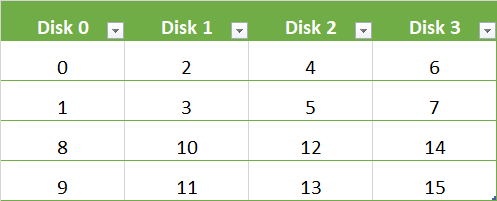
**RAID-0 (Striping)**

* Blocks are “striped” across disks.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/raid01.png)

In the figure, blocks “0,1,2,3” form a stripe.

* Instead of placing just one block into a disk at a time, we can work with two (or more) blocks placed into a disk before moving on to the next one.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/raid0_chunk.png)

**Evaluation:**

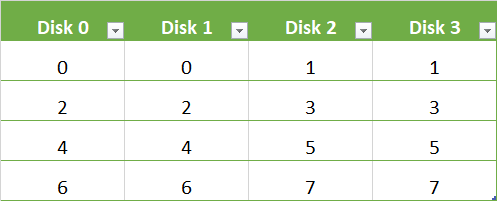
* Reliability: 0

There is no duplication of data. Hence, a block once lost cannot be recovered.

* Capacity:N\*B  
  The entire space is being used to store data. Since there is no duplication, N disks each having B blocks are fully utilized.

**RAID-1 (Mirroring)**

* More than one copy of each block is stored in a separate disk. Thus, every block has two (or more) copies, lying on different disks.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/raid1.png)

The above figure shows a RAID-1 system with mirroring level 2.

* RAID 0 was unable to tolerate any disk failure. But RAID 1 is capable of reliability.

**Evaluation:**

Assume a RAID system with mirroring level 2.

* Reliability: 1 to N/2

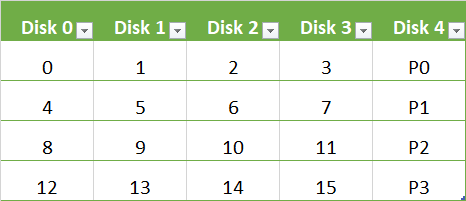
1 disk failure can be handled for certain, because blocks of that disk would have duplicates on some other disk. If we are lucky enough and disks 0 and 2 fail, then again this can be handled as the blocks of these disks have duplicates on disks 1 and 3. So, in the best case, N/2 disk failures can be handled.

* Capacity: N\*B/2

Only half the space is being used to store data. The other half is just a mirror to the already stored data.

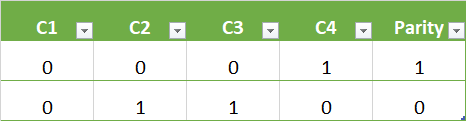
**RAID-4 (Block-Level Striping with Dedicated Parity)**

* Instead of duplicating data, this adopts a parity-based approach.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/raid4.png)

In the figure, we can observe one column (disk) dedicated to parity.

* Parity is calculated using a simple XOR function. If the data bits are 0,0,0,1 the parity bit is XOR(0,0,0,1) = 1. If the data bits are 0,1,1,0 the parity bit is XOR(0,1,1,0) = 0. A simple approach is that even number of ones results in parity 0, and an odd number of ones results in parity 1.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/parityy.png)

Assume that in the above figure, C3 is lost due to some disk failure. Then, we can recompute the data bit stored in C3 by looking at the values of all the other columns and the parity bit. This allows us to recover lost data.

**Evaluation:**

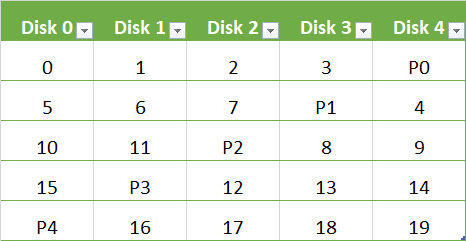
* Reliability: 1

RAID-4 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data.

* Capacity: (N-1)\*B
* One disk in the system is reserved for storing the parity. Hence, (N-1) disks are made available for data storage, each disk having B blocks.

**RAID-5 (Block-Level Striping with Distributed Parity)**

* This is a slight modification of the RAID-4 system where the only difference is that the parity rotates among the drives.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/raid5.png)

In the figure, we can notice how the parity bit “rotates”.

* This was introduced to make the random write performance better.

**Evaluation:**

* **Reliability: 1**

**RAID-5 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data. This is identical to RAID-4.**

* **Capacity: (N-1)\*B**

**Overall, space equivalent to one disk is utilized in storing the parity. Hence, (N-1) disks are made available for data storage, each disk having B blocks.**

**What about the other RAID levels?**

**RAID-2 consists of bit-level striping using a Hamming Code parity. RAID-3 consists of byte-level striping with a dedicated parity. These two are less commonly used.**

**RAID-6 is a recent advancement which contains a distributed double parity, which involves block-level striping with 2 parity bits instead of just 1 distributed across all the disks. There are also hybrid RAIDs, which make use of more than one RAID levels nested one after the other, to fulfill specific requirements.**

**File System**

A file can be "free formed", indexed or structured collection of related bytes having meaning only to the one who created it. Or in other words an entry in a directory is the file. The file may have attributes like name, creator, date, type, permissions etc.

**File Structure**

A file has various kinds of structure. Some of them can be:

* **Simple Record Structure** with lines of fixed or variable lengths.
* **Complex Structures** like formatted document or reloadable load files.
* **No Definite Structure** like sequence of words and bytes etc.

**Attributes of a File**

Following are some of the attributes of a file :

* **Name.** It is the only information which is in human-readable form.
* **Identifier**. The file is identified by a unique tag (number) within file system.
* **Type**. It is needed for systems that support different types of files.
* **Location**. Pointer to file location on device.
* **Size**. The current size of the file.
* **Protection**. This controls and assigns the power of reading, writing, executing.
* **Time, date, and user identification**. This is the data for protection, security, and usage monitoring.

**File Access Methods**

The way that files are accessed and read into memory is determined by Access methods. Usually a single access method is supported by systems while there are OS's that support multiple access methods.

**Sequential Access**

* Data is accessed one record right after another is an order.
* Read command cause a pointer to be moved ahead by one.
* Write command allocate space for the record and move the pointer to the new End Of File.
* Such a method is reasonable for tape.

**Direct Access**

* This method is useful for disks.
* The file is viewed as a numbered sequence of blocks or records.
* There are no restrictions on which blocks are read/written; it can be done in any order.
* User now says "read n" rather than "read next".
* "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

**Indexed Sequential Access**

* It is built on top of Sequential access.
* It uses an Index to control the pointer while accessing files.

**What is a Directory?**

Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

Following is the information maintained in a directory :

* **Name** : The name visible to user.
* **Type** : Type of the directory.
* **Location** : Device and location on the device where the file header is located.
* **Size** : Number of bytes/words/blocks in the file.
* **Position** : Current next-read/next-write pointers.
* **Protection** : Access control on read/write/execute/delete.
* **Usage** : Time of creation, access, modification etc.
* **Mounting** : When the root of one file system is "grafted" into the existing tree of another file system its called Mounting.

### Directory Implementation in Operating System

Directory efficiency, performance and reliability are totally dependent on the technique used to store them. They are saved on Disks with two well Known Techniques:

**1) Linear List:** It is the Simple method of directory implementation. It makes use of a file name and a pointer to data blocks. Whenever a new file is created, whole directory is checked to make sure that no existing file has the same name. If the directory contains no file with this name it is added to the end of directory. For the deletion operation file is searched and the allocated space is released.  
  
The Main Disadvantage is of this technique is that every time we create a new file, directory has to be checked for existing file name .The second problem is that files are accessed in a serial fashion which results in slow performance  
  
**2) Hash Tables:** In technique Directories are saved in same linear fashion but a hash table is used. It takes a value that is returned from file name and returns a pointer to file name. This technique decreases the search time for files.  
  
Disadvantage of using this technique is fixed size of Hash Tables. E.g. if we have hash table which can store 32 entries and we want to add 33th entry to it, then it is required to be extended. This new extended Hash Table will have 64 entry capacity( 2 X32).As the size of Hash Table increases we also need a Hash Table function that can reference range of 0 to 64 entries.