File System Interface

The file system provides the mechanism for on-line storage of and access to both data and programs of the operating system and all the users of the computer system. The file system consists of two distinct parts:

- **A collection of files:** Each storing related data,
- **A directory structure:** This organizes and provides information about all the files in the system.

File Concept

A file is a collection of related information that is recorded on secondary storage. From a user's perspective, a file is the smallest allotment of logical secondary storage and data cannot be written to secondary storage unless they are within a file.
The following four terms are in common use when discussing files:

1. **Filed:** A field is the basic element of data. An individual field contains a single value, such as an employee's last name, a date, or the value of a sensor reading. It is characterized by its length and data type.
2. **Record:** A record is a collection of related fields that can be treated as a unit by some application program. For example, an employee record would contain such fields as name, social security number, job classification, date of hire, and so on.
3. **File:** A file is a collection of similar records. The file is treated as a single entity by users and applications and may be referenced by name.
4. **Database:** A database is a collection of related data. A database may contain all of the information related to an organization or project, such as a business or a scientific study. The database itself consists of one or more types of files.

File Attributes

A file has the following attributes:

1. **Name:** The symbolic file name is the only information kept in human readable form.
2. **Identifier:** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
3. **Type:** This information is needed for those systems that support different types.
4. **Location:** This information is a pointer to a device and to the location of the file on that device.
5. **Size:** The current size of the file (in bytes, words, or blocks), and possibly the maximum allowed size are included in this attribute.
6. **Protection:** Access-control information determines who can do reading, writing, executing, and so on.
7. **Time, date, and user identification:** This information may be kept for creation, modification and last use. These data can be useful for protection, security, and usage monitoring.

Manish Gupta

File Operations

The operating system can provide system calls to create, write, read, reposition, delete, and truncate files. The common file operations are described as follows:

1. **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. The directory entry records the name of the file and the location in the file system, and possibly other information.
2. **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 location of the file. 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.
3. **Reading a file:** To read from a file, we use a system call that specifies the name of the file and where (in main memory) the next block of the file should be put. Again, the directory is searched for the associated directory 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.
4. **Repositioning within a file:** The directory is searched for the appropriate entry, and the current-file-position is set to a given value. Repositioning within a file does not need to involve any actual I/O. This file operation is also known as a file seeks.
5. **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 by other files, and erase the directory entry.
6. **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 file be reset to length zero and its file space released

File Types

The files are classified into different categories as follows:

File Type	Usual Extension	Function
Executable	• exe • com • bin • none	Read to run machine language program
Object	• obj • o	• Compiled • Machine language
Source Code	• c • cpp • java • py	Source code in various languages

Manish Gupta

Batch	• bat • sh	Commands to the command interpreter
Text	• txt • doc	• Textual data • Documents
Word Processor	• wp • tex • rrf • doc	• Various word-processor formats
Library	• lib • a • so • dll • mpeg • mov • rm	• Libraries of routines for programmers
Archive	• arc • zip • tar	• Related files grouped into one file • Sometimes compresses for archiving or storage

File Access Mechanisms

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

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 advances 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.

2. Direct/Random 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.

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

3. Indexed sequential access

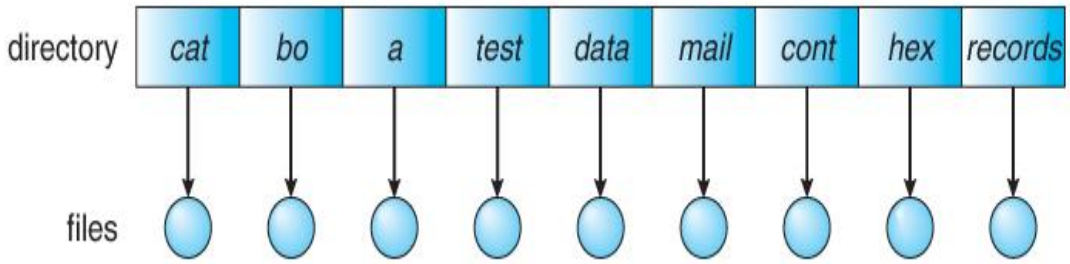
It is the other method of accessing a file which is built on the top of the direct access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index and then by the help of pointer we access the file directly.

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

Directory Structure

A directory is an object that contains the names of file system objects. File system allows the users to organize files and other file system objects through the use of directories. The structure created by placement of names in directories can take a number of forms:

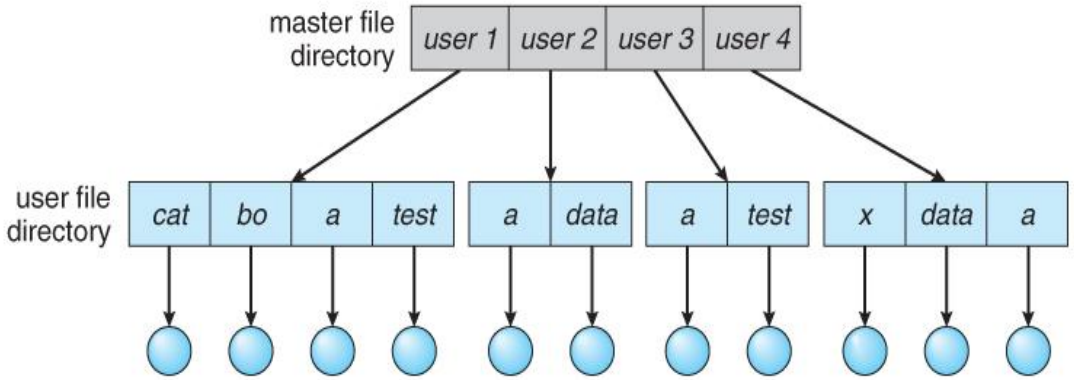
1. **Single-Level Directory:** The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand. A single-level directory has significant limitations, when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names.



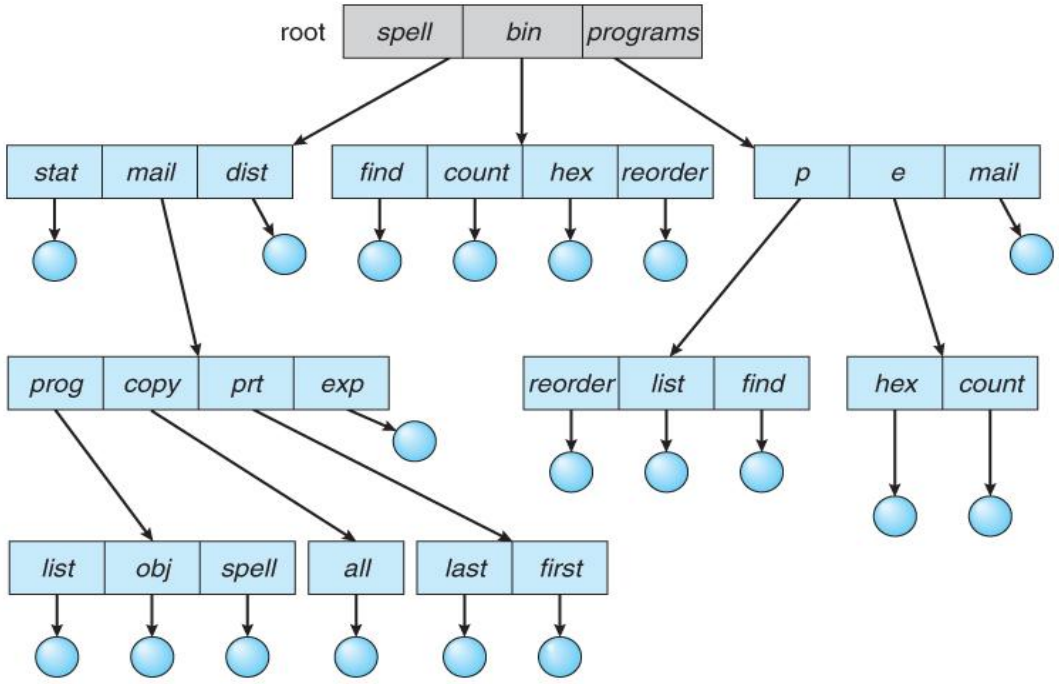
2. **Two-Level Directory:** In the two-level directory structure, each user has its own user file directory (UFD). Each UFD has a similar structure, but lists only the files of a single user. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.

   When a user refers to a particular file, only his own UFD is searched. Different users may have files with the same name, as long as all the file names within each UFD are unique.

   To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.



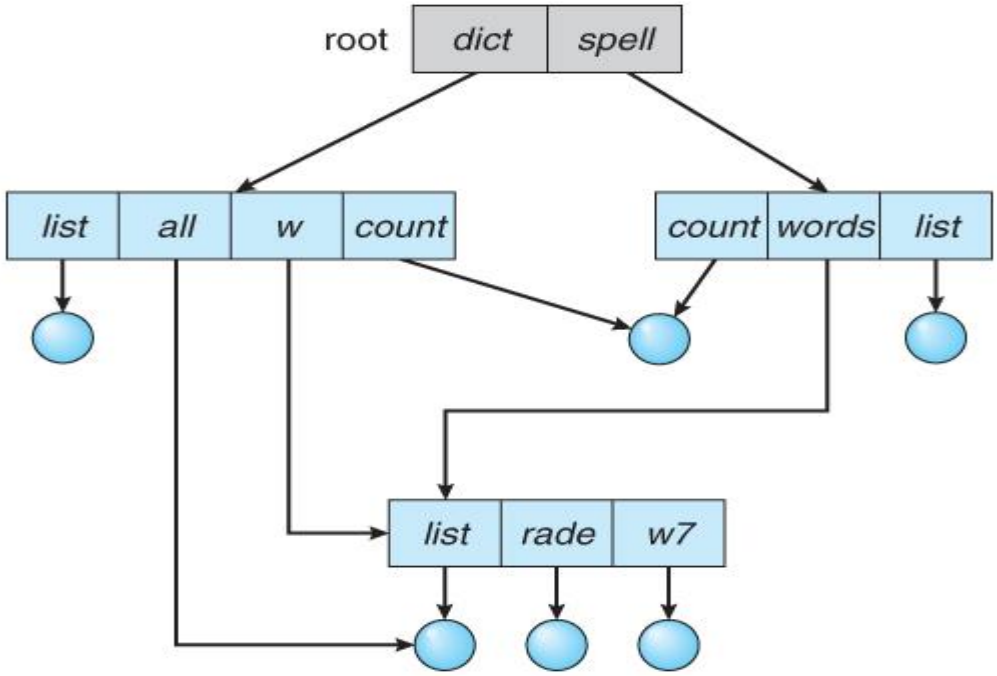
3. **Tree-structured directories:** A tree structure is a more powerful and flexible approach to organize files and directories in hierarchical. There is a master directory, which has under it a number of user directories. Each of these user directories may have subdirectories and files as entries. This is true at any level: That is, at any level, a directory may consist of entries for subdirectories and/or entries for files.



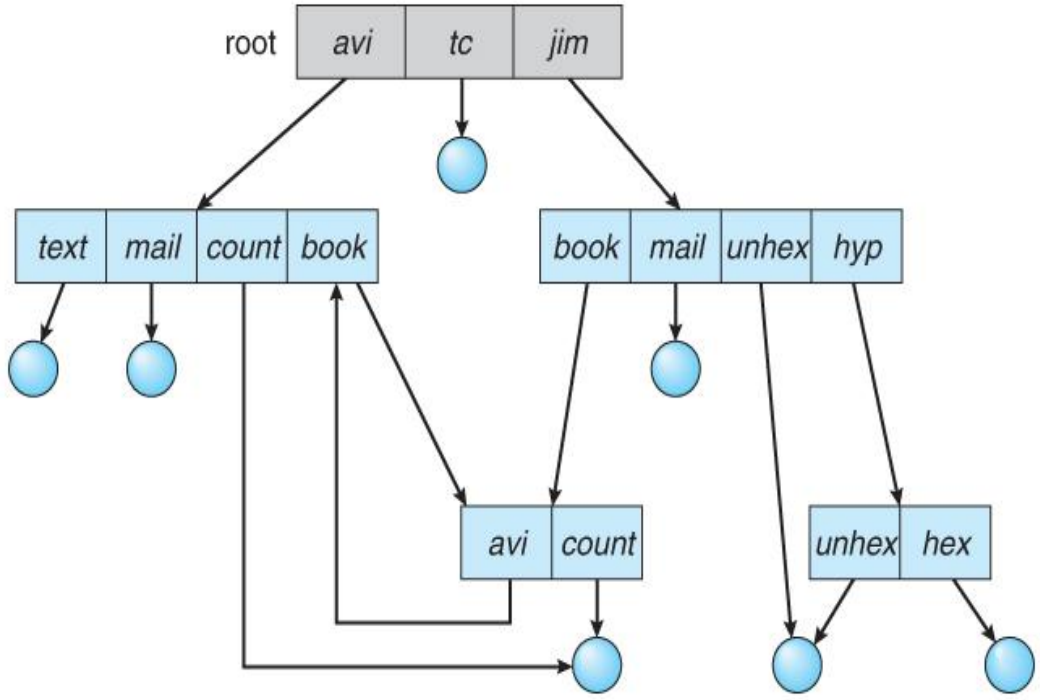
4. **Acyclic-Graph Directories:** An acyclic graph allows directories to have shared subdirectories and files. The same file or subdirectory may be in two different directories. An acyclic graph is a natural generalization of the tree structured directory scheme.

   A shared file (or directory) is not the same as two copies of the file. With two copies, each programmer can view the copy rather than the original, but if one programmer changes the file, the changes will not appear in the other's copy.

   Shared files and subdirectories can be implemented in several ways. A common way is to create a new directory entry called a link. A link is a pointer to another file or subdirectory.



5. **General Graph Directory:** When we add links to an existing tree-structured directory, the tree structure is destroyed, resulting in a simple graph structure.



Difference between File and Directory

File	Directory
It is a collection of different kind of data like text files, picture files, documents, workbook, presentation, music or films etc. in your computer memory in a single unit	It is used to store different files and sub directories.

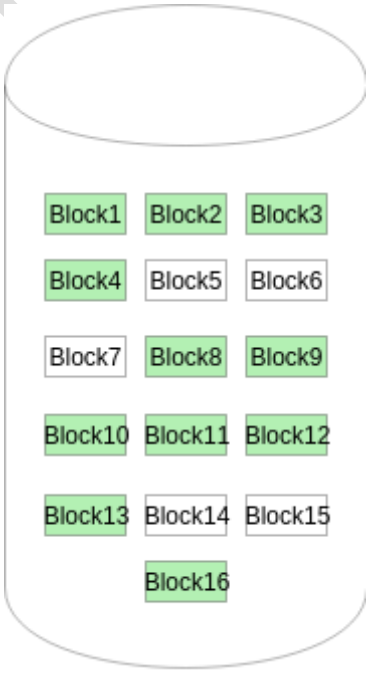
Files are taking spaces on computer memory	Directories are not taking spaces on computer memory.
After creating files one can easily open, save, rename, print, email and modify file contents.	After creating directory one can move rename and delete directory.
One can easily move or copy data from one file to another	One can copy or move files from one directory to another directory.
Each file has its own extension.	A directory does not have any extension.
We cannot create any directory or sub directory within a file	We can create different types of files or directories in a directory.
We cannot share file on network	We can share directory on network.

Free space management

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* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.



Manish Gupta

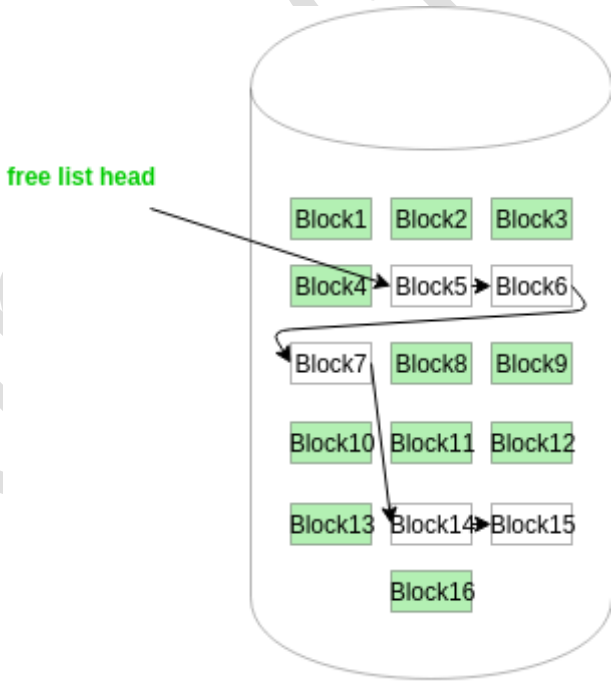
Advantages

- 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*, 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.

2. 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.

3. 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.

4. 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:

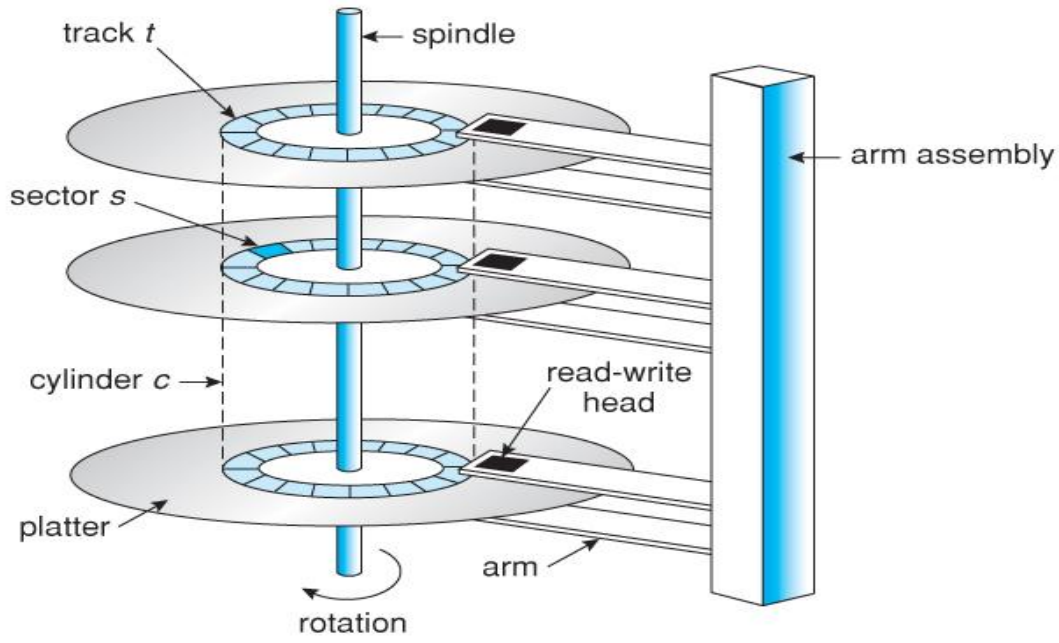
1. Address of first free disk block
2. A number n

For example, *in Figure-1*, the first entry of the free space list would be: ([Address of Block 5], 2), because 2 contiguous free blocks follow block 5.

Secondary Storage Structure

Disks Structure

Magnetic disks provide the bulk of secondary storage for modern computer systems. Each disk **platter** has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 5.25 inches. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters. A read-write head "flies" just above each surface of every platter. The heads are attached to a **disk arm**, which moves all the heads as a unit. The surface of a platter is logically divided into circular **tracks**, which are subdivided into **sectors**. The set of tracks that are at one arm position forms a **cylinder**. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacity of common disk drives is measured in gigabytes.



When the disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 200 times per second. Disk speed has two parts. The transfer rate is the rate at which data flow between the drive and the computer. The positioning time (or random-access time) consists of seek time and rotational latency. The seek time is the time to move the disk arm to the desired cylinder. And the rotational latency is the time for the desired sector to rotate to the disk head. Typical disks can transfer several megabytes of data per second and they have seek times and rotational latencies of several milliseconds.

Capacity of Magnetic disks(C) = S x T x P x N

Where,

S= no. of surfaces = 2 x no. of disks,
T= no. of tracks in a surface,
P= no. of sectors per track,
N= size of each sector

Transfer Time: The transfer time to or from the disk depends on the rotation speed of the disk in the following fashion:

T = b / (r x N)

Where

T= transfer time,
b=number of bytes to be transferred,
N= number of bytes on a track,
r= rotation speed in revolutions per second

Disk Scheduling

The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector. The rotational latency is the time waiting for the disk to rotate the desired sector to the disk head. The disk bandwidth is the total number of bytes transferred divided by the total time between the first request for service and the completion of the last transfer. We can improve both the access time and the bandwidth by scheduling the servicing of disk I/O requests in a good order. Several algorithms exist to schedule the servicing of disk I/O requests as follows:

1. FCFS
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK

Manish Gupta

6. C-LOOK

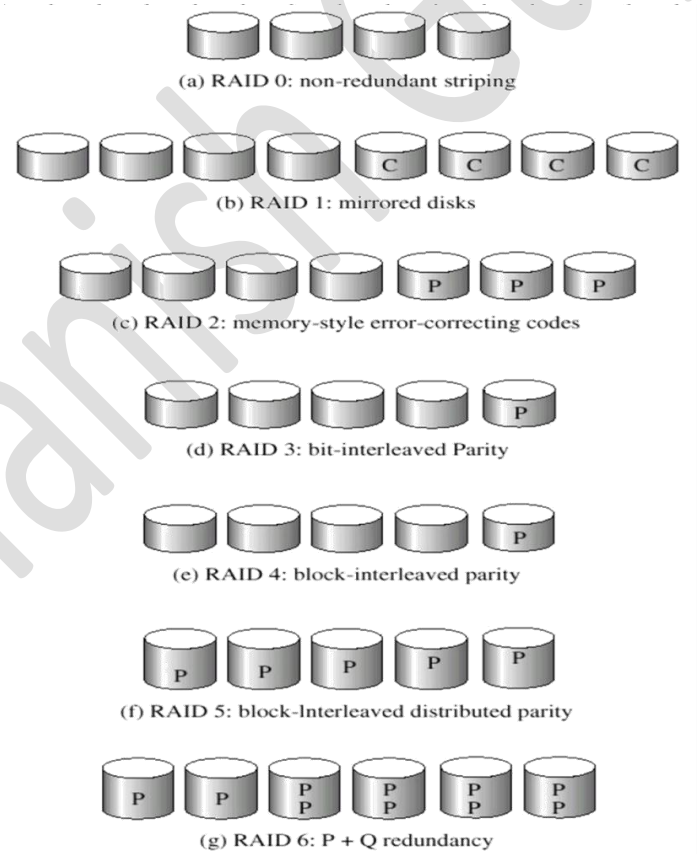
Refer class notes for above algorithms

Raid Structure

A Redundant Array of Inexpensive Disks (RAID) may be used to increase disk reliability. RAID may be implemented in hardware or in the operating system.

The RAID consists of seven levels, zero through six. These levels designate different design architectures that share three common characteristics:

- RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
- Data are distributed across the physical drives of an array in a scheme known as striping, described subsequently.
- Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.



(RAID levels)

(Here P indicates error-correcting bits and C indicates a second copy of the data)

The RAID levels are described as follows:

Manish Gupta

RAID Level 0: RAID level 0 refers to disk arrays with striping at the level of blocks, but without any redundancy (such as parity bits). Figure(a) shows an array of size 4.

RAID Level 1: RAID level 1 refers to disk mirroring. Figure (b) shows a mirrored organization that holds four disks' worth of data.

RAID Level 2: RAID level 2 is also known as memory-style error-correcting code (ECC) organization. Each byte in a memory system may have a parity bit associated with it that records whether the numbers of bits in the byte set to 1 is even (parity=O) or odd (parity=l). The idea of ECC can be used directly in disk arrays via striping of bytes across disks.

RAID level 3: RAID level 3, or bit-interleaved parity organization, improves on level 2 by noting that, disk controllers can detect whether a sector has been read correctly, so a single parity bit can be used for error correction, as well as for detection. The idea is as follows. If one of the sectors gets damaged, we know exactly which sector it is, and, for each bit in the sector, we can figure out whether it is a 1 or a 0 by computing the parity of the corresponding bits from sectors in the other disks. If the parity of the remaining bits is equal to the stored parity, the missing bit is 0; otherwise, it is 1.

RAID Level 4: RAID level 4 or block-interleaved parity organization uses block-level striping, as in RAID 0 and in addition keeps a parity block on a separate disk for corresponding blocks from N other disks. This scheme is shown pictorially in Figure(e). If one of the disks fails, the parity block can be used with the corresponding blocks from the other disks to restore the blocks of the failed disk.

RAID level 5: RAID level 5 or block-interleaved distributed parity is similar as level 4 but level 5 spreading data and parity among all N + 1 disks, rather than storing data in N disks and parity in one disk. For each block, one of the disks stores the parity, and the others store data. By spreading the parity across all the disks in the set, RAID 5 avoids the potential overuse of a single parity disk that can occur with RAID 4.

RAID Level 6: RAID level 6 (is also called the P+Q redundancy scheme) is much like RAID level 5, but stores extra redundant information to guard against multiple disk failures. Instead of using parity, error-correcting codes such as the Reed-Solomon codes are used.

Kernel I/O Subsystem

Kernels provide many services related to I/O. Several services (i.e. scheduling, buffering, caching, spooling, device reservation and error handling) are provided by the kernel's I/O subsystem and build on the hardware and device driver infrastructure.

1. **I/O Scheduling:** It is used to schedule a set of I/O requests that means to determine a good order in which to execute them. Scheduling can improve overall system performance and can reduce the average waiting time for I/O to complete.

Manish Gupta

2. **Buffering:** A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application. Buffering is done for three reasons:
    - One reason is to cope with a speed mismatch between the producer and consumer of a data stream
    - A second use of buffering is to adapt between devices that have different data-transfer sizes.
    - A third use of buffering is to support copy semantics for application I/O.
3. **Caching:** A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. Cache is used to improve the I/O efficiency for files that are shared by applications or that are being written and reread rapidly. The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache just holds a copy on faster storage of an item that resides elsewhere. Caching and buffering are distinct functions, but sometimes a region of memory can be used for both purposes.
4. **Spooling:** A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together. The operating system solves this problem by intercepting all output to the printer. Each application's output is spooled to a separate disk file. When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer. The spooling system copies the queued spool files to the printer one at a time. The operating system provides a control interface that enables users and system administrators to display the queue, to remove unwanted jobs before those jobs print, to suspend printing while the printer is serviced, and so on.
5. **Device Reservation:** It provides support for exclusive device access, by enabling a process to allocate an idle device, and to deallocate that device when it is no longer needed. Many operating systems provide functions that enable processes to coordinate exclusive access among them. It watches out for deadlock to avoid.

Manish Gupta