

3.1. Characteristics of storage Devices:

3.1.1. Physical Characteristics of Disks

1. The storage capacity of a single disk ranges from 10MB to 10TB. A typical commercial database may require hundreds of disks.
2. Disks have moving-head disk mechanism.
 - Each disk *platter* has a flat circular shape. Its two surfaces are covered with a magnetic material and information is recorded on the surfaces. The platters of *hard disks* are made from rigid metal or glass, while *floppy disks* are made from flexible material.
 - The disk surface is logically divided into tracks, which are subdivided into *sectors*. A sector (varying from 32 bytes to 4096 bytes, usually 512 bytes) is the smallest unit of information that can be read from or written to disk. There are 4-32 sectors per track and 20-1500 tracks per disk surface.
 - The arm can be positioned over any one of the tracks.
 - The platter is spun at high speed.
 - To read information, the arm is positioned over the correct track.
 - When the data to be accessed passes under the head, the read or write operation is performed.
3. A disk typically contains multiple platters. The read-write heads of all the tracks are mounted on a single assembly called a *disk arm*, and move together.
 - Multiple disk arms are moved as a unit by the actuator.
 - Each arm has two heads, to read disks above and below it.
 - The set of tracks over which the heads are located forms a cylinder.
 - This cylinder holds that data that is accessible within the disk latency time.
 - It is clearly sensible to store related data in the same or adjacent cylinders.
4. Disk platters range from 1.8" to 14" in diameter, and 5"1/4 and 3"1/2 disks dominate due to the lower cost and faster seek time than do larger disks, yet they provide high storage capacity.
5. A disk controller interfaces between the computer system and the actual hardware of the disk drive. It accepts commands to r/w a sector, and initiate

actions. Disk controllers also attach checksums to each sector to check read error.

6. *Remapping of bad sectors*: If a controller detects that a sector is damaged when the disk is initially formatted, or when an attempt is made to write the sector, it can logically map the sector to a different physical location.
7. *SCSI (Small Computer System Interconnect)* is commonly used to connect disks to PCs and workstations. Mainframe and server systems usually have a faster and more expensive bus to connect to the disks.
8. **Write Mechanism**
 - o Writes binary data by magnetizing small areas or zones of the disk.
 - o Current through the coil produces magnetic field.
 - o Corresponding magnetic pattern is recorded on the surface.
9. **Read Mechanism**
 - o It reads data by detecting current pulses induced in a coil.

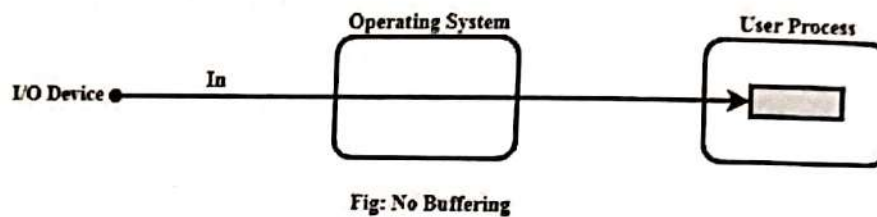
3.1.2. Characteristics Optical Disks:

1. CD-ROM has become a popular medium for distributing software, multimedia data, and other electronic published information.
2. Capacity of CD-ROM: ~ 500 MB. Disks are cheap to mass-produce and drives.
3. CD-ROM: much longer seek time (250m-sec), lower rotation speed (400 rpm), leading to high latency and lower data-transfer rate (about 150 KB/sec).
4. **DVD**, in full **digital videodisc** or **digital versatile disc**, is a type of optical disc used for data storage and as a platform for multimedia. Its most prominent commercial application is for playing back recorded motion pictures and television programs (hence "digital video disc"), though read-only, recordable, and even erasable and rewritable versions can be used on personal computers to store large quantities of almost any kind of data (hence "digital versatile disc").

DVD storage capacity is 4.7 GB for a single-layered, single-sided disc and 8.5 GB for a dual-layered, single-sided disc.

3.2.1.1. No Buffering:

- Process must read/write a device a byte/word a time.
- Each individual system call adds significant overhead.
- Process must wait until each I/O is complete.
 - Blocking /waking adds to overhead
 - Many short run of a process is inefficient.
- What happens if buffer is paged out to disk
 - Could lose data while buffer is paged in
 - Could lock buffer in memory (needed for DMA), however many processes doing I/O reduce RAM available for paging.
 - Can cause deadlock as RAM is limited resource.



The various I/O buffering techniques are as follows:

3.2.1.2. Single buffering:

- When a user process issues an I/O request, the Operating System assigns a buffer in the system portion of main memory to the operation.
- In the block oriented devices, the techniques can be used as follows: Input transfers are made to the system buffer. When the transfer is complete, the process moves the block into user space and request another block.
- This approach will generally provide a speed up compared to the lack of system buffering. The Operating System must keep track of the assignment of system buffers to user processes.
- Similar considerations apply to block oriented output. When data are being transmitted to a device, they are first copied from user space into the system buffer, from which they will ultimately be written. The requesting process is now free to continue.

- Suppose T is the time required to input one block and C is the computation time required for input request.
 - Without buffering: Execution time is $T+C$.
 - Single buffering: Execution time is $\max [C,T]+M$, where M is time required to move the data from system buffer to user memory.

stream oriented Input / Output, it can be used in two ways,

- Line-at a time fashion. Line-at a time operation is used for scroll mode terminals. User inputs one line at a time, with a carriage return signalling at the end of a line.
- Byte-at a time fashion. Byte-at a time operation is used on forms mode, terminals when each keystroke is significant.

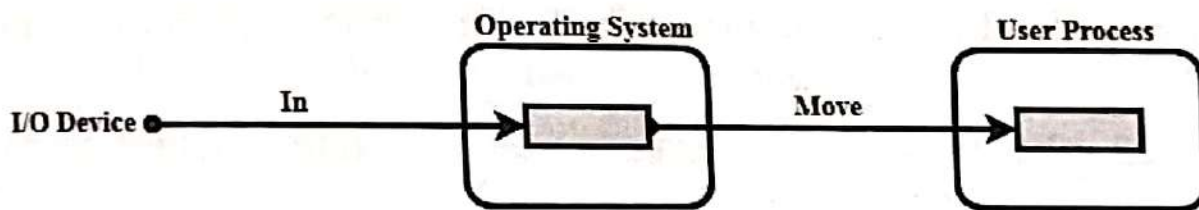


Fig: Single Buffering

3.2.1.3. Double buffering or buffer swapping:

- An improvement over single buffering is by assigning two system buffers to the operations.
- A process transfers data to one buffer while operating system empties the other as shown in fig.
- For block oriented transfer execution time is $\text{Max}[C,T]$. It is possible to keep the block oriented device going at full speed.
 - If $C \leq T$, i.e. computation time is less than the time required to input one block.
 - If $C > T$, i.e. computation time is greater than the time required to input one block, then double buffering ensures that the process will not have to wait on I/O.

For Stream oriented Input / Output again two types.

- For line- at a time I/O, the user process need not be suspended for input or output, unless process runs ahead of double buffer.
- For byte- at a time operations, double buffer offers no advantage over a buffer of twice the length.

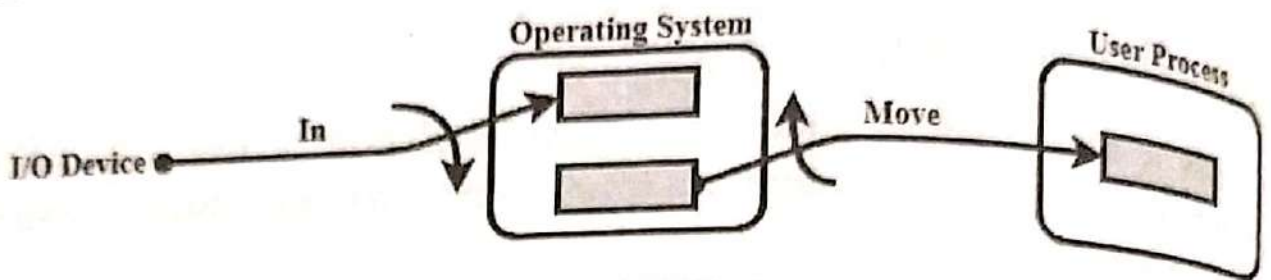


Fig: Double Buffering

3.2.1.4. Circular buffer:

- Double buffering may be inadequate, if the process performs rapid byte- at a time I/O. When two or more buffers are used.
- The collection of buffers are called as a circular buffer, with each buffer one unit in the circular buffer.

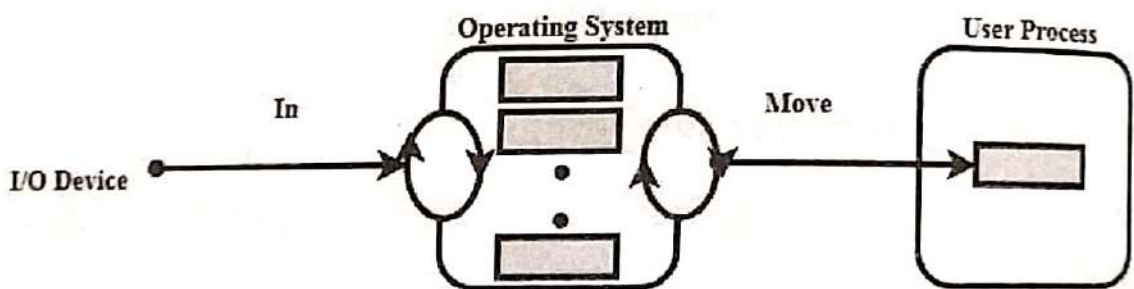


Fig: Circular Buffering

3.3. Basic File System Operations:

A file is a named collection of related information, usually as a sequence of bytes, with two views:

1. Logical (Programmer's) view, as the users see it.
2. Physical (operating system) view, as it actually resides on secondary storage.

Files are intended to be non-volatile; hence in principle, they are long lasting. Files are intended to be moved around (i.e., copied from one place to another) accessed by different programs and users, and so on.

Opening and Closing Files:

Regardless of how the descriptive information about a file is organized, most of it is maintained on disk. When the file is to be used, relevant portions of this information is brought into main memory for efficient continued access to the file data. For that purpose, the file system maintains an *open file table* (OFT) to keep track of currently open files. Each entry in the OFT corresponds to one open file, i.e., a file being used by one or more processes. The OFT is managed by the open and close functions. The open function is invoked whenever a process first wishes to access a file. The function finds and allocates a free entry in the OFT, fills it with relevant information about the file, and associates with it any resources, e.g., read/write buffers, necessary to access the file efficiently.

Some systems do not require an explicit open command; in that case, it is generated implicitly by the system at the time the file is accessed for the first time. When a file is no longer needed, it is closed either by calling the close command, or implicitly as the result of a process termination. The close function frees all resources used for accessing the file, saves all modified information to the disk, and releases the OFT entry, thereby rendering the file inactive for the particular process.

The implementation of the open and close functions differ widely with different file systems, but the following list gives the typical tasks performed in some form as part of these functions.

3.3.1. Open Operation:

- Using the protection information in the file descriptor, verify that the process (user) has the right to access the file and perform the specified operations.
- Find and allocate a free entry in the OFT.
- Allocate read/write buffers in main memory and other resources as necessary for the given type of file access.
- Complete the components of the OFT entry. This includes initialization information, such as the current position (zero) of a sequentially accessed file. It also includes relevant information copied from the file descriptor, such as the file length and its location on the disk. Additional runtime information, such as the pointers to the allocated buffers or other resources, also is placed into the OFT entry.

- If all of the above operations are successful, return the index or the pointer to the allocated OFT entry to the calling process for subsequent access to the file.

3.3.2. Close Operation:

- Flush any modified main memory buffers by writing their contents to the corresponding disk blocks.
- Release all buffers and other allocated resources.
- Update the file descriptor using the current data in the OFT entry. This could include any changes to the file length, allocation of disk blocks, or use information (e.g., date of last access/modification).
- Free the OFT entry.

3.3.3. Create Operation

Creation of the file is the most important operation on the file. Different types of files are created by different methods for example text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files. The file is created with no data. The file create operation is the first step of the file. Without creating any file, there is no any operation can be performed.

- Required space for the file must be allocated.
- An entry for new file must be made in the directory.
- The directory entry records the name of the file and the location in the file system.

3.3.4. Delete Operation

Deleting the file will not only delete all the data stored inside the file, It also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files. File must has to be deleted when it is no longer needed just to free up the disk space. The file delete operation is the last step of the file. After deleting the file, it doesn't exist.

- File deletion operation also requires searching of a specified file entry within the directory structure.
- As soon as the file is deleted, space allocated to that file becomes available for further use.

3.3.5. Append Operation:

This operation is a restricted form of write. It can only add data to the end of the file. Systems that provide a minimal set of system calls do not generally have append, but many systems provide multiple ways of doing the same thing, and these systems sometimes have append.

- The file append operation is same as the file write operation except that the file append operation only add the data at the end of the file.

3.4. File Protection Mechanisms:

All multiuser operating systems must provide some minimal protection to files to keep one user from intentionally or unintentionally accessing or modifying the files of another.

Files often contain information that is highly valuable to their users. One of the major functions of the file system is to protect this information against unauthorized access and physical damage. Physical damage may occur because of hardware problems, power failure, head crashes, dirt and extreme temperatures. In order to prevent such damage some systems performs backup at regular intervals. Protection is achieved by limiting the type of file access which can be made. Access is permitted or denied depending upon several factors, one of which is the type of access requested. Several operations on files can be controlled.

Some of the file operations are:

- **read** - read a file
- **write** - write a file
- **execute** - load and execute a file
- **append** - append information at the end of a file
- **delete** - free the space allocated to a file
- **update** - modifying, deleting and adding to a file
- **copy** - copy the contents of a file
- **list** - listing the name of the file

The most common implementation of the file systems allow the owners of the file to do operations read, write, execute, append and delete, whereas other users can only invoke those operations that do not modify the file, e.g., read. However, in some