File management

Files

Concepts:

- A ***file*** is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.
- In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

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.

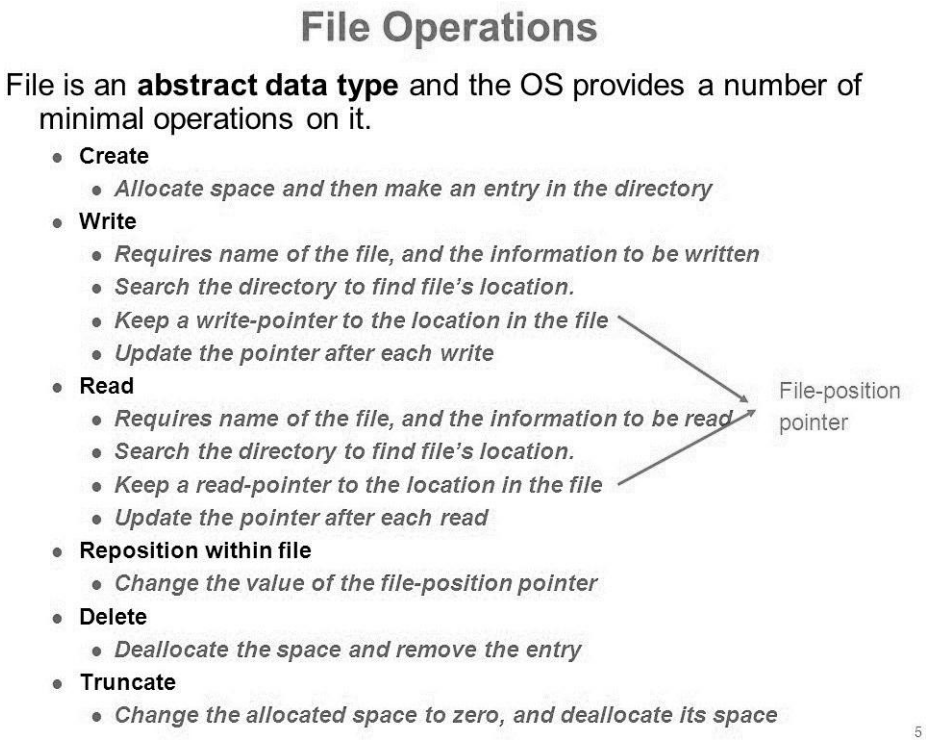
File Operations

The operating system must do to perform basic file operations given below.

- **Creating a file:** Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file:** To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- **Reading a file:** To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated.
- **Repositioning within a file**: The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seek.
- **Deleting a file.** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused bv other files, and erase the directory entry.

- **Protection:** Access-control information determines who can do reading, writing, executing, and so on.
- **Truncating a file:** The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the tile be reset to length zero and its file space released.

In brief



File Types

File Type	Usual extension	Function
Executable	exe, com, bin or none	ready-to-run machine-language program
Object	obj, o	complied, machine language, not linked
Source code	c, p, pas, 177, asm, a	source code in various languages
Batch	bat, sh	commands to the command interpreter
Text	txt, doc	textual data documents
Word processor	wp, tex, rrf, etc.	various word-processor formats
Library	lib, a	libraries of routines
Print or view	ps, dvi, gif, pdf	ASCII or binary file
Archive	arc, zip, tar, gz	related files grouped into one file, sometimes compressed.

File System Structure

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

Files can be structured in several ways in which three common structures are given in this tutorial with their short description one by one.

File Structure 1

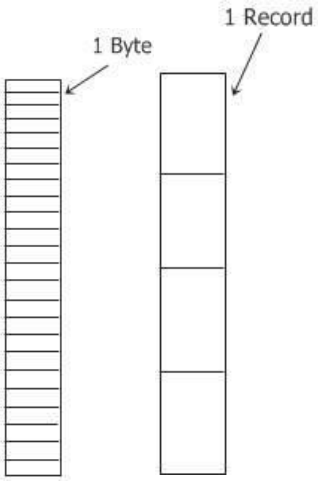
- Here, as you can see from the figure 1, the file is an unstructured sequence of bytes.
- Therefore, the OS doesn't care about what is in the file, as all it sees are bytes.

File Structure 2

- Now, as you can see from the figure 2 that shows the second structure of a file, where a file is a sequence of fixed-length records where each with some internal structure.
- Central to the idea about a file being a sequence of records is the idea that read operation returns a record and write operation just appends a record.

File Structure 3

- Now in the last structure of a file that you can see in the figure 3, a file basically consists of a tree of records, not necessarily all the same length, each containing a key field in a fixed position in the record. The tree is stored on the field, just to allow the rapid searching for a specific key.



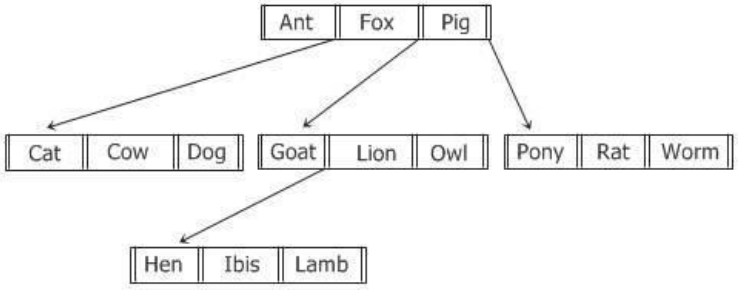


Fig.1 **Fig.2** **Fig.3**

File Access method

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

- Sequential access
- Direct/Random access
- Indexed sequential access

1. Sequential Access

- A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one.
- The idea of Sequential access is based on the tape model which is a sequential access device.
- The Sequential access method is best because most of the records in a file are to be processed. For example, transaction files.
- **Example:** Compilers usually access files in this fashion.

In Brief:

- 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.

Advantages of sequential access

- It is simple to program and easy to design.
- Sequential file is best use if storage space.

Disadvantages of sequential access

- Sequential file is time consuming process.
- It has high data redundancy.
- Random searching is not possible.

2. Direct Access

- Sometimes it is not necessary to process every record in a file.
- It is not necessary to process all the records in the order in which they are present in the memory. In all such cases, direct access is used.
- The disk is a direct access device which gives us the reliability to random access of any file block.
- In the file, there is a collection of physical blocks and the records of that blocks.
- **Example:** Databases are often of this type since they allow query processing that involves immediate access to large amounts of information. All reservation systems fall into this category.

In brief:

- 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 dobe 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.

Advantages:

- Direct access file helps in online transaction processing system (OLTP) like online railway reservation system.
- In direct access file, sorting of the records are not required.
- It accesses the desired records immediately.
- It updates several files quickly.
- It has better control over record allocation.

Disadvantages:

- Direct access file does not provide backup facility.
- It is expensive.
- It has less storage space as compared to sequential file.

3. Indexed Sequential Access

- The index sequential access method is a modification of the direct access method.
- Basically, it is kind of combination of both the sequential access as well as direct access.
- The main idea of this method is to first access the file directly and then it accesses sequentially.
- In this access method, it is necessary for maintaining an index.
- The index is nothing but a pointer to a block.
- The direct access of the index is made to access a record in a file.
- The information which is obtained from this access is used to access the file. Sometimes the indexes are very big.
- So to maintain all these hierarchies of indexes are built in which one direct access of an index leads to information of another index access.
- It is built on top of Sequential access.
- It uses an Index to control the pointer while accessing files.

Advantages:

- In indexed sequential access file, sequential file and random file access is possible.
- It accesses the records very fast if the index table is properly organized.
- The records can be inserted in the middle of the file.
- It provides quick access for sequential and direct processing.
- It reduces the degree of the sequential search.

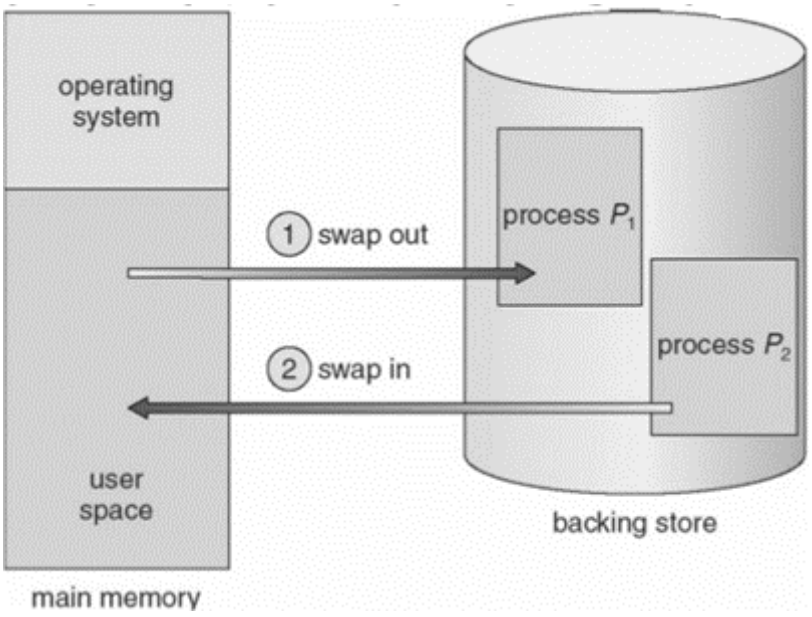
Disadvantages:

- Indexed sequential access file requires unique keys and periodic reorganization.
- Indexed sequential access file takes longer time to search the index for the data access or retrieval.

- It requires more storage space.
- It is expensive because it requires special software.
- It is less efficient in the use of storage space as compared to other file organizations.

Swapping:

- ***Swapping*** is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes.
- At some later time, the system swaps back the process from the secondary storage to main memory.
- Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason
- **Swapping is also known as a technique for memory compaction**.
- Swap space is a space on hard disk which is a substitute of physical memory.
- It is used as virtual memory which contains process memory image.
- Whenever our computer run short of physical memory it uses its virtual memory and stores information in memory on disk.



File Space Allocation:

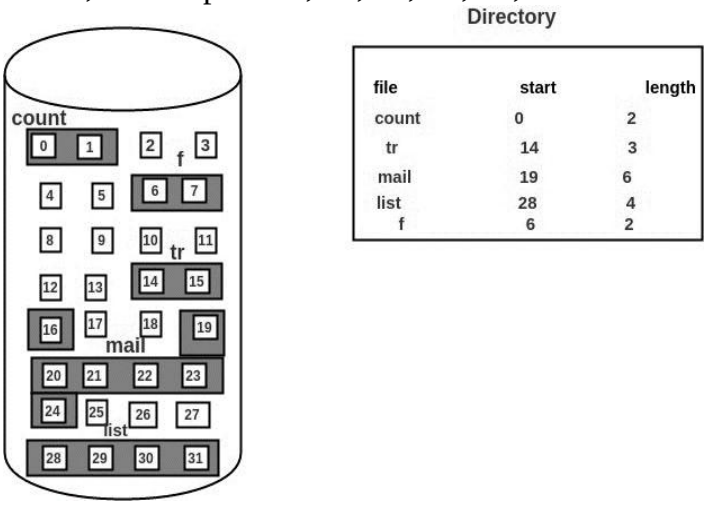
Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

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
    1. Address of starting block
    2. Length of the allocated portion.
- The *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.



- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

Advantages:

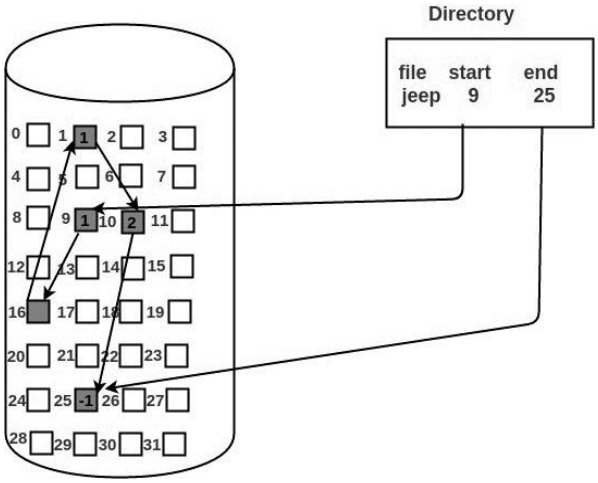
- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked Allocation

- In this scheme, each file is a linked list of disk blocks which **need not be** contiguous.
- The disk blocks can be scattered anywhere on the disk.
- The directory entry contains a pointer to the starting and the ending file block.
- Each block contains a pointer to the next block occupied by the file.
- *The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.*



- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Advantages:

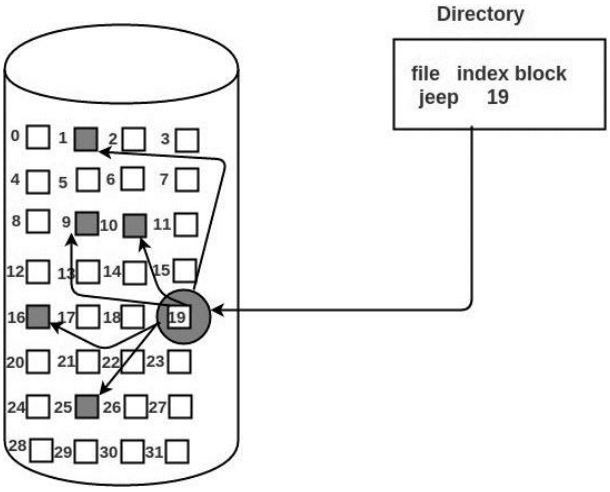
1. File size does not have to be specified.
2. No external fragmentation.

Disadvantages:

1. It does sequential access efficiently and is not for direct access
2. Each block contains a pointer, wasting space
3. Blocks scatter everywhere and a large number of disk seeks may be necessary
4. Reliability: what if a pointer is lost or damaged?

3. Indexed Allocation

- In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block.
- The ith entry in the index block contains the disk address of the ith file block.
- The directory entry contains the address of the index block as shown in the image:



- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.

- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

Advantages:

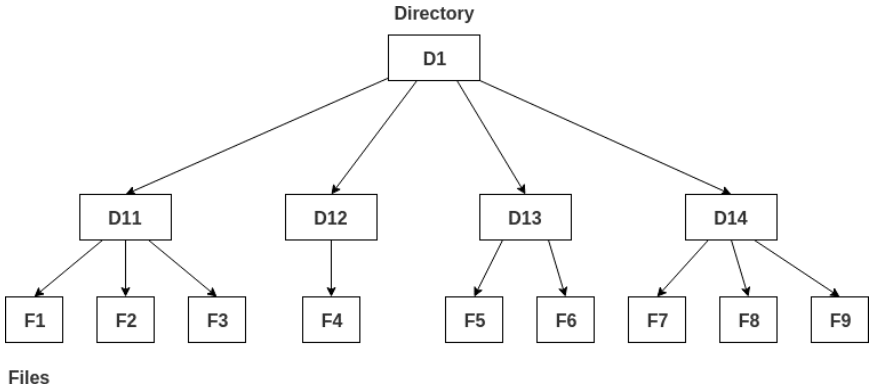
- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

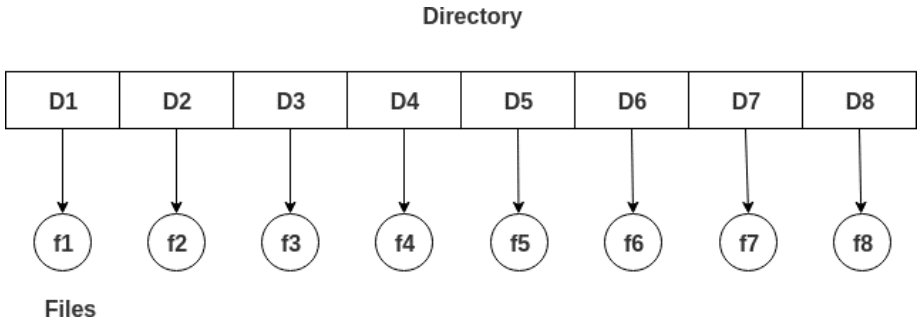
Directory Structure

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



1. Single-level directory –

- Single level directory is simplest directory structure.
- In it all files are contained in same directory which make 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 the unique name. if two users call their dataset test, then the unique name rule violated.



Advantages:

- Since it is a single directory, so its implementation is very easy.
- If files are smaller in size, searching will faster.

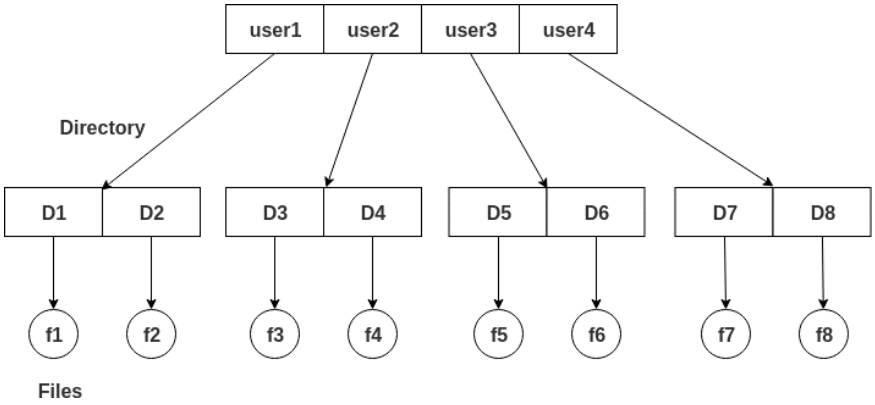
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

- There may chance of name collision because two files cannot have the same name.
- Searching will become time taking if directory will large.
- In this cannot group the same type of files together.

2. Two-level directory –

- As, a single level directory often leads to confusion of files names among different users hence 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 has 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.



Advantages:

- We can give full path like /User-name/directory-name/.
- Different users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

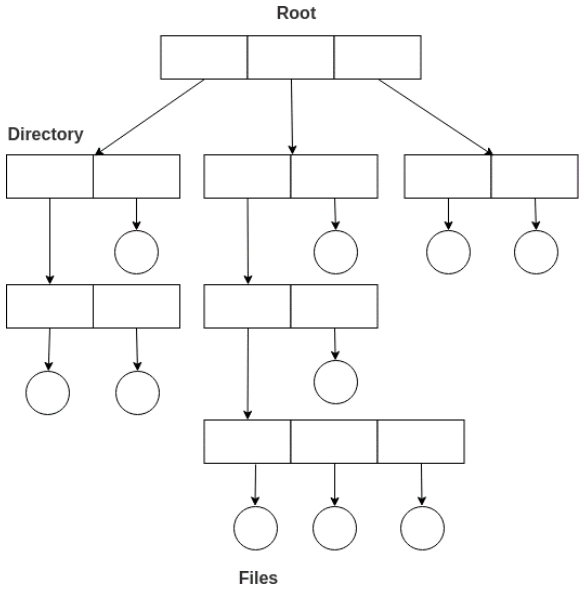
Disadvantages:

- 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.

3. 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 on their files accordingly.
- A tree structure is the most common directory structure. The tree has a root directory, and every file in the system have a unique path.



Advantages:

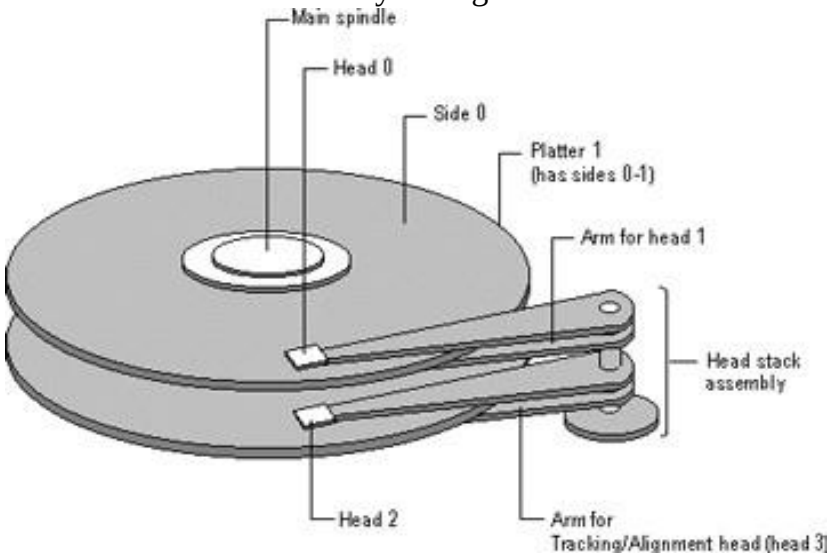
- Very generalize, since full path name can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute path as well as relative.

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.

Disk Organization:

A physical structure of disk is a memory storage device which looks like this:

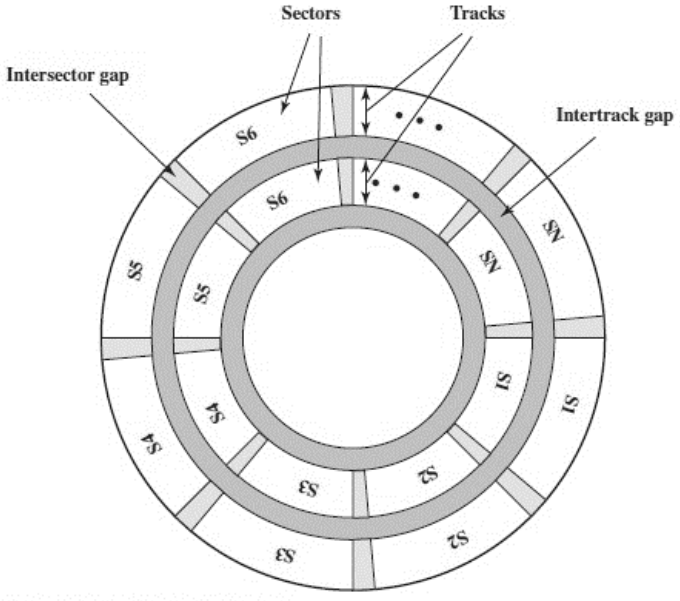


- Hard disk consists of a number of **platters**
- The platters **rotate** at a very high speed (5400 RPM to 10,000 RPM)
- Disk (read/write) heads move over the platter surface to read and write (magnetize) data bits

- The disk head can read or write data only when the desired **disk surface area** is under the disk head.
- Read-Write(R-W) head moves over the rotating hard disk.
- It is this Read-Write head that performs all the read and write operations on the disk and hence, position of the R-W head is a major concern.
- To perform a read or write operation on a memory location, we need to place the R-W head over that position. Some important terms must be noted here:
    1. **Seek time** – The time taken by the R-W head to reach the desired track from it's current position.
    2. **Rotational latency** – Time taken by the sector to come under the R-W head.
    3. **Data transfer time** – Time taken to transfer the required amount of data. It depends upon the rotational speed.
    4. **Controller time** – The processing time taken by the controller.
    5. **Average Access time** – seek time + Average Rotational latency + data transfer time + controller time.

Logical Disk Structure:

- The disk is divided into ***tracks***.
- Each track is further divided into ***sectors***.
- The point to be noted here is that outer tracks are bigger in size than the inner tracks but they contain the same number of sectors and have equal storage capacity.
- This is because the storage density is high in sectors of the inner tracks whereas the bits are sparsely arranged in sectors of the outer tracks.
- Some space of every sector is used for formatting.
    - So, the actual capacity of a sector is less than the given capacity.



Basically, hard disk can be divided in the logical structure in the following five logical terms:

- MBR (Master Boot Record)
- DBR (DOS Boot Record)
- FAT (File Allocation Tables)
- Root Directory
- Data Area

1. The Master Boot Record (or MBR)

- At the beginning of the hard drive is the MBR. When your computer starts using your hard drive, this is where it looks first.
- The MBR itself has a specific organization. The size of the MBR is 512 bytes.
- The **boot loader** is the first 446 bytes of the MBR. This section contains executable code, where programs are housed.
- The **partition tables** are 4 slots of 16 bytes each, containing the description of a partition (primary or extended) on the disk.

Here is how to describe a partition:

- State of the partition (inactive partition bootable) - (1 byte)
- Custom heads at the beginning of the partition - (1 byte)
- Cylinder sector and the beginning of the partition - (2 bytes)
- Type of partition (file system, eg, 32 fat, ext2 etc ...) - (1 bytes)
- Head of the end of the partition (1 byte)
- Cylinder sector and the end of the score - (2 bytes)
- Number of sectors between the MBR and the first sector of the partition - (4 bytes)
- Number of sector of the partition - (4 bytes)
- The **Magic Number** is two bytes used to determine if the hard disk has a bootloader or not. If it does, the magic number should be equal in value to hexadecimal 55AA.

2. DOS Boot Record (DBR) / DOS Boot Sector

- After the partition table, the DOS Boot Record (DBR) or sometimes called DOS Boot Sector is the second most important information on your hard drive.
- The DOS Boot Record (DBR) for the first partition on a hard disk is usually found at Absolute Sector 63 (the 64th sector on the disk drive) or in CHS form we can say C–H–S = 0–1–1 for most drives.
- The sector on which DBR resides becomes logical sector 1 of that particular partition for the DOS. The sector number used by DOS starts from the physical sector on which DBR is located.
- First logical sector of each DOS partition will contain a DOS Boot Record (DBR) or DOS Boot Sector. The job of the DBR is to load the operating system from

the hard disk drive into the main memory of computer and give the systems control to the loaded program.

3. FAT (File Allocation Tables)

- Following DBR are the File Allocation Tables.
- The FAT has been modified several times to accommodate expanding needs.
- FAT keeps a map of the complete surface of the disk drive such that, which area is free, which area is bad, which area is taken up by which file etc.
- When some data stored on the disk surface is to be accessed, the DOS consults the FAT to find out the areas of the hard disk surface that contains the data.
- The FAT does not keep track of each and every sector on the disk surface instead it manages the disk area in a group of sectors called "cluster" or "allocation unit" (See the Cluster Discussed before, in the same chapter).
- A cluster is the smallest unit of hard disk drive space that DOS allocates to a file, it consists of one or more sectors depending on the drive size.
- The cluster size is decided and fixed by the DOS FORMAT program during the high level formatting of the hard disk drive. (See the "size of clusters" discussion, given before)
- Actually, the FAT is an index of the clusters of the entire volume.
- The FAT has one entry for each cluster. The first two entries in a FAT contain information about the FAT.
- The third and subsequent entries in the FAT are assigned to clusters of disk space, starting with the first cluster available for use by files

4. Root Directory or Directory table

- Following the last FAT is the Root Directory.
- The root directory is like a table of contents for the information stored on the hard disk drive.
- The location of the Root Directory can easily be established by adding up the values from boot record, as it is positioned following the FATs.
- The directory area keeps the information about the file name, date and time of the file creation, file attribute, file size and starting cluster of the particular file.
- Each directory entry describing this information about a file is a 32 byte information.
- The root directory contains information about the files and directories branching from the root directory.
- All further directories are themselves stored as files, in the same format as the root directory.

- Previously the root directory used to be fixed in size and located at a fixed position on disk but now it is free to grow as necessary as it is now treated as a file.

5. **Data Area (or Files Area)**

- Following the Root Directory, the Data Area (or Files Area) starts. Rather we can say that the remainder of the volume after Root Directory is the Data Area.
- The data area contains the actual data stored on the disk surface.
- DOS uses cluster number 2 for the first sector of the data area therefore we should keep it in mind while performing various calculations that the cluster number should start from 2.

Raid Structure of Disk:

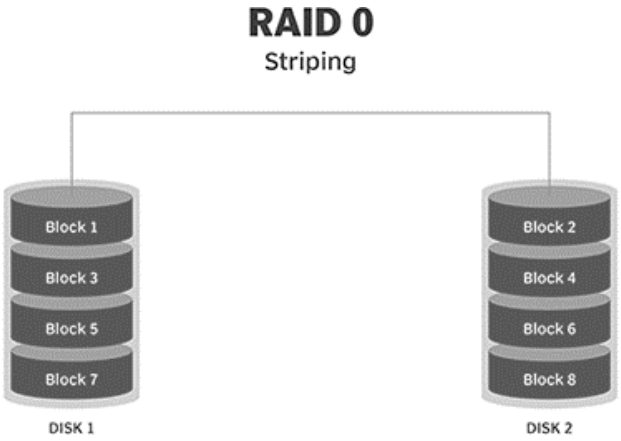
- RAID, or "Redundant Arrays of Independent Disks" is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both.

Why data redundancy?

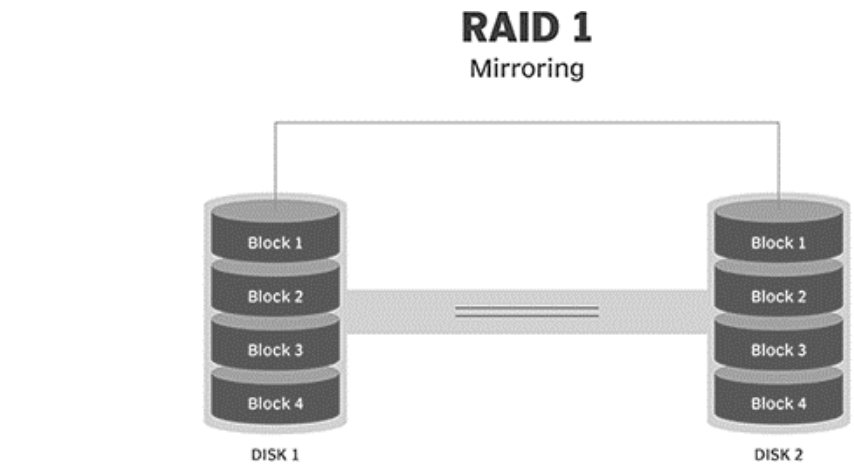
- Data redundancy, although taking up extra space, adds to disk reliability.
- This means, in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation.
- On the other hand, if the data is spread across just multiple disks without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

Standard RAID levels

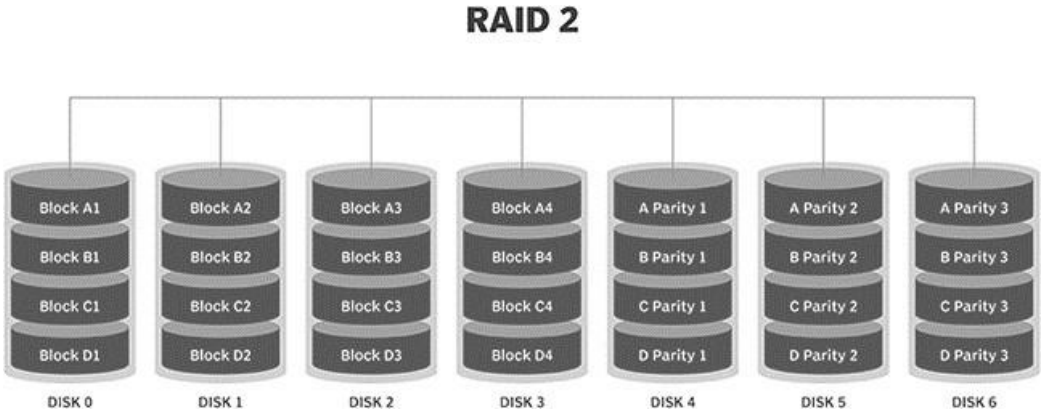
RAID 0: This configuration has striping, but no redundancy of data. It offers the best performance, but no fault tolerance.



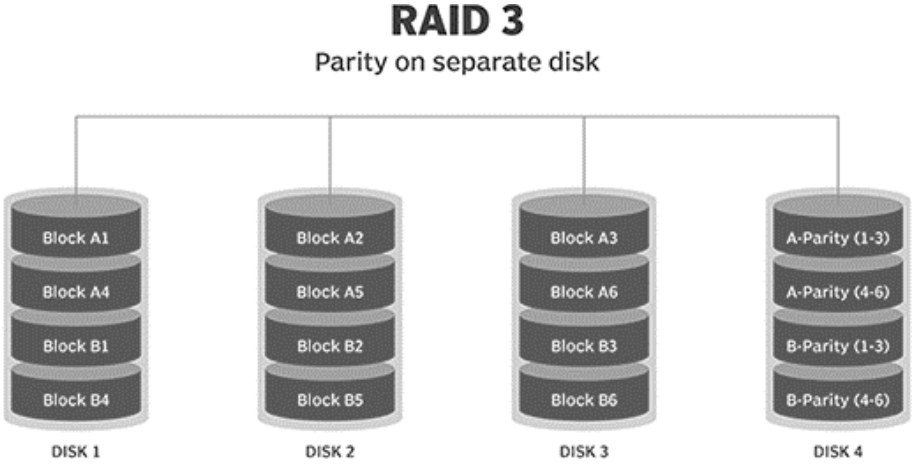
RAID 1: Also known as disk mirroring, this configuration consists of at least two drives that duplicate the storage of data. There is no striping. Read performance is improved since either disk can be read at the same time. Write performance is the same as for single disk storage**.**



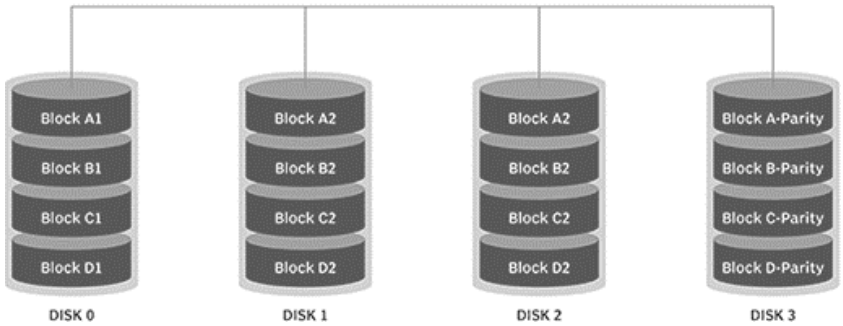
RAID 2: This configuration uses striping across disks, with some disks storing error checking and correcting (ECC) information. It has no advantage over RAID 3 and is no longer used.



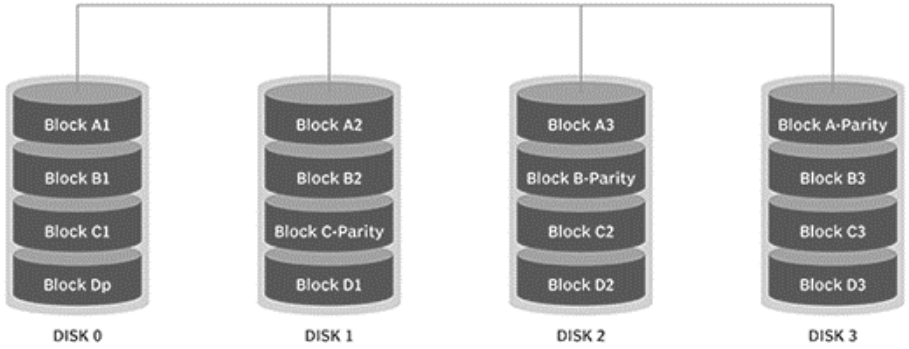
RAID 3: This technique uses striping and dedicates one drive to storing parity information. The embedded ECC information is used to detect errors. Data recovery is accomplished by calculating the exclusive OR (XOR) of the information recorded on the other drives. Since an I/O operation addresses all the drives at the same time, RAID 3 cannot overlap I/O. For this reason, RAID 3 is best for single-user systems with long record applications.



RAID 4: This level uses large stripes, which means you can read records from any single drive. This allows you to use overlapped I/O for read operations. Since all write operations have to update the parity drive, no I/O overlapping is possible. RAID 4 offers no advantage over RAID 5.



RAID 5: This level is based on block-level striping with parity. The parity information is striped across each drive, allowing the array to function even if one drive were to fail. The array's architecture allows read and write operations to span multiple drives. This results in performance that is usually better than that of a single drive, but not as high as that of a RAID 0 array. RAID 5 requires at least three disks, but it is often recommended to use at least five disks for performance reasons.



RAID 6: This technique is similar to RAID 5, but includes a second parity scheme that is distributed across the drives in the array. The use of additional parity allows the array to continue to function even if two disks fail simultaneously. However, this extra protection comes at a cost. RAID 6 arrays have a higher cost per gigabyte (GB) and often have slower write performance than RAID 5 arrays.