5.1 INTRODUCTION

- A file can be described as a container that stores data that is accessible by a computer system - it is basically a container holding a set of related information that will be stored on some form of secondary storage. A data file may consist of for example, a set of computer program instructions, text or another form of data.
- A text file may contain just a short letter, or contain a complete book. In other words a file is basically a defined set of named data.
- A file is a named collection of related information that is recorded on secondary storage. Also, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.
- Information may be stored in are :
  - file-source programs,
  - object programs,
  - executable programs,
  - numeric data,
  - text,
  - payroll records,
  - graphic images,
  - Sound recordings.

5.1.1 Basic Concept of File System

UQ. 5.1.1 Write short note on File System. MU(IT) - Q. 5(c), May 18, 5 Marks

The file system provides the resource abstractions typically associated with secondary storage. The file system permits users to create data collections, called files, with desirable properties, such as

- **Long-term existence :** Files are stored on disk or other secondary storage and do not disappear when a user logs off.
- **Sharable between processes :** Files have names and can have associated access permissions that permit controlled sharing.
- **Structure :** Depending on the file system, a file can have an internal structure that is convenient for particular applications. In addition, files can be organized into hierarchical or more complex structure to reflect the Relationships among files.

File Naming

- The rules for file naming vary somewhat from system to system, but all current operating systems allow strings of one to eight letters as legal file names (in MSDOS).
- Many file systems support names as long as 255 characters (in Windows). Unix considers upper and lower case as different while MSDOS consider both as same. Hence techneo.txt and TECHNEO.txt will be different file in Unix but in MSDOS both will considered as same.
- Many operating systems support two-part file names, with the two parts separated by a period, as in *techneo.txt*. The part following the period is called the **file extension** and usually indicates something about the file.
- In MS-DOS, for example, file names are 1 to 8 characters, plus an optional extension of 1 to 3 characters.
- In UNIX, the size of the extension, if any, is up to the user, and a file may even have two or more extensions, as in *manoj.html.zip*, where *.html* indicates a Web page in HTML and *.zip* indicates that the file (*manoj.html*) has been compressed using the *zip* program.

- Some of the more common file extensions and their meanings are shown in Fig. 5.1.1.

Extension	Meaning
.bak	Backup file
.c	C source program
.gif	Compuserve Graphical Interchange Format Image
.html	World Wide Web Hyper Text markup Language document
.jpg	Still picture encoded with the JPEG standard
.mp3	Music encoded in MPEG layer 3 audio format
.mpg	Movie encoded with the MPEG standard
.o , .obj	Object file (compiler output, not yet linked)
.pdf	Portable Document Format file
.ps	PostScript file
.txt	General text file
.zip	Compressed archive

(IET)**Fig. 5.1.1 : Example of File Extension**

5.1.2 File Attributes

A file is named, for the convenience of its human users, and is referred to by its name. A name is usually a string of characters, such as *examp1e.java or .vbp*. Some systems differentiate between upper- and lowercase characters in names, whereas other systems consider the two cases to be equivalent. The file system maintains a set of attributes associated with the file. A file has certain other attributes, which vary from one operating system to another, but typically consist of these :

1. Name	2 Identifier	3. Type	4. Location	5. Size
6. Protection	7. Time of creation	8. Date of creation, and	9. User identification	

- **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 of creation and user identification :**
  - This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.
  - The information about all files is kept in the directory structure that also resides on secondary storage. Typically, the directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes.
  - It may take more than a kilobyte to record this information for each file. In a system with many files, the size of the directory itself may be megabytes.

Module 5

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/Binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Data and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

5.1.3 File Operation

UQ. 5.1.2 List and explain different operation performed on File. MU(IT) - Q. 1(d), May 16, 5 Marks

Any file system provides not only a means to store data organized as files, but a collection of functions that can be performed on files. Typical operations include the a following:

1. Create 2. Write 3. Read 4. Reposition 5. Delete, and 6. Truncate files.

1. **Creating a file :** Two steps are necessary to create a file.
   - Space in the file system must be found for the file.
   - 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.
3. **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 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 *seek*.

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.

Other operation which are performed on file

Other common operations include

1. Appending a file 2. Renaming a file 3. Creating a Copy 4. Searching

1. **Appending a File :** *An appending* new information to the end of an existing file. This system call is a restricted form of write. It can add data only to the end of the file.

2. **Renaming a File :** *Renaming* an existing file i.e. changing the name of the file for example vipul.txt is renamed to new name vipul.pub.

3. **Creating a copy of File :** Basic operations may then be combined to perform other file operations. For instance, creating a *copy* of a file, or copying the file to another I/O device, such as a printer or a display, may be accomplished by creating a new file, and reading from the old and writing to the new.

4. **Searching :** The file operations mentioned involve searching the directory for the entry associated with the named file. To avoid this constant searching, many systems require that an open system call be used before that file is first used actively. The operating system keeps a small table containing information about all open files (the **open-file table**).

5.1.4 File Structure

Four terms are in common use when discussing files

1. Field 2. Record 3. File 4. Database

1. **Field**

- A **field is basic unit in which can be stored.** It contains a single value, such as an Student name, a date of birth, or the it store the data from the sensor, gender etc.
- The data stored in this field can defined by its length and data type (e.g., ASCII string, decimal). Depending on the file design, fields may be fixed length or variable length.

2. **Record**

- When related fields are collected together it becomes a record and it can be considered as unit identity by application program. Let say for example, an student details record can be made using fields as name, Roll number, Class, date of birth, gender, and so on. Records may be of fixed length or variable length.

3. **File**

- A **file** is a collection of similar records. If file does not have record i.e. related field in such case file is collection of field.
- The file is treated as a single entity by users and applications. File is referenced by name. Files can be created or deleted. Access control restrictions usually apply at the file level.

Module 5

4. Database

- A **database** is a collection of related data. The essential aspects of a database are that the relationships that exist among elements of data are explicit
- The database is designed for use by a number of different applications. A database may contain all of the information related to an organization or project, such as a Office, shop, or a geographic study or scientific study.

5.1.5 Internal File Structure

- Internally, locating an offset within a file can be complicated for the operating system. Disk systems typically have a well-defined block size determined by the size of a sector. All disk I/O is performed in units of one block (physical record), and all blocks are the same size.
- It is unlikely that the physical record size will exactly match the length of the desired logical record. Logical records may even vary in length. Packing a number of logical records into physical blocks is a common solution to this problem.
- For example, the UNIX operating system defines all files to be simply streams of bytes. Each byte is individually addressable by its offset from the beginning (or end) of the file. In this case, the logical record size is 1 byte. The file system automatically packs and unpacks bytes into physical disk blocks say, 512 bytes per block's as necessary
- The logical record size, physical block size, and packing technique determine how many logical records are in each physical block. The packing can be done either by the user's application program or by the operating system.
- In either case, the file may be considered to be a sequence of blocks. All the basic I/O functions operate in terms of blocks. The conversion from logical records to physical blocks is a relatively simple software problem.
- Because disk space is always allocated in blocks, some portion of the last block of each file is generally wasted. If each block were 512 bytes, for example, then a file of 1,949 bytes would be allocated four blocks (2,048 bytes); the last 99 bytes would be wasted.
- The waste incurred to keep everything in units of blocks (instead of bytes) is internal fragmentation. All file systems suffer from internal fragmentation; the larger the block size, the greater the internal fragmentation.

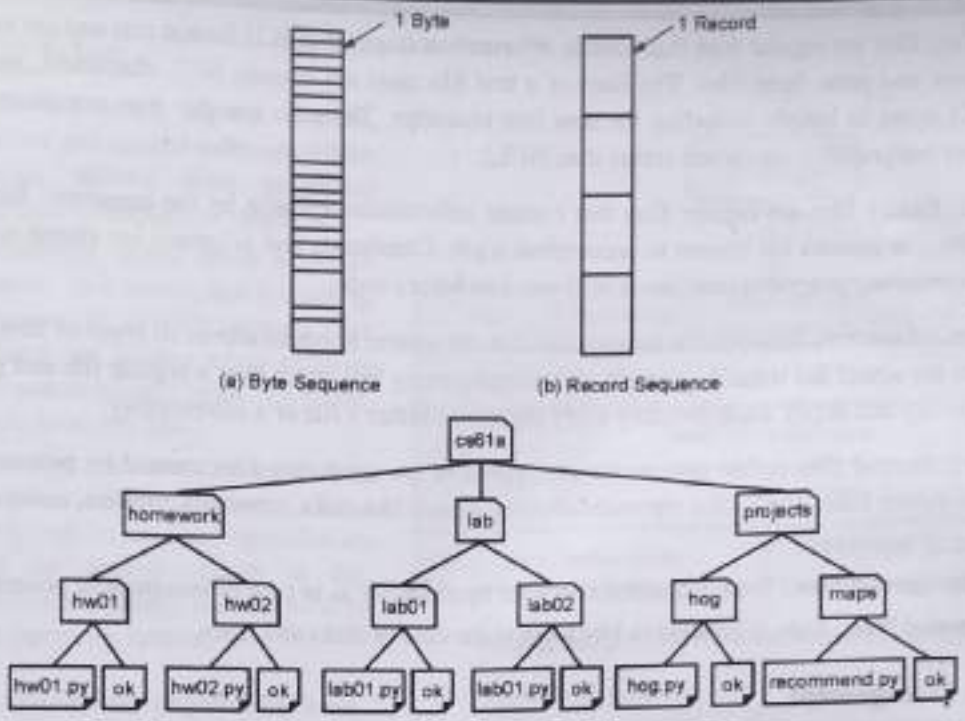
5.1.6 Types of File Structure

UQ. 5.1.3 List different File structure and explain any two. MU(IT) - Q. 5(c), May 16, 10 Marks

- A file has a certain defined **structure** according to its type. For example,
  1. A *text* file is a sequence of characters organized into lines (and possibly pages).
  2. A *source* file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.
  3. An *object* file is a sequence of bytes organized into blocks understandable by the system's linker.
  4. An *executable* file is a series of code sections that the loader can bring into memory and execute.
  5. 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 any of the are shown Fig. 5.1.2 several ways. The three common possibilities are as follows :

a. Stream of Bytes	b. Records	c. Tree

a. Stream of Bytes : For operating system regard files is just sequences of bytes and provides the maximum amount of flexibility. User programs can put any type of data they want in their files and accordingly. All versions of UNIX (including Linux and OS X) and Windows use this file model. The main advantage of such file structuring is that it simplifies file management for the operating system and Applications can have their own structure.



(182)**Fig. 5.1.2 : Types of File Structure**

- **b. Record File Structure :** A file is a sequence of fixed length record, each with some Internal structure shown in Fig. 5.2.1(b) Collection of bytes is treated as a unit. For example, employee record. Operations are at the level of records (read_record, write_record). File is a collection of similar records. Operating System can optimize operations on records. No current general-purpose system uses this model as its primary file system any more, but back in the days of 80-column punched cards and 132-character line printer paper this was a common model on mainframe computers.

- **c. Tree File Structure :** In this organization, a file 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 sorted on the key field, to allow rapid searching for a particular key.

5.1.7 File Type

The types of files recognized by the system are either **regular**, **directory**, or **special**. However, the operating system uses many variations of these basic types. The following basic types of files exist :

1	**Regular**	Contain user information, Stores data (text, binary, and executable). All files are regular file.
2	**Directory**	Contains information used to access other files. for maintaining the structure of the file system.
3	**Special**	Character special files are related to input/output and used to model serial I/O devices, such as terminals, printers, and networks. Block special files are used to model disks.

Module 5

1. **Regular files :** Regular files are the most common files and are used to contain data. Regular files are in the form of text files or binary files :
   - Regular files are generally either ASCII files or binary files. ASCII files consist of lines of text. In some systems each line is terminated by a carriage return character. In others, the line feed character is used. Some systems (e.g., Windows) use both. Lines need not all be of the same length.
   - The great advantage of ASCII files is that they can be displayed and printed as is, and they can be edited with any text editor.

2. **Text files :** Text files are regular files that contain information stored in ASCII format text and are readable by the user. You can display and print these files. The lines of a text file must not contain NUL characters, and none can exceed [LINE_MAX] bytes in length, including the new line character. The term *text file* does not prevent the inclusion of control or other nonprintable characters (other than NUL).

3. **Binary files :** Binary files are regular files that contain information readable by the computer. Binary files might be executable files that instruct the system to accomplish a job. Commands and programs are stored in executable, binary files. Special compiling programs translate ASCII text into binary code.

4. **Directory files :** Directory files contain information that the system needs to access all types of files, but directory files do not contain the actual file data. As a result, directories occupy less space than a regular file and give the file system structure flexibility and depth. Each directory entry represents either a file or a subdirectory.

5. **Special files :** Special files define devices for the system or are temporary files created by processes. These files are also known as device files. These files represent physical device like disks, terminals, printers, networks, tape drive etc. These files are of two types :
   - **Character special files :** Data is handled character by character as in case of terminals or printers.
   - **Block special files :** Data is handled in blocks as in the case of disks and tapes.

5.1.8 File Management

- A file management system is that set of system software that provides services to users and applications in the use of files. User or Application program can access files is through the file management system. Out of many functions of operating system, File Management is one of the function.
- The following are the objectives for a file management system :
  - To provide a standardized set of I/O interface routines to user processes.
  - To meet the data management needs and requirements of the user, which include storage of data and the ability to perform therefore mentioned operations.
  - To guarantee, to the extent possible, that the data in the file are valid.
  - To provide I/O support for a variety of storage device types.
  - To optimize performance, both from the system point of view in terms of overall throughput and from the user's point of view in terms of response time.
  - To provide I/O support for multiple users, in the case of multiple-user systems.
  - To minimize or eliminate the potential for lost or destroyed data.
- For an interactive, general-purpose system, the following constitute a minimal set of requirements :
  - Each user should be able to create, delete, read, write, and modify files.
  - Each user may have controlled access to other users' files.
  - Each user may control what types of accesses are allowed to the user's files.
  - Each user should be able to restructure the user's files in a form appropriate to the problem.
  - Each user should be able to move data between files.
  - Each user should be able to back up and recover the user's files in case of damage.
  - Each user should be able to access his or her files by name rather than by numeric identifier.

5.1.9 File System Architecture

Fig. 5.1.3 shows the file system software architecture. It consists of :

1. Disk Device Driver	2. Basic File System	3. Basic IO Supervisor	4. Logical IO

1. Disk Device Driver

- At the lowest level, **device drivers. Device driver are special software which** communicate directly with peripheral devices or their controllers or channels.
- The responsibility device driver is to start I/O operations on a device and processing the completion of an I/O request. Like Device driver are needed when software want to communicate with CDROM, Pendrive etc to access file.
- For file storage, devices like hard disk, floppy disk, CDROM, tape drives pendrive etc. can be used and operation are performed on devices related to file. Device drivers are usually considered to be part of the operating system.

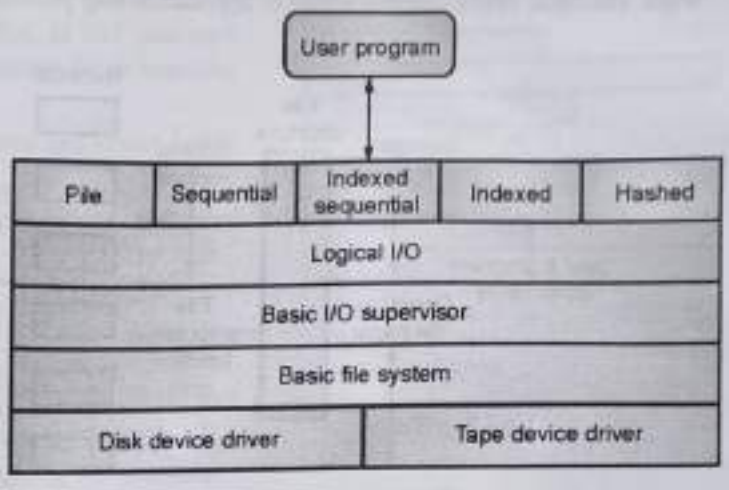


Fig. 5.1.3 : File System Software Architecture

2. Basic file system / the physical I/O level

- This is the primary interface with the environment outside of the computer system.It helps in exchanging blocks of data with disk or other storage systems.
- The placement of those blocks on the secondary storage device and on the buffering of those blocks in main memory is done basic file system.
- It does not understand the content of the data or the structure of the files involved. The basic file system is often part of the operating system.

3. Basic IO Supervisor

- All file I/O initiation and termination is done by basic IO supervisor. At this level, control structures are maintained that deal with device I/O, scheduling, and file status.
- The basic I/O supervisor selects the device on which file I/O is to be performed, based on the particular file selected. It is also concerned with scheduling disk and tape accesses to optimize performance.
- I/O buffers are assigned and secondary memory is allocated at this level. The basic I/O supervisor is part of the operating system.

4. Logical I/O

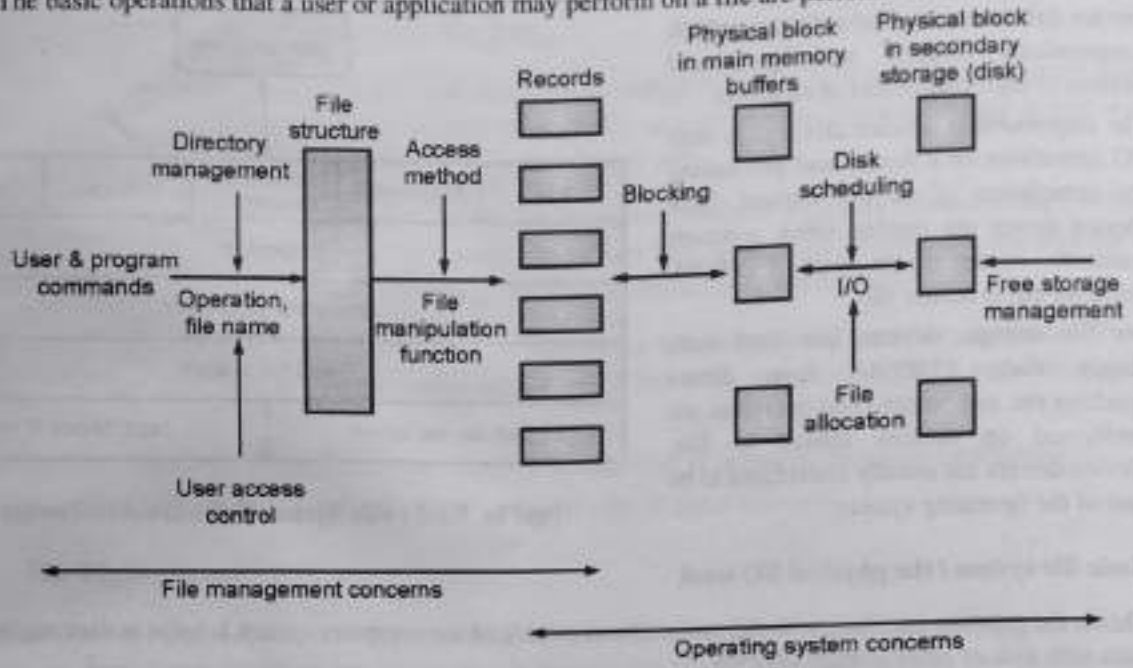
- It enables users and applications to access records. The basic file system deals with blocks of data, the logical I/O module deals with file records.
- Logical I/O provides a general-purpose record I/O capability and maintains basic data about files.The level of the file system closest to the user is often termed the **access method**.
- It provides a standard interface between applications and the file systems and devices that hold the data.
- Different access methods are used for different file structures and there are different ways of accessing and processing the data.

Module 5

5.1.10 File Management Functions

- Users and application programs interact with the file system using commands for creating and deleting files and for performing different operations on files. Any operation on the file can be performed after the file system identify and locate the selected file.

- This requires the use of some sort of directory that serves to describe the location of all files, plus their attributes. Also shared systems apply user access control i.e. only authorized users are allowed to access particular files in particular ways. The basic operations that a user or application may perform on a file are performed at the record level.



(164)**Fig. 5.1.4 : Element of File Management**

- The user or application views the file as having some structure that organizes the records, such as a sequential structure (e.g., personnel records are stored alphabetically by last name).
- Thus, to translate user commands into specific file manipulation commands, the access method appropriate to this file structure must be employed.
- Whereas users and applications are concerned with records or fields, I/O is done on a block basis.Thus, the records or fields of a file must be organized as a sequence of in various systems.

5.2 ACCESS METHODS

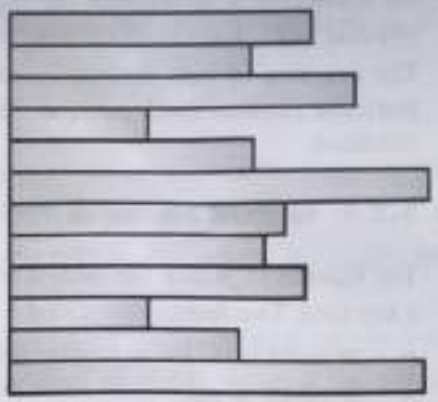
- *File organization* means the logical structuring of the records as determined by the way in which they are accessed.
- The physical organization of the file on secondary storage depends on the blocking strategy and the file allocation strategy. Following are the important criteria in selecting a file organization :
  1. Short access time 2. Ease of update 3. Economy of storage
  4. Simple maintenance 5. Reliability
- These criteria's depend on the applications in which the file is used. For example, in batch system if the file is used, where all the records are accessed every time, then rapid access for retrieval of a single record is not of that importance. A file stored on CD-ROM will never be updated is always read so ease of update is not an important issue.
- Most structures used in actual systems either fall into one of these categories or can be implemented with a combination of these five organizations.

1. Pile File	2. Sequential file	3. Indexed sequential file
4. Indexed file	5. Direct, or hashed, file	

5.2.1 Pile File

- Data are collected in the order in which they arrive i.e. they are piled. Each record consists of one burst of data.
- The purpose of the pile is simply to accumulate the mass of data and save it. Records may have different fields, or similar fields in different orders.

- Thus, each field should be self-describing, including a field name as well as a value.
- Record access is time consuming since it does not have any data structure. Pile is least complicated type of file organization. Pile file structure is shown if Fig. 5.2.1.
- If any record is to searched with a particular value, in that case each record is to checked till that record is found or complete file is searched. It is same as Linear search of a data in a array.
- Pile files are used generally when data are collected and stored before processing or data organization is not easy.
- Even though its pile files are easy to update their applications are limited and not mostly used.



Variable-length records
Variable set of fields
Chronological order

(1E5)Fig. 5.2.1 : Pile File

5.2.2 Sequential File

- Sequential file organization is one of the most common file structure. It is fixed format structure in which all records and field-length are fixed. Since it uses fixed format all record in sequential file structure is of same length.
- In this structure the field name and length for each field are attributes of the file structure is only parameter to be stored because we know length and position of each field. Fig. 5.2.2 Shows sequential File Structure.
- One field generally the first field in each record, is referred to as the **key field**. The key field is used to uniquely identify the record. The key values are different records for different records.
- Further, the records are stored in key sequence :
  - alphabetical order for a text key, and
  - numerical order for a numerical key.
- Sequential files are typically used in batch applications and are generally optimum for such applications if they involve the processing of all the records (e.g., a billing or payroll application).
- The sequential file organization is the only one that is easily stored on tape as well as disk.

Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(1E6)Fig. 5.2.2 : Sequential File Structure

Disadvantages of Sequential File Structure

1. Sequential file performance is poor for interactive applications that involve queries and/or updates of individual records.
2. Access requires the sequential search of the file for a key match. If the entire file, or a large portion of the file, can be brought into main memory at one time, more time and memory is needed for storing and searching.
3. Also processing and delay are is additional disadvantage encountered to access a record in a large sequential file.

Module 5

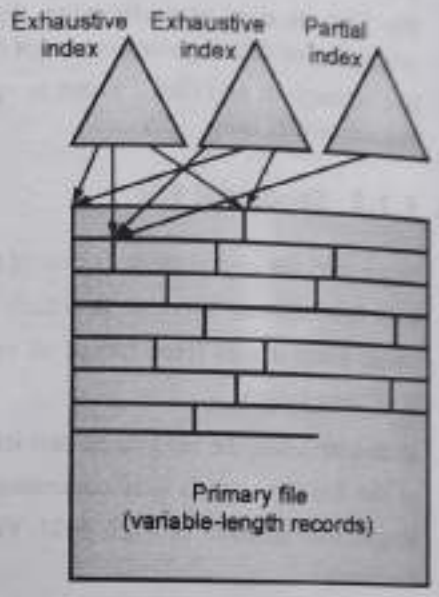
Issue with Sequential File

- Typically, a sequential file is stored in simple sequential ordering of the records within blocks. That is, the physical organization of the file on tape or disk directly matches the logical organization of the file.
- In this case, the usual procedure is to place new records in a separate pile file, called a log file or transaction file.
- Periodically, a batch update is performed that merges the log file with the master file to produce a new file in correct key sequence.

- An alternative is to organize the sequential file physically as a linked list. One or more records are stored in each physical block. Each block on disk contains a pointer to the next block.
- The insertion of new records involves pointer manipulation but does not require that the new records occupy a particular physical block position. Thus, some added convenience is obtained at the cost of additional processing and overhead.

5.2.3 Indexed Sequential File

- The indexed sequential file records are organized in sequence based on a key field. Two features are added :
  - an index to the file to support random access, and
  - an overflow file.
- The index provides a lookup capability to record faster.
- The overflow file is similar to the log file used with a sequential file but is integrated so that a record in the overflow file is located by following a pointer from its predecessor record.
- Fig. 5.2.3 shows indexed sequential file Structure.
- Index sequential structure may have single level or Multilevel indexing to improve efficiency.
- The single level is simple sequential file of indexing.
- In this each record consists of two fields :
  - a key field, which is the same as the key field in the main file, and
  - a pointer into the main file.



(1E7)Fig. 5.2.3 : Indexed Sequential File Structure

- To find a specific field, the index is searched to find the highest key value that is equal to or precedes the desired key value. The search continues in the main file at the location indicated by the pointer.

Benefits of Index Sequential File Structure

a) Searching Record

- Let say for example sequential file has 1 million records. In order to search for a particular key value on average one-half 10 million record accesses is required. Now suppose that an index created is with1000 entries, with the keys in the index more or less evenly distributed over the main file.
- This technique will take on average 500 accesses to the index file followed by 500 accesses to the main file to find the record. Hence the average search length is reduced from 500,000 to 1000, Searching becomes faster.

b) File handling

- Each record in the main file contains an additional field not visible to the application, which is a pointer to the overflow file. When a new record is to be inserted into the file, it is added to the overflow file.
- The record in the main file that immediately precedes the new record in logical sequence is updated to contain a pointer to the new record in the overflow file.
- If the immediately preceding record is itself in the overflow file, then the pointer in that record is updated.As with the sequential file, the indexed sequential file is occasionally merged with the overflow file in batch mode.

c) Time Saving

- The indexed sequential file greatly reduces the time required to access a single record, without sacrificing the sequential nature of the file.

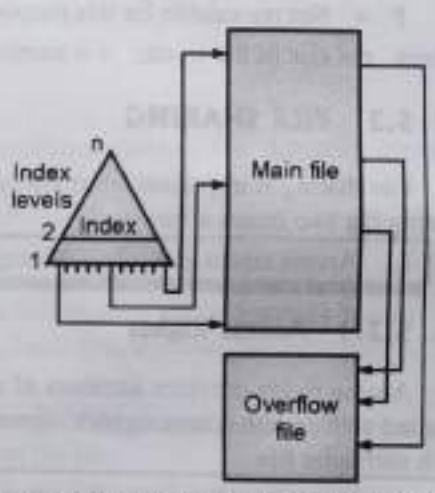
- To process the entire file sequentially, the records of the main file are processed in sequence until a pointer to the overflow file is found, then accessing continues in the overflow file until a null pointer is encountered, at which time accessing of the main file is resumed where it left off.

d) Improved Efficiency

- To provide even greater efficiency in access, multiple levels of indexing can be used. Thus the lowest level of index file is treated as a sequential file and a higher-level index file is created for that file.
- Consider again a file with 1 million records. A lower-level index is created with 10,000 entries. In multiple index higher-level index into the lower level index is created with 100 entries.
- The searching starts at the higher-level index (average length = 50 accesses) to find an entry point into the lower-level index. This index is then searched (average length = 50) to find an entry point into the main file, which is then searched (average length = 50).Thus the average length of search has been reduced from 500,000 to 1000 to 150.

5.2.4 Indexed File

- Limitation of the indexed sequential file is effective processing is limited to that which is based on a single field of the file. In both sequential access searching of record is done on key field.
- But when it is needed to search a record on some other attribute than the key field, both forms of sequential file are inadequate.
- Searching on various attribute many time provide efficiency and flexibility. To achieve this a structure with multiple indexes is needed, one for each type of field that may be the subject of a search.
- Records are accessed only through their indexes, in this concept of single index for search is not used. The advantage of this technique is that there is now no restriction on the placement of records as long as a pointer in at least one index refers to that record.



(TE)Fig. 5.2.4 : Indexed File Structure

- Furthermore, variable-length records can be employed as compared to fixed length in both sequential structure. Two types of indexes are used.

  1. Exhaustive Index 2. Partial Index

- An exhaustive index contains one entry for every record in the main file like key field. The index itself is organized as a sequential file for ease of searching.
- A partial index contains entries to records where the field of interest exists. With variable-length records, some records will not contain all fields.
- When a new record is added to the main file, all of the index files must be updated. Indexed files are used mostly in applications where timeliness of information is critical and where data are rarely processed exhaustively. Examples are Railway reservation systems, Airways reservation and inventory control systems.

Module 5

5.2.5 Direct or Hashed File

- The direct, or hashed, file exploits the capability found on disks to access directly any block of a known address.
- As with sequential and indexed sequential files, a key field is required in each record. However, there is no concept of sequential ordering here. The direct file makes use of hashing on the key value.
- Direct files are often used where very rapid access is required, where fixed length records are used, and where records are always accessed one at a time. Examples are directories, pricing tables, schedules, and name lists.

File Method	Space Attributes		Update Record Size		Retrieval		
	Variable	Fixed	Equal	Greater	Single Record	Subset	Exhaustive
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed Sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

A = Excellent, well suited to this purpose = $O(r)$

B = Good = $O(o \times r)$

C = Adequate = $O(r \log n)$

D = Requires some extra effort = $O(n)$

E = Possible with extreme effort = $O(r \times n)$

F = Not reasonable for this purpose = $O(n^{>1})$

Where r = size of the result; o = number of records that overflow; n = number of records in file

5.3 FILE SHARING

File sharing is important when the system is multiuser, where file is shared among a number of users. In multiuser while sharing file two issues arise:

1. Access rights and 2. Management of simultaneous access

5.3.1 Access Rights

Access rights provides numbers of options in which way a particular file can be accessed. User or group of users are granted with certain access rights to access the file. Following are list of access right which can be assigned to particular user with particular file.

1. None 2. Knowledge 3. Execution 4. Reading 5. Appending 6. Updating
7. Changing Protection 8. Deletion 9. Specific user 10. User group 11. All

1. **None :** To enforce this restriction, the user would not be allowed to read the user directory that includes this file.
2. **Knowledge :** The user can determine that the file exists and who its owner is. The user is then able to petition the owner for additional access rights.
3. **Execution :** The user can load and execute a program but cannot copy it. Proprietary programs are often made accessible with this restriction.
4. **Reading :** The user can read the file for any purpose, including copying and execution. Some systems are able to enforce a distinction between viewing and copying. In the former case, the contents of the file can be displayed to the user, but the user has no means for making a copy.
5. **Appending :** The user can add data to the file, often only at the end, but cannot modify or delete any of the file's contents. This right is used when data is to be collected from a number of sources.
6. **Updating :** The user can modify, delete, and add to the file's data. This normally includes writing the file initially, rewriting it completely or in part, and removing all or a portion of the data.
7. **Changing protection :** This right is held by the owner of the file and it may be extended to others by the owner. The user can change the access rights granted to other users. To prevent abuse of this mechanism, the file owner will decide which rights can be changed by the holder of this right.

8. **Deletion :** The user can delete the file from the file system. These rights can be considered to constitute a hierarchy, with each right implying those that precede it. Thus, if a particular user is granted the updating right for a particular file, then that user is also granted the following rights: knowledge, execution, reading, and appending created a file. The owner has all of the access rights listed previously and may grant rights to others. Access can be provided to different classes of users:
9. **Specific user :** Individual users who are designated by user ID.
10. **User groups :** A set of users who are not individually defined. The system must have some way of keeping track of the membership of user groups.
11. **All :** All users who have access to this system. These are public files.

5.3.2 Simultaneous Access

- When access is granted to append or update a file to more than one user, the operating system or file management system must enforce discipline.
- A brute-force approach is to allow a user to lock the entire file when it is to be updated. A finer grain of control is to lock individual records during update.

5.4 DIRECTORY STRUCTURE

Collection of files is a file directory and it is associated with any file management system. The directory contains information about the files, including attributes, location, and ownership

5.4.1 Storage Structure

- The directory is turn a file, can be accessed by file management routines. The information about directory is provided by the system routine and this information is available to user and application.Table below shows this information it is stored in the directory for each file in the system.
- The directory stored file and sub directories and provides a mapping between file names, known to users and applications, and the files themselves. Hence each file entry includes the name of the file.
- But Virtually all systems deal with different types of files and different file organizations, and this information is also provided.
- An important category of information about each file concerns its storage, including its location and size.
- In shared systems, it is also important to provide information that is used to control access to the file.
- Typically, one user is the owner of the file and may grant certain access privileges to other users. Finally, usage information is needed to manage the current use of the file and to record the history of its usage.

Basic Information	
File Name	Name as chosen by creator (user or program). Must be unique within a specific directory.
File	Type For example: text, binary, load module, etc.
File Organization	For systems that support different organizations.
Address Information	
Volume	Indicates device on which file is stored.
Starting Address	Starting physical address on secondary storage (e.g., cylinder, track, and block number on disk).
Size Used	Current size of the file in bytes, words, or blocks.
Size Allocated	The maximum size of the file.

Module 5

Basic Information	
Access Control Information	
Owner	User who is assigned control of this file. The owner may be able to grant/deny access to other users and to change these privileges.
Access Information	A simple version of this element would include the user's name and password for each authorized user.
Permitted Actions	Controls reading, writing, executing, transmitting over a network.
Usage Information	
Date Created	When file was first placed in directory.
Identity of Creator	Usually but not necessarily the current owner.
Date Last Read Access	Date of the last time a record was read.
Identity of Last Reader	User who did the reading.
Date Last Modified	Date of the last update, insertion, or deletion.
Identity of Last Modifier	User who did the modifying.
Date of Last Backup	Date of the last time the file was backed up on another storage medium.
Current Usage	Information about current activity on the file, such as process or processes that have the file open, whether it is locked by a process, and whether the file has been updated in main memory but not yet on disk.

5.4.2 Directory Over View and Structure

UQ. 5.4.1 What is Directory? Explain Directory structures in detail? MU(IT) - Q. 3(b), May 14, 10 Marks

- Information stored is directory file is different for different system. Some of the information may be stored in a header record associated with the file; this reduces the amount of storage required for the directory, making it.
- easier to keep all or much of the directory in main memory to improve speed.
- The simplest form of structure for a directory is that of a list of entries, one for each file. This structure could be represented by a simple sequential file, with the name of the file serving as the key.
- In some earlier single-user systems, this technique has been used. However, it is inadequate when multiple users share a system and even for single users with many files.

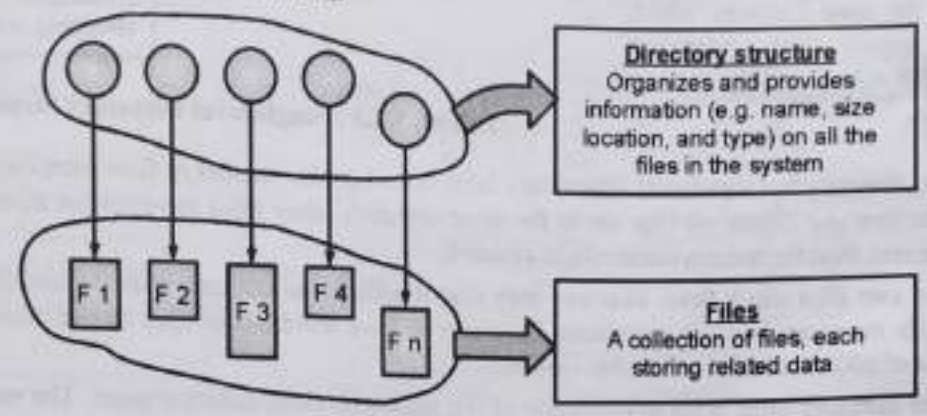
a) Operation performed on Directory

To understand the requirements for a file structure, it is helpful to consider the types of operations that may be performed on the directory :

1. **Search :** When a user or application references a file, the directory must be searched to find the entry corresponding to that file.
2. **Create file :** When a new file is created, an entry must be added to the directory.
3. **Delete file :** When a file is deleted, an entry must be removed from the directory.
4. **List directory :** All or a portion of the directory may be requested. Generally, this request is made by a user and results in a listing of all files owned by that user, plus some of the attributes of each file (e.g., type, access control information, usage information).
5. **Update directory :** Because some file attributes are stored in the directory, a change in one of these attributes requires a change in the corresponding directory entry.

b) **Logical Structure of Directory**

The file systems of computers can be extensive. Some systems store Megabytes of files on terabytes of disk. To manage all these data, we need to organize them. When there are many files searching time increases hence in order to save that time organization of data can be done on disk.



(1E9)**Fig. 5.4.1 : Directory Structure**

This organization is usually done in two parts.

1. Disks are split into one or more *partitions*, also known as *minidisks* in the *IBM* world or *volumes* in the *PC* and Macintosh. Each disk on a system contains at least one partition, which is a low-level structure in which files and directories reside.

   About Partitions and Volume :

   a. Partitions are used to provide several separate areas within one disk, each treated as a separate storage device, whereas other systems allow partitions to be larger than a disk to group disks into one logical structure.

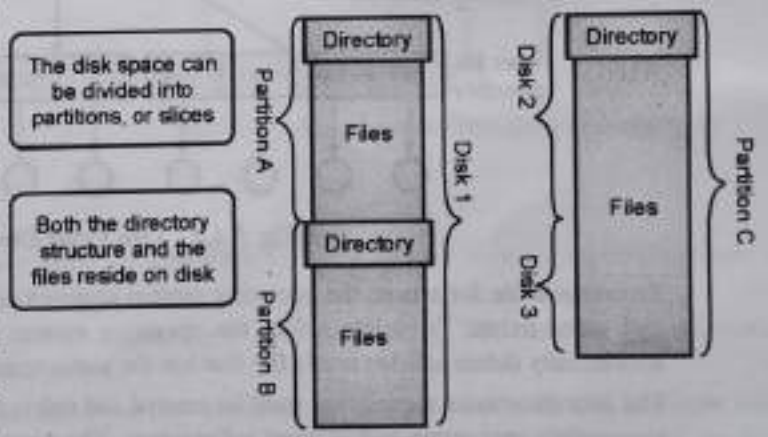
   b. Hence the user needs to be concerned with only the logical directory and file structure, and can ignore completely the problems of physically allocating space for files. Hence for this reason partitions can be thought of as virtual disks.

   Partitions can also store multiple operating systems, allowing a system to boot and run more than one.

2. Each partition contains information about files within it. This information is kept in entries in a device directory or volume table of contents.

   The device directory (more commonly known simply as a *directory)* records information-such as name, location, size, and type-for all files on that partition. Fig. 5.4.2 shows the typical file-system organization.

   The directory can be viewed as a symbol table that translates file names into their directory entries. If we take such a view, we see that the directory itself can be organized in many ways. The most common schemes for defining the logical structure of a directory.



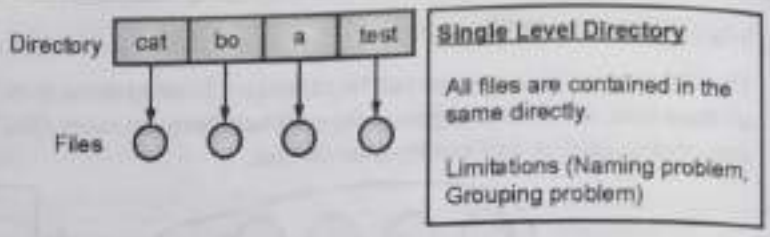
(1E10)**Fig. 5.4.2 : File system Organization**

(i) Single Level Directory	(ii) Two-Level Directory
(iii) Tree Directory Structure	(iv) Acyclic-Graph Directories

Module 5

(i) 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 shown in Fig. 5.4.3.

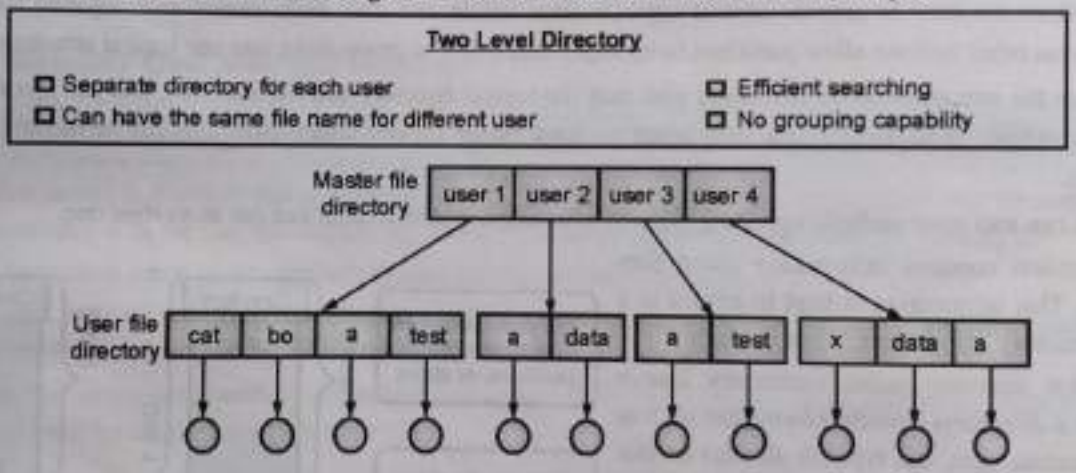


(IE11)**Fig. 5.4.3 : Single level Directory Structure**

- A single-level directory has significant limitations, however 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. If two users call their data file *test*, then the unique-name rule is violated.
- Even a single user on a single-level directory may find it difficult to remember the names of all the files, as the number of files increases. It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system.
- A single-level directory often leads to confusion of file names between different users. The standard solution is to create a *separate* directory for each user.

(ii) Two Level Directory Structure

- In the two-level directory structure, each user has her 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. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.



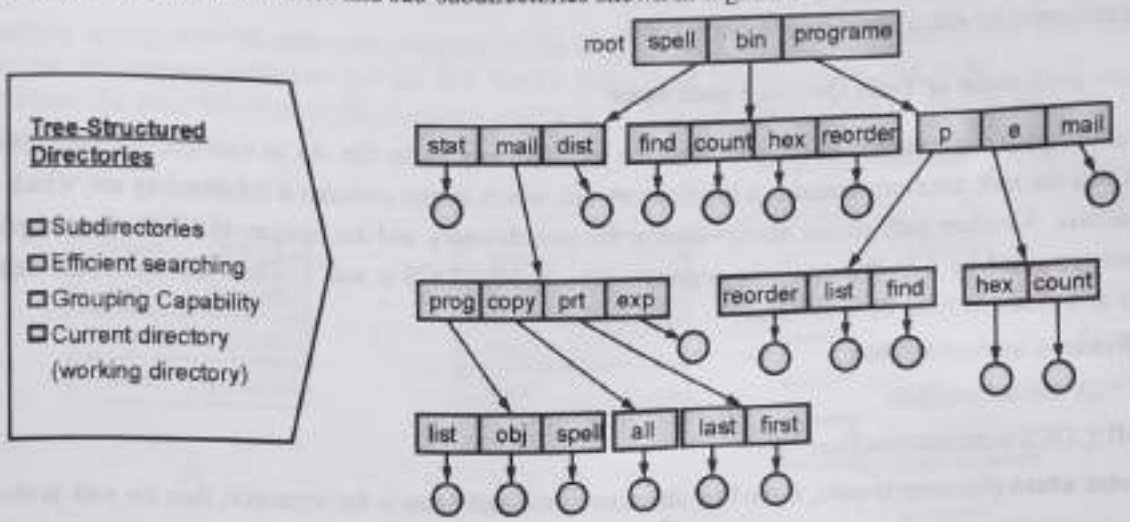
(IE12)**Fig. 5.4.4 : Two Level Directory Structure**

- 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. Fig. 5.4.4 shows Two level File Structure.
- The user directories themselves must be created and deleted as necessary. A special system program is run with the appropriate user name and account information. The program creates a new UFD and adds an entry for it to the MFD. The execution of this program might be restricted to system administrators.

(iii) Hierarchical / Tree Directory

- Tree directory structure is the most popular tree structure. It is very useful because it solves the problem of grouping, i.e. it can group different users or directories. Again the data can be efficiently searched due to path concept. Path describes where exactly that file or directory is stored.

- In the tree-structured directory, the directory themselves are files. This leads to the possibility of having sub-directories that can contain files and sub-subdirectories shown in Fig. 5.4.5.

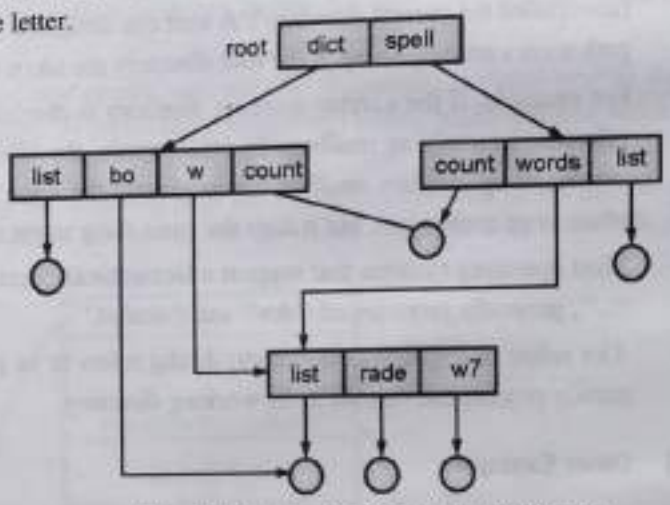


(1E13)**Fig. 5.4.5 : Tree structured Directory**

- The Microsoft Windows family of operating systems (95, 95, NT, 2000) maintains an extended two-level directory structure, with devices and partitions assigned a drive letter.

(iv) Acyclic-Graph Directories

The acyclic directory structure shown in Fig. 5.4.6 is an extension of the tree-structured directory structure. In the tree-structured directory, files and directories starting from some fixed directory are owned by one particular user. In the acyclic structure, this prohibition is taken out and thus a directory or file under directory can be owned by several users.



(1E14)**Fig. 5.4.6 : Acyclic Directory Structure**

5.4.3 Naming

- Users need to be able to refer to a file by a symbolic name. Clearly, each file in the system must have a unique name in order that file references be unambiguous.
- User is relieved from burden of providing unique names, especially in a shared system. The use of a tree-structured directory minimizes the difficulty in assigning unique names.
- Any file in the system can be located by following a path from the root or master directory down various branches until the file is reached. The series of directory names, culminating in the file name itself, constitutes a **pathname** for the file.

Module 5

a) Path Name

- A path, the general form of the name of a file or directory, specifies a unique location in a file system. A path points to a file system location by following the directory tree hierarchy expressed in a string of characters in which path components, separated by a delimiting character, represent each directory.
- The delimiting character is most commonly the slash ("/"), the backslash character ("\"), or colon (":"), though some operating systems may use a different delimiter.

- Paths are used extensively in computer science to represent the directory/file relationships common in modern operating systems, and are essential in the construction of Uniform Resource Locators (URLs). Resources can be represented by either absolute or relative paths.

1) Absolute path name or Fully Qualified path name

An absolute path name consist of the path from the root directory to the file. As an example, the path /usr/ast/mailbox means that the root directory contains a subdirectory usr, which in turn contains a subdirectory ast, which contains the file mailbox. Absolute path names always start at the root directory and are unique. In UNIX the components of the path are separated by /. In Windows the separator is \ . In MULTICS it was >. Thus, the same path name would be written as follows in these three systems :

- Windows \usr\ast\mailbox
- UNIX /usr/ast/mailbox
- MULTICS >usr>ast>mailbox

No matter which character is used, if the first character of the path name is the separator, then the path is absolute.

2) Relative path name

- The other kind of name is the relative path name. This is used in conjunction with the concept of the working directory (also called the current directory). A user can designate one directory as the current working directory, in which case all path names not beginning at the root directory are taken relative to the working directory.
- For example, if the current working directory is /usr/ast, then the file whose absolute path is /usr/ast/mailbox can be referenced simply as mailbox. In other words, the UNIX command cp /usr/ast/mailbox /usr/ast/mailbox.bak and the command cp mailbox mailbox.bakdo exactly the same thing if the working directory is /usr/ast. The relative form is often more convenient, but it does the same thing as the absolute form.
- Most operating systems that support a hierarchical directory system have two special entries in every directory, "." and "..", generally pronounced "dot" and "dotdot."
- Dot refers to the current directory; dotdot refers to its parent (except in the root directory, where it refers to itself). A certain process has /usr/ast as its working directory.

b) Other Examples

If a file name begins with only a disk designator but not the backslash after the colon(C:), it is interpreted as a relative path to the current directory on the drive with the specified letter. Note that the current directory may or may not be the root directory depending on what it was set to during the most recent "change directory" operation on that disk. Examples of this format are as follows :

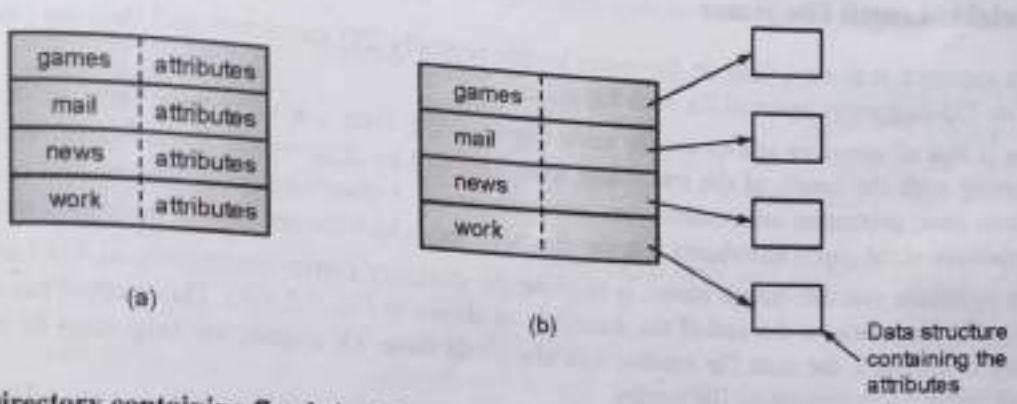
- "C:tmp.txt" refers to a file named "tmp.txt" in the current directory on drive C.
- "C:tempdir\tmp.txt" refers to a file in a subdirectory to the current directory on drive C.

A path is also said to be relative if it contains "double-dots"; that is, two periods together in one component of the path. This special specifier is used to denote the directory above the current directory, otherwise known as the "parent directory". Examples of this format are as follows :

- "..\tmp.txt" specifies a file named tmp.txt located in the parent of the current directory.
- "..\..\tmp.txt" specifies a file that is two directories above the current directory.
- "..\tempdir\tmp.txt" specifies a file named tmp.txt located in a directory named tempdir that is a peer directory to the current directory.

5.5 IMPLEMENTATION OF DIRECTORIES

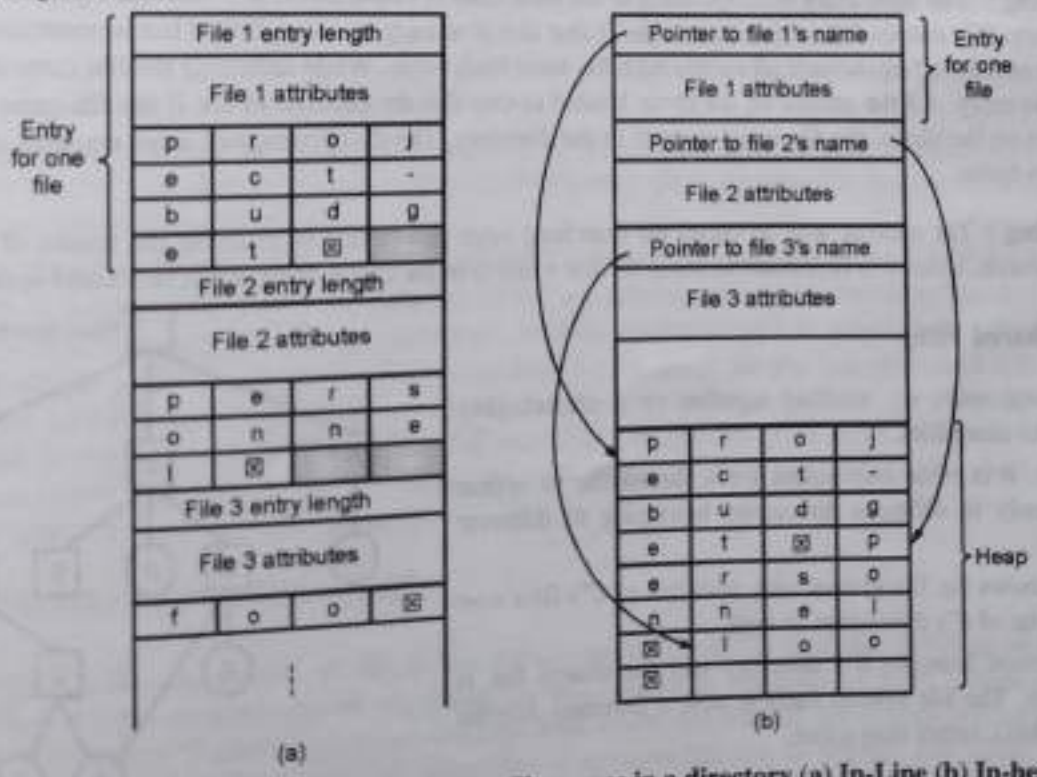
The operating system uses the path name supplied by the user to locate the directory entry on the disk. The directory entry provides the information needed to find the disk blocks. File can be located on the disk using the method discussed above. In all cases, the main function of the directory system is to map the ASCII name of the file onto the information needed to locate the data.



(a)A Simple Directory containing fixed sized Directory Entry (b) A directory in which entry just refer to an-node
(1E22)Fig. 5.5.1

The attributes of file system are generally stored with directory entry as shown in Fig. 5.5.1(a) following are different ways of implementing :

- In this simple design, a directory consists of a list of fixed-size entries, one per file, containing a (fixed-length) file name, a structure of the file attributes, One or more disk addresses (up to some maximum) telling where the disk blocks are.
- For systems that use i-nodes, another possibility for storing the attributes is in the i-nodes, rather than in the directory entries. In that case, the directory entry can be shorter: just a file name and an i-node number. This approach is illustrated in Fig. 5.5.1.(b).



(1E22)Fig. 5.5.2 : Two ways of handling long file names in a directory.(a) In-Line (b) In-heap

5.5.1 Fixed Length File Name

Generally, files have short, fixed-length names. In MS-DOS files have a 1–8 characters base name and an optional extension of 1–3 characters. In UNIX Version 7, file names were 1–14 characters, including any extensions. But all modern operating systems support longer, variable- length file names.

5.5.2 Variable Length File Name

- The simplest approach is to set a limit on file-name length, typically 255 characters, and then use one of the designs of Fig. 5.5.1 with 255 characters reserved for each file name.
- Another way is that all directory entries are the same size or fixed. Here, each directory entry contains a fixed portion, typically starting with the length of the entry, and then followed by data with a fixed format, usually including the owner, creation time, protection information, and other attributes. A disadvantage of this method is that when a file is removed, a variable-sized gap is introduced into the directory into which the next file to be entered may not fit.
- Another way to handle variable-length names is to make the directory entries themselves all fixed length and keep the file names together in a heap at the end of the directory, as shown in Fig. 5.5.1(b). This method has the advantage that when an entry is removed, the next file entered will always fit there. Of course, the heap must be managed and page faults can still occur while processing file names.

5.5.3 Searching Directories

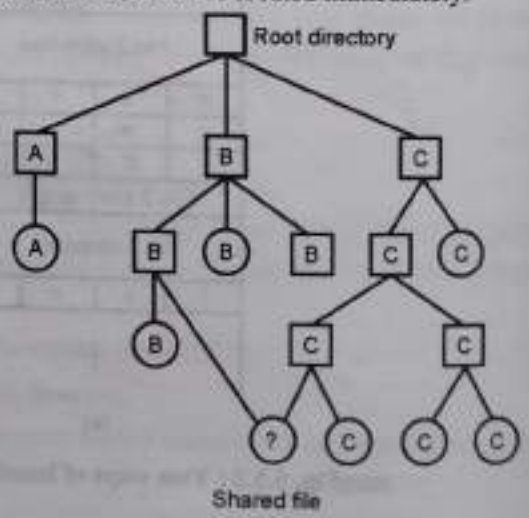
Different way of searching in directories are :

1. Linear Search	2. Hashing	3. Caching

- **1. Linear Search :** Generally, directories are searched linearly from beginning to end when a file name has to be looked up. For extremely long directories, linear searching can be slow. Hence to speed up search is to use a hash table in each directory. For the size of the table *n*. To enter a file name, the name is hashed onto a value between 0 and *n*-1,
- **2. Hashing :** The table entry corresponding to the hash code is inspected. If it is unused, a pointer is placed there to the file entry. File entries follow the hash table. If that slot is already in use, a linked list is constructed, headed at the table entry and threading through all entries with the same hash value. While searching the file name is hashed to select a hash-table entry. All the entries on the chain headed at that slot are checked to see if the file name is present. If the name is not on the chain, the file is not present in the directory. The disadvantage of more complex administration but searching is faster.
- **3. Caching :** Yet another way to speed up searching large directories is to cache the results of searches. Before starting a search, a check is first made to see if the file name is in the cache. If so, it can be located immediately.

5.5.4 Shared Files

- When several users are working together on a project, they often need to share files.
- As a result, it is often convenient for a shared file to appear simultaneously in different directories belonging to different users.
- Fig. 5.5.3 shows the file system only with one of *C*'s files now present in one of *B*'s directories as well.
- The connection between *B*'s directory and the shared file is called a link. The file system itself is now a Directed Acyclic Graph, or DAG, rather than a tree.



(1E24)**Fig. 5.5.3 : Shared Files**

a) Problem with sharing

If directories contain disk addresses, then a copy of the disk addresses will have to be made in *B*'s directory when the file is linked. If either *B* or *C* subsequently appends to the file, the new blocks will be listed only in the directory of the user doing the append. The changes will not be visible to the other user, thus defeating the purpose of sharing. This problem can be solved in two ways.

(i) Disk blocks are not listed in directories, but in a little data structure associated with the file itself. The directories would then point just to the little data structure. This is the approach used in UNIX (where the little data structure is the i-node).

(ii) *B* links to one of *C*'s files by having the system create a new file, of type LINK, and entering that file in *B*'s directory. The new file contains just the path name of the file to which it is linked. When *B* reads from the linked file, the operating system sees that the file being read from is of type LINK, looks up the name of the file, and reads that file. This approach is called **symbolic linking**, to contrast it with traditional (hard) linking.

b) Log Structured File System

In the early 90's, a group at Berkeley led by Professor John Ousterhout and graduate student Mendel Rosenblum developed a new file system known as the log-structured file system.

Their motivation to do so was based on the following observations :

- **System memories are growing :** As memory gets bigger, more data can be cached in memory. As more data is cached, disk traffic increasingly consists of writes, as reads are serviced by the cache. Thus, file system performance is largely determined by its write performance.
- **There is a large gap between random I/O performance and sequential I/O performance :** Hard-drive transfer bandwidth has increased a great deal over the years ; as more bits are packed into the surface of a drive, the bandwidth when accessing said bits increases. Seek and rotational delay costs, however, have decreased slowly; it is challenging to make cheap and small motors spin the platters faster or move the disk arm more quickly. Thus, if you are able to use disks in a sequential manner, you gain a sizeable performance advantage over approaches that cause seeks and rotations.
- **Existing file systems perform poorly on many common workloads :** For example, FFS would perform a large number of writes to create a new file of size one block: one for a new i-node, one to update the i-node bitmap, one to the directory data block that the file is in, one to the directory i-node to update it, one to the new data block that is a part of the new file, and one to the data bitmap to mark the data block as allocated. Thus, although FFS places all of these blocks within the same block group, FFS incurs many short seeks and subsequent rotational delays and thus performance falls far short of peak sequential bandwidth.
- **File systems are not RAID-aware :** For example, both RAID-4 and RAID-5 have the small-write problem where a logical write to a single block causes 4 physical I/Os to take place. Existing file systems do not try to avoid this worst-case RAID writing behavior.

An ideal file system would thus focus on write performance, and try to make use of the sequential bandwidth of the disk. Further, it would perform well on common workloads that not only write out data but also update on-disk metadata structures frequently. Finally, it would work well on RAIDs as well as single disks. The new type of file system **Rosenblum** and **Ousterhout** introduced was called LFS, short for the Log-structured File System. When writing to disk, LFS first buffers all updates (including metadata) in memory segment; when the segment is full, it is written to disk in one long, sequential transfer to an unused part of the disk. LFS never overwrites existing data, but rather always writes segments to free locations. Because segments are large, the disk is used efficiently, and performance of the file system approaches its best.

Module 5

5.6 FILE-SYSTEM IMPLEMENTATION

- The file system provides the mechanism for on-line storage and access to file contents, including data and programs. The file system resides permanently on *secondary storage*, which is designed to hold a large amount of data permanently. File System Implementation is concerned with issues surrounding file storage and access on the most common secondary-storage medium, the disk. Also ways to structure file use, to allocate disk space, to recover freed space, to track the locations of data, and to interface other parts of the operating system to secondary storage.
- Operating systems implement open () and close () systems calls for processes to request access to file contents.

5.6.1 Overview

- File system is implemented on many types of memories Like RAM (RAMDISK), Magnetic Storage (Hard disk, Tapes), Optical Storage (CDROM) and Semiconductor Storage (Pen drives). The Structure depend on operating system (Unix, Mac, Windows) and Files system Implemented.
- On disk, the file system may contain information about :
  - how to boot an operating system stored there,
  - the number and location of free blocks,
  - individual files.
  - the total number of blocks,
  - the directory structure, and
- Some of the general points related to Disk file system are :
  - A **boot control block** (per volume) can contain information needed by the system to boot an operating system from that volume. If the disk does not contain an operating system, this block can be empty. It is typically the first block of a volume. In UFS, it is called the boot block; in. NTFS, it is the partition boot sector.
  - A **volume control block** (per volume) contains volume (or partition) details, such as the number of blocks in the partition, size of the blocks, free block count and free-block pointers, and free FCB count and FCB pointers. In UFS(Unix File System), this is called a superblock; in NTFS(New Technology File System), it is stored in the master file table.
  - A **directory structure** per file system is used to organize the files. In UFS, this includes file names and associated inode numbers. In NTFS it is stored in the master file table.
  - A per-file FCB contains many details about the file, including file permissions, ownership, size, and location of the data blocks. In UFS, this is called the inode. In NTFS, this information is actually stored within the master file table, which uses a relational database structure, with a row per file.
- The in-memory information is used for both file-system management and performance improvement via caching. The data are loaded at mount time and discarded at dismount. Some point related in-memory are
  - An in-memory mount table contains information about each mounted volume.
  - An in-memory directory-structure cache holds the directory information of recently accessed directories. (For directories at which volumes are mounted, it can contain a pointer to the volume table.)
  - The system-wide open-file table contains a copy of the FCB (File Control Block) of each open file, as well as other information.
  - The per-process open-file table contains a pointer to the appropriate entry in the system-wide open-file table, as well as other information.
- An application program calls the logical file system when new file is created. The logical file system knows the format of the directory structures. To create a new file, it allocates a new FCB. (Alternatively, if the file-system implementation creates all FCBs at file-system creation time, an FCB is allocated from the set of free FCBs.)
- The system then reads the appropriate directory into memory updates it with the new file name and FCB, and writes it back to the disk. A typical FCB is shown in Fig. 5.6.1.

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Fig. 5.6.1 : File Control Block

- For UNIX, directory exactly the same as a file-one with a type field it is differentiated as directory. Windows NT and other Operating, implement separate system calls for files and directories and both files and directories are treated differently.
- The logical file system can calls the file-organization module to map the directory I/O into disk-block numbers, which are passed on to the basic file system and I/O control system. Once a file has been created, it can be used for I/O.

How File is Created ?

- File must be opened.
- The open () call passes a file name to the file system.
- The open () system call first searches the system-wide open-file table to see if the file is
- already in use by another process.
- If it is, a per-process open-file table entry is created pointing to the existing system-wide open-file table.

What happen after File is Created/Opened ?

- When a file is opened, the directory structure is searched for the given file name.
- Parts of the directory structure are usually cached in memory to speed directory operations.
- Once the file is found, the FCB is copied into a system-wide open-file table in memory. This table not only stores the FCB but also tracks the number of processes that have the file open.

Updation of File Table

- Then an entry is made in the per-process open-file table, with a pointer to the entry in the system-wide open-file table and some other fields.
- These other fields can include a pointer to the current location in the file (for the next read () or write () operation) and the access mode in which the file is open.
- The open () call returns a pointer to the appropriate entry in the per-process file-system table. All file operations are then performed via this pointer.
- The file name may not be part of the open-file table, as the system has no use for it once the appropriate FCB is located on disk.
- It could be cached, though, to save time on subsequent opens of the same file. The name given to the entry varies.
- UNIX systems refer to it as a file descriptor; Windows refers to it as a file handle. Consequently, as long as the file is not closed, all file operations are done on the open-file table.

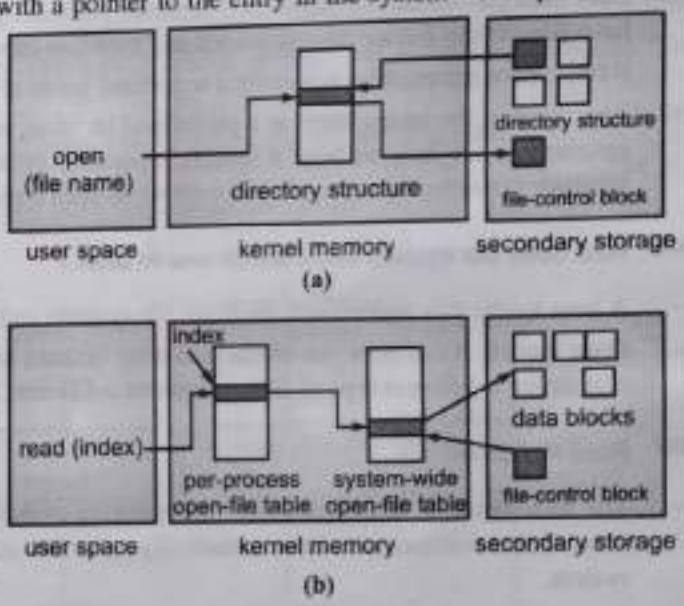


Fig. 5.6.2 : In-memory file-system structures. (a) File open. (b) File read

Module 5

What happen after File is Closed ?

- The per-process table entry is removed, and the system-wide entry's open count is decremented.
- When all users that have opened the file close it, any updated metadata is copied back to the disk-based directory structure, and the system-wide open-file table entry is removed.

In network Environment

- The file system is used as an interface to other system aspects, such as networking.

- For example, in UFS(Unix File System), the system-wide open-file table holds the i-nodes and other information for files and directories.
- It also holds similar information for network connections and devices. In this way, one mechanism can be used for multiple purposes.
- The operating structures of a file-system implementation is shown in Fig. 5.6.2(b).

5.6.2 Partitions and Mounting

- The layout, formatting and usage of a disk can have many vary, depending on the operating system.
- A disk can be divided into multiple partitions, or a volume can use multiple partitions on multiple disks.

RAW Disk

- Each partition can be either "raw," containing no file system, or "prepared," containing a file system. Raw disk is used where no file system is appropriate.
- UNIX swap space can use a raw partition, for example, as it uses its own format on disk and does not use a file system. Likewise, some databases use raw disk and format the data to suit their needs.
- Raw disk can also hold information needed by disk RAID systems, such as bit maps indicating which blocks are mirrored and which have changed and need to be mirrored. Similarly, raw disk can contain a miniature database holding RAID configuration information, such as which disks are members of each RAID set.

Boot Information

- Boot information can be stored in a separate partition. It has its own format, because at boot time the system does not have file-system device drivers loaded and therefore cannot interpret the file-system format.
- Rather, boot information is usually a sequential series of blocks, loaded as an image into memory.
- Execution of the image starts at a predefined location, such as the first byte. This boot image can contain more than the instructions for how to boot a specific operating system. For instance, PCs and other systems can be dual-booted. Multiple operating systems can be installed on such a system.

How does the system know which one to boot ?

- A boot loader that understands multiple file systems and multiple operating systems can occupy the boot space.
- Once loaded, it can boot one of the operating systems available on the disk. The disk can have multiple partitions, each containing a different type of file system and a different operating system.

Root Partition

- The root partition, which contains the operating-system kernel and sometimes other system files, is mounted at boot time. Other volumes can be automatically mounted at boot or manually mounted later, depending on the operating system.
- As part of a successful mount operation, the operating system verifies that the device contains a valid file system.
- It does so by asking the device driver to read the device directory and verifying that the directory has the expected format. If the format is invalid, the partition must have its consistency checked and possibly corrected, either with or without user intervention.

In-Memory Mount Table Structure

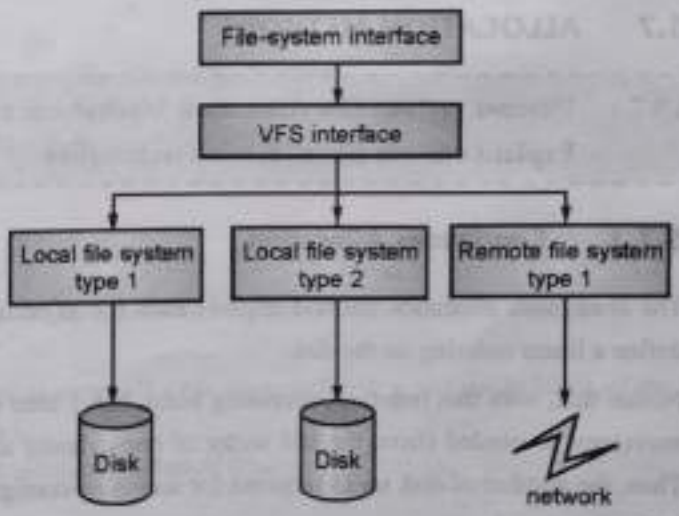
- Finally, the operating system notes in its in-memory mount table structure that a file system is mounted, along with the type of the file system. The details of this function depend on the operating system.
- **Microsoft Windows**-based systems mount each volume in a separate name space, denoted by a letter and a colon. To record that a file system is mounted at D:, for example, the operating system places a pointer to the file system in a field of the device structure corresponding to D:.

- When a process specifies the driver letter, the operating system finds the appropriate file-system pointer and traverses the directory structures on that device to find the specified file or directory. Later versions of Windows can mount a file system at any point within the existing directory structure.
- On **UNIX**, file systems can be mounted at any directory. Mounting is implemented by setting a flag in the in-memory copy of the inode for that directory. The flag indicates that the directory is a mount point.
- A field then points to an entry in the mount table, indicating which device is mounted there. The mount table entry contains a pointer to the superblock of the file system on that device.
- This scheme enables the operating system to traverse its directory structure, switching among file systems of varying types, seamlessly.

5.6.3 Virtual File System

UQ. 5.6.1 Explain Virtual File System ? **MU(IT) - Q. 5(b), Dec. 16, 10 Marks**

- Modern operating systems now a days concurrently support multiple types of file systems. The method of implementing multiple types of file systems is to write directory and file routines for each type.
- Instead, however, most operating systems, including UNIX, use object-oriented techniques to simplify, organize, and modularize the implementation.
- The use of these methods allows very dissimilar file-system types to be implemented within the same structure, including network file systems, such as NFS. Users can access files that are contained within multiple file systems on the local disk or even on file systems available across the network.
- Data structures and procedures are used to isolate the basic system call functionality from the implementation details. Thus, the file-system implementation consists of three major layers, as depicted schematically in Fig. 5.6.3.
  1. The first layer is the file-system interface, based on the open(), read(), write(), and close() calls and on file descriptors.
  2. The second layer is called the Virtual File System (VFS) layer. The VFS layer serves two important functions:
- It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally.
- It provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a v node that contains a numerical designator for a network-wide unique file. (UNIX inodes are unique within only a single file system.) This network-wide uniqueness is required for support of network file systems. The kernel maintains one v node structure for each active node (file or directory).

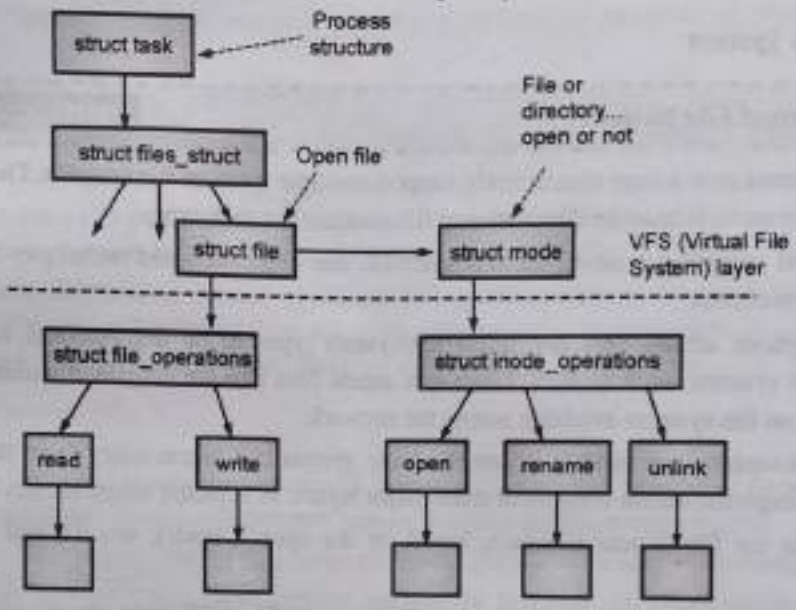


(1E25)**Fig. 5.6.3 : A Virtual File System**

- Thus, the VFS distinguishes local files from remote ones, and local files are further distinguished according to their file-system types.
- The VFS activates file-system-specific operations to handle local requests according to their file-system types and calls the NFS protocol procedures for remote requests.

Module 5

- File handles are constructed from the relevant v nodes and are passed as arguments to these procedures. The layer implementing the file-system type or the remote-file-system protocol is the third layer of the architecture.
- Let's briefly examine the VFS architecture in Linux. The four main object types defined by the Linux VFS are :
  - The inode object, which represents an individual file.
  - The file object, which represents an open file.
  - The superblock object, which represents an entire file system.
  - The entry object, which represents an individual directory entry.



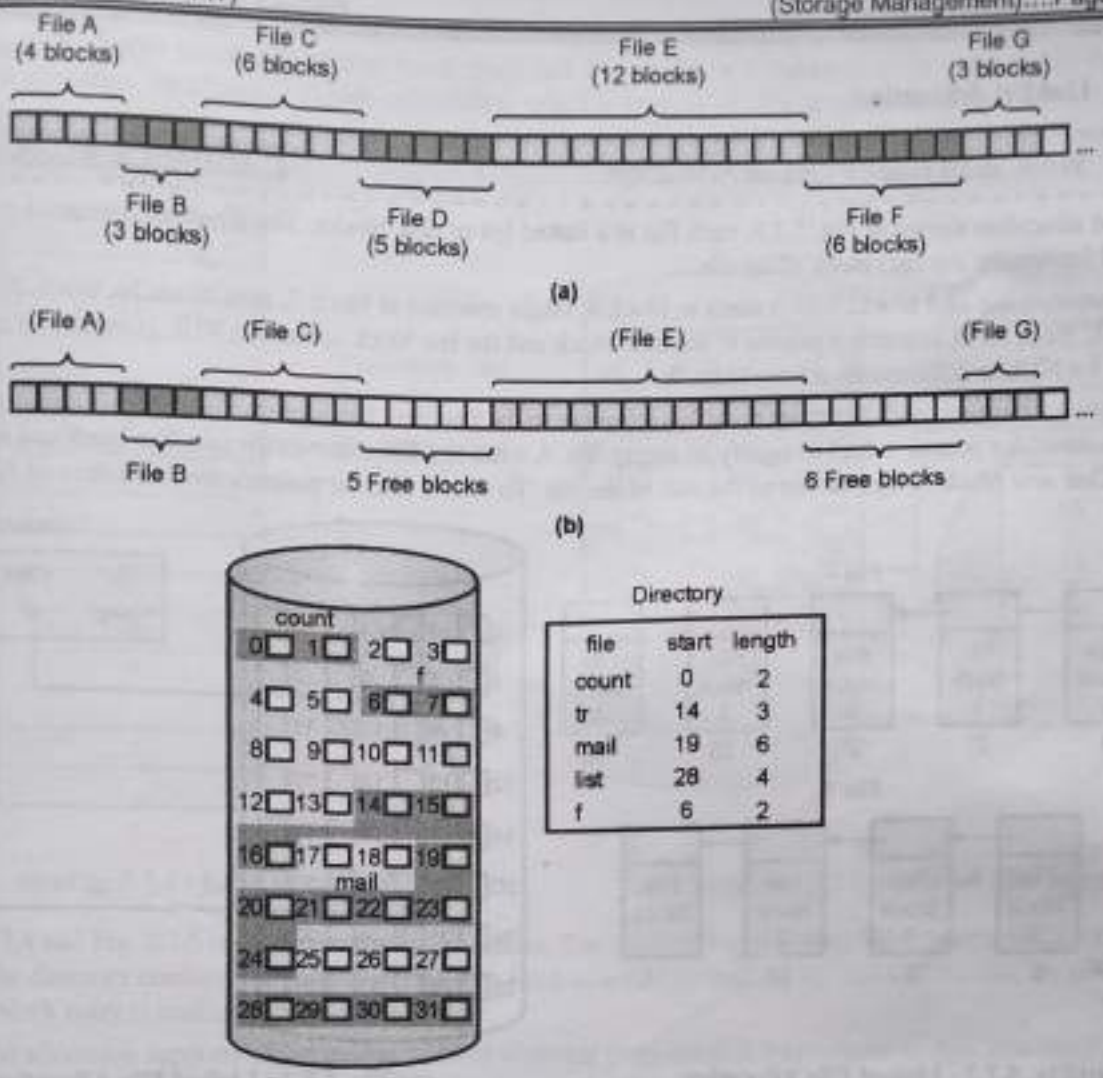
(1E26)Fig. 5.6.4 : Detail Look in Virtual File System

5.7 ALLOCATION METHODS

GQ. 5.7.1 Discuss various File Allocation Mechanism and their advantages.
OR Explain various file allocation techniques.

5.7.1 Contiguous Allocation

- The contiguous allocation method requires each file to occupy a set of contiguous address on the disk. Disk addresses define a linear ordering on the disk.
- Notice that, with this ordering, accessing block b + 1 after block b normally requires no head movement. When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), it is only one track. Thus, the number of disk seeks required for accessing contiguous allocated files in minimal, as is seek time when a seek is finally needed.
- Contiguous allocation of a file is defined by the disk address and the length of the first block. If the file is n blocks long, and starts at location b, then it occupies blocks b, b + 1, b + 2, ..., b + n – 1.
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.
- Fig. 5.7.1(a), (b) and (c) shows contiguous File allocation method.



(1E16)**Fig. 5.7.1 : Contiguous File Allocation Method**

5.7.1(a) Advantages and Disadvantages of Contiguous Allocation

Advantages

1. Supports both sequential and direct access methods.
2. Contiguous allocation is the best form u allocation for sequential files. Multiple blocks can be brought in at a time to Improve I/O performance for sequential processing.
3. It is also easy to retrieve a single block from a file. For example, if a file starts at block n and the ith block of the file is wanted, its location on secondary storage is simply n + i.
4. Quick and easy calculation of block holding data - just offset from start of file.
5. For sequential access, almost no seeks required.
6. Even direct access is fast - just seek and read only one disk access.

Disadvantages

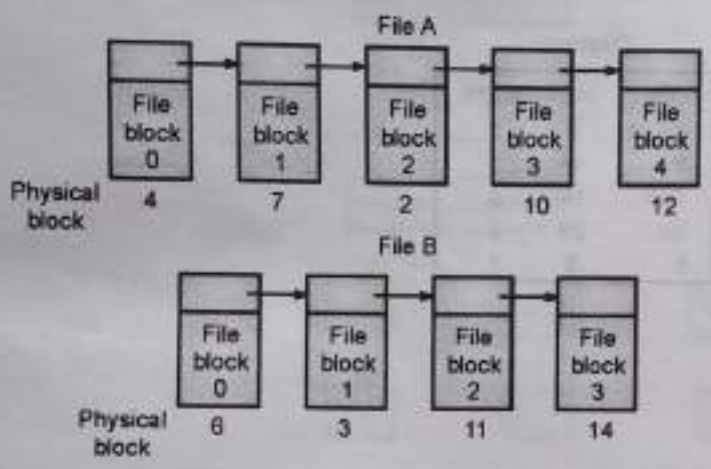
1. There is no best place to put a new file.
2. Problems when file gets bigger - may have to move whole file.
3. External Fragmentation.
4. Compaction may be required, and it can be very expensive.

Module 5

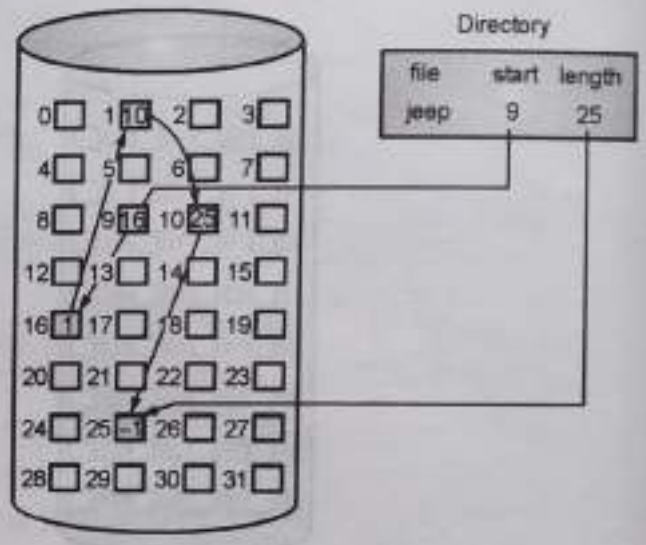
5.7.2 Linklist Allocation

UQ. 5.7.2 Write Short note on Linked Allocation. **MU(IT) - Q. 6(a), Dec. 18, 5 Marks**

- In linked allocation shown in Fig. 5.7.3, each file is a linked list of disk blocks. The directory contains a pointer to the first and (optionally the last) block of the file.
- For example, a file of 5 blocks which starts at block 4, might continue at block 7, then block 16, block 10, and finally block 27. Each block contains a pointer to the next block and the last block contains a NUL pointer. The value -1 may be used for NUL to differentiate it from block 0.
- With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file. A write to a file removes the first free block and writes to that block. This new block is then linked to the end of the file. To read a file, the pointers are just followed from block to block.



(1E17)**Fig. 5.7.2 : Linked File Allocation**



(1E18)**Fig. 5.7.3 : Linked File Allocation**

5.7.2(A) Advantages and Disadvantage of Linked Allocation

Advantages of Linked Allocation

1. Does not suffer from external fragmentation.
2. Support both sequential and direct access to the file.
3. No more variable-sized file allocation problems. Everything takes place in fixed-size chunks, which makes memory allocation a lot easier.
4. No more external fragmentation.
5. No need to compact or relocate files.

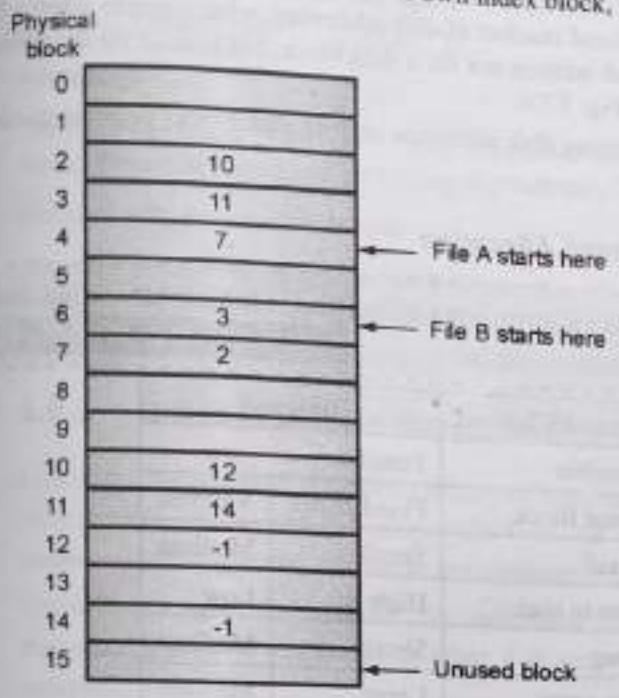
Disadvantages of linked allocation

1. Potentially terrible performance for direct access files - have to follow pointers from one disk block to the next!
2. Even sequential access is less efficient than for contiguous files because may generate long seeks between blocks.
3. Reliability if loses one pointer, have big problems.

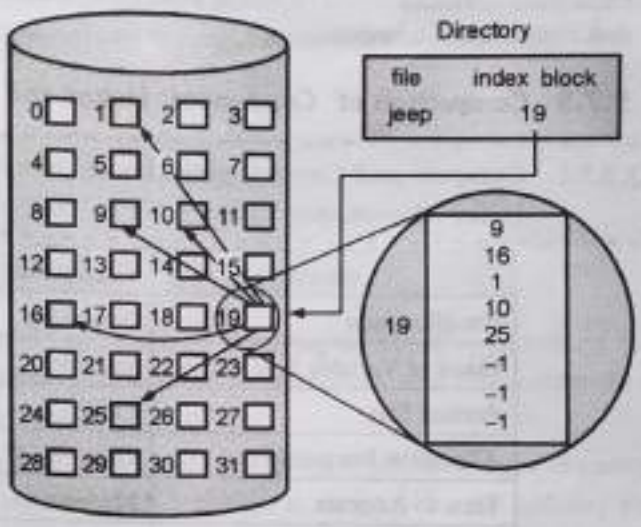
5.7.3 Indexed Allocation

- The indexed allocation method is the solution to the problem of both contiguous and linked allocation. This is done by bringing all the pointers together into one location called the index block.

- Of course, the index block will occupy some space and thus could be considered as an overhead of the method. In indexed allocation, each file has its own index block, which is an array of disk sector of addresses.



(1E19)**Fig. 5.7.4 : Indexed Allocation**

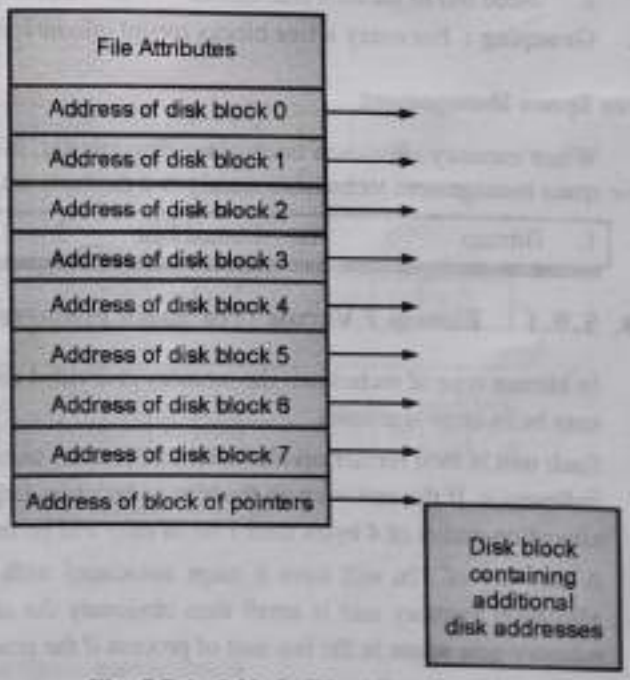


(1E20)**Fig. 5.7.5 : Indexed Allocation**

- Fig. 5.7.4 and Fig. 5.7.5 shows indexed File Allocation. The with entry in the index block points to the ith sector of the file. The directory contains the address of the index block of a file. To read the ith sector of the file, the pointer in the ith index block entry is read to find the desired sector.
- Indexed allocation supports direct access, without suffering from external fragmentation. Any free block anywhere on the disk may satisfy a request for more space.

5.7.4 I-Node Allocation

- For keeping track of which blocks belong to which file is to associate with each file a data structure called an **i-node (index-node)**, which lists the attributes and disk addresses of the file's blocks. A simple example is shown in Fig. 5.7.6. Given the i-node, it is then possible to find all the blocks of the file.
- The main advantage of this scheme over linked files using an in-memory table is that the i-node need be in memory only when the corresponding file is open. If each inode occupies n bytes and a maximum of k files may be open at once, the total memory occupied by the array holding the i-nodes for the open files is only kn bytes. But this space need be reserved in advance.
- This array occupies less memory or space then file table. The table for holding the linked list of all disk blocks is proportional in size to the disk itself. If the disk has n blocks, the table needs n entries. As disks grow larger, this table grows linearly with them.



(1E21)**Fig. 5.7.6 : I-Node File Allocation**

Module 5

- But, the i-node scheme requires an array in memory whose size is proportional to the maximum number of files that may be open at once. It does not matter if the disk is 100 GB to 10,000 GB.
- One problem with i-nodes is that if each one has room for a fixed number of disk addresses, what happens when a file grows beyond this limit? One solution is to reserve the last disk address not for a data block, but instead for the address of a block containing more disk-block addresses, as shown in Fig. 5.7.6.
- Even more advanced would be two or more such blocks containing disk addresses or even disk blocks pointing to other disk blocks full of addresses.

5.7.5 Comparison of Continuous, Linked and Indexed Allocation

UQ. 5.7.3 Compare and Contrast give allocation methods : Contiguous allocation, Linked Allocation and Index Allocation. MU(IT) Q5B May 18 10 Marks

Criteria	Contiguous	Linked/Chained	Indexed	
Pre-allocation	Necessary	Possible	Possible	
Fixed or Variable Size Portion	variable	Fixed Block	Fixed Block	Variable
Portion Size	Large	Small	Small	Medium
Allocation Frequency	Once	Low to high	High	Low
Time to Allocate	Medium	Long	Short	Medium
File Allocation Table	One entry	One entry	Large	Medium

5.8 FREE SPACE MANAGEMENT

Keep Track of Free Space following are the techniques :

1. **Keep a bit vector** (0 or 1; 1 for a free disk block)
   Simple to find first free block (bits_per_word) × (0_words) + (offset_first_1_bit)
   4 GB disk with 512 bytes block needs 1 MB space
2. **Linked list of free blocks :**
   a. Need I/O to locate *n* free blocks b. Alternatively: keep info in the FAT
3. **Grouping :** For every *n* free blocks record info in *1* about the *n-1*

Free Space Management

When memory allocation takes place dynamically, it is responsibility of an operating system to manage it properly. In free space management techniques mainly two methods are used as follows:

1. Bitmap	2. Linked List	3. Grouping	4. Counting

5.8.1 Bitmap / Vector Free Space Management

- In bitmap type of techniques the memory is divided into number of allocation units. They are also divided into words or may be in large size units.
- Each unit is then further divided into bits. Bits are zeros and ones, 0 indicates that unit is free and if it is occupied then 1 indicates it. If the unit is small the bitmap becomes large. The size of unit is very important issue in bitmap technique. If allocation unit is of 4 bytes then 1 bit of map will be required.
- A memory of 32n will have n maps associated with it. Therefore bitmap will take only 1/33 of memory. In case if allocation memory unit is small then obviously the size of bitmap will be small. In this case considerable amount of memory gets waste in the last unit of process if the process size is not equal to exact multiple of the allocation unit.

- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free, and the rest of the blocks are allocated. The free-space bit map would be:
  1100001100000011100111111000111 1...

(a) Advantage

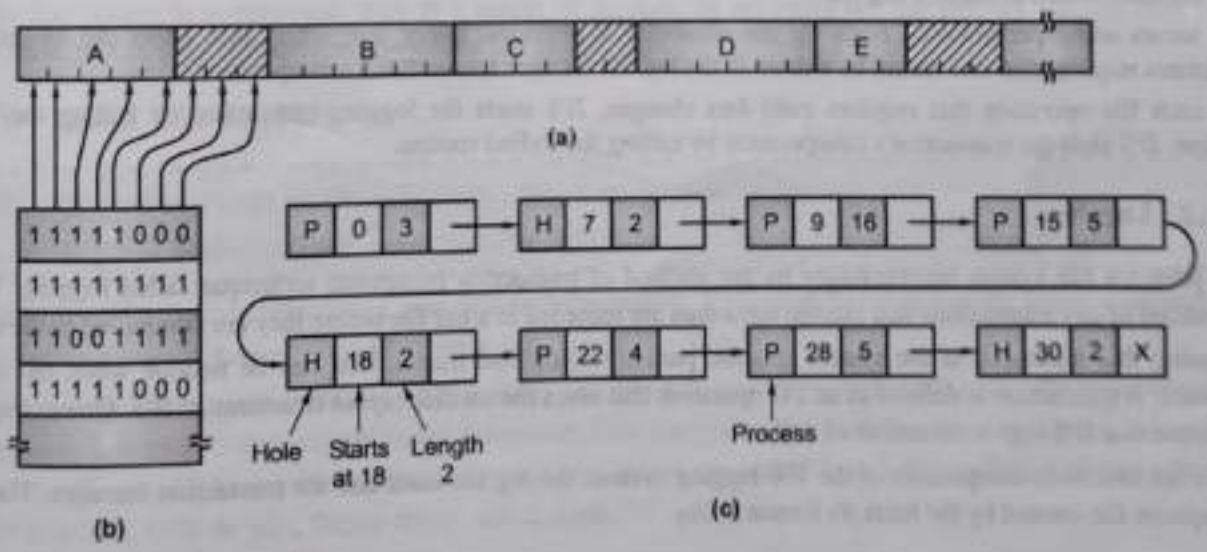
1. The main advantage of bitmap is that it provides a very simple way to keep track of memory words in a fixed amount of memory because the size of the bitmap depends only on the size of memory and the size of the allocation unit.

(b) Problem with Bitmap Technique

There is one problem associated with bitmap that when k unit processes run in the memory, the memory manager has to search a '0' bitmap which are consecutive 'n' units.
The main argument against bitmap is that the searching a bitmap for a run of a given length is a slow operation.

5.8.2 Linked Free Space Management

- Link list is collection of allocated memory or hole between two processes. By observing the figure we understand that P indicates the process and H indicates the hole. All the entries present in the list specifies a hole (H) process (P) and address at which it starts, the length and a pointer to the next entry.
- The example shown in Fig. 5.8.1, we understand that the list is kept sorted. One of the major advantages of sorting is that when a process terminates or when a process terminates or when a process is swapped out the updating list is straightforward.
- A terminating memory has two neighbors only. In exceptional cases are top of the memory it do not have neighbor at the left side and the bottom of the memory there is no neighbor at the right side. The neighbors could be holes or process. There could be for combinations as shown in the Fig. 5.8.1(a) shows modifying the list requires replacing a P by H.
- Fig. 5.8.1(b) and (c) also two entries and combined into one and the list becomes one entry shorter. Doubly link list is useful to find out most recent entry and to see if merge is possible.
- When processes or even holes are kept sorted in the list, then different algorithms could be used to allocate the memory. These algorithm are called as
  1. First Fit 2. Best Fit 3. Next Fit 4. Worst Fit



(1D4e)**Fig. 5.8.1 : Linked Free Space management**

5.8.3 Grouping Free Space Management

A modification of the free-list approach is to store the addresses of n free blocks in the first free block. The first n-1 of these is actually free. The last one is the disk address of another block containing addresses of another n free blocks. The importance of this implementation is that addresses of a large number of free blocks can be found quickly.

5.8.4 Counting Free Space Management

Another approach is to take advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when contiguous allocation is used. Thus, rather than keeping a list of free disk addresses, the address of the first free block is kept and the number n of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

5.9 JOURNAL FILE SYSTEM

- The Journaled File System (JFS) provides a log based, byte-level file system that was developed for transaction-oriented, high performance systems. Scalable and robust, one of the key features of the file system is logging.
- JFS, a recoverable file system, ensures that if the system fails during power outage, or system crash, no file system transactions will be left in an inconsistent state. The on-disk layout of JFS will remain consistent without the need to run fsck.
- JFS provides the extra level of stability with a small decrease in performance when meta-data changes need to be logged. JFS uses a transaction-based logging technique to implement recoverability. This design ensures a full partition recovery within seconds for even large partition sizes. JFS limits its recovery process to the file system structures to ensure that at the very least the user will never lose a partition because of a corrupted file system.
- Note, that user data is not guaranteed to be fully updated if a system crash has occurred. JFS is not designed to log user data and therefore is able to keep all file operations to an optimal level of performance. The development of a recoverable file system can be viewed as the next step in file system technology

5.9.1 Recoverable File Systems

- A recoverable file system, such as IBM's Journaled File System-(JFS), ensures partition consistency by using database-logging techniques originally developed for transaction processing.
- If the operating system crashes, JFS restores consistency by executing the log redo procedure that accesses information that has been stored in the JFS log file.
- JFS incurs some performance costs for the reliability it provides. Every transaction that alters the on-disk layout structures requires that one record be written to the log file for each transaction's sub-operations.
- For each file operation that requires meta-data changes, JFS starts the logging transaction by calling the TxBegin routine. JFS ends the transaction's suboperation by calling the TxEnd routine.

5.9.2 Logging

- JFS provides file system recoverability by the method of transaction processing technique called logging. The sub-operations of any transactions that change meta-data are recorded in a log file before they are committed to the disk.
- By using this technique, if the system crashes, partially completed transactions can be undone when the system is rebooted. A transaction is defined as an I/O operation that alters the on-disk layout structures of JFS. One example of an operation that JFS logs is the unlink of a file.
- There are two main components of the JFS logging system: the log file itself and the transaction manager. The log file is a system file created by the mkfs.jfs format utility.

5.9.3 Extents, I-nodes, Block Map

- A "file" is allocated in sequences of extents. An Extent is a sequence of contiguous aggregate blocks allocated to a JFS object as a unit.
- An extent is wholly contained within a single aggregate (and therefore a single partition); however, large extents may span multiple allocation groups. An extent can range in size from 1 to 2(24) -1 aggregate blocks. Every JFS object is represented by an inode. Inodes contain the expected object-specific information such as time stamps and file type (regular vs. directory, etc.).
- The Block Allocation Map is used to track the allocated or freed disk blocks for an entire aggregate. The Inode Allocation Map is a dynamic array of Inode Allocations Groups (IAGS). The IAG is the data for the Inode Allocation Map.

5.9.4 Transaction Manager

The Transaction Manager provides the core functionality that JFS uses to do logging. A brief explanation of the transaction flow follows :

- A call to TxBegin allocates a transaction "block", tblk, which represents the entire transaction.
- When meta-data pages are created, modified, or deleted, transaction "locks", tlck's, are allocated and attached to the tblk.
- There is a 1 to 1 correspondence between a tlck and a meta-data buffer (excepting that extra tlcks may be allocated if the original overflows).
- The buffers are marked 'nohomeok' to indicate that they shouldn't be written to disk yet. In txCommit (), the tlck's are processed and log records are written (at least to the buffer). The affected inodes are "written" to the inode extent buffer (they are maintained in separate memory from the extent buffer). Then a commit record is written. After the commit record has actually been written (I/O complete), the block map and inode map are updated as needed, and the tlck'ed metadata pages are marked 'homeok'. Then the tlcks are released.
- Finally the tblk is released.

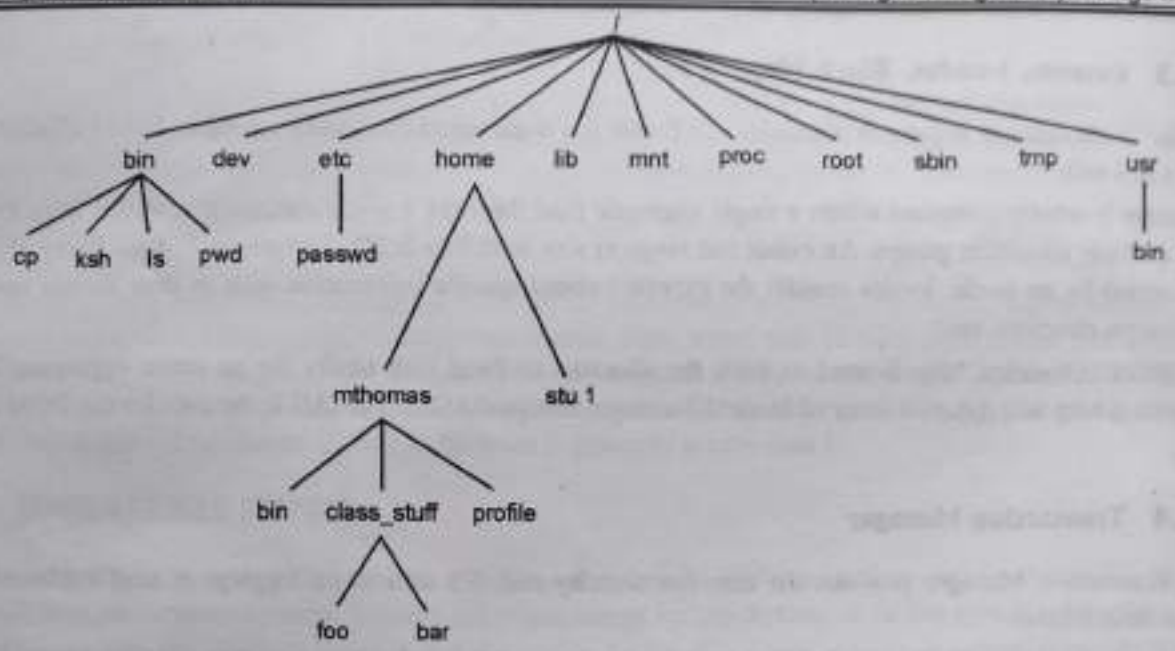
When recovering after a system failure, JFS reads through the log file and if needed redoes each committed transaction. After redoing the committed transactions during a file system recovery, JFS locates all the transactions in the log file that were not committed at failure time and rolls back (undoes) each sub-operation that had been logged. By logging JFS can restore the file system to a consistent state in a matter of seconds, by replaying the log file. By logging only meta-data changes JFS is able to keep the performance of this file system high.

5.10 UNIX FILE SYSTEM

GQ. 5.10.1 Write short note on the following : File management in Linux.
OR Explain in detail file management in UNIX.

Module 5

- A file system is a logical collection of files on a partition or disk. A partition is a container for information and can span an entire hard drive if desired. Unix file system is shown in Fig. 5.10.1.
- Hard drive can have various partitions which usually contains only one file system, such as one file system housing the / file system or another containing the /home file system. One file system per partition allows for the logical maintenance and management of differing file systems. Everything in Unix is considered to be a file, including physical devices such as DVD-ROMs, USB devices, floppy drives, and so forth.



(MU)Fig. 5.10.1 : Unix File System

(a) Directory Structure

Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.

A UNIX file system is a collection of files and directories that has the following properties:

- It has a root directory (/) that contains other files and directories.
- Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an inode.
- By convention, the root directory has an inode number of 2 and the lost+found directory has an inode number of 3. Inode numbers 0 and 1 are not used. File inode numbers can be seen by specifying the -i option to ls command.
- It is self-contained. There are no dependencies between one file system and any other.

The directories have specific purposes and generally hold the same types of information for easily locating files.

Following are the directories that exist on the major versions of Unix :

Directory	Description
/	This is the root directory which should contain only the directories needed at the top level of the file structure.
/bin	This is where the executable files are located. They are available to all user.
/dev	These are device drivers.
/etc	Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, ethernet, hosts, where to send critical messages.
/lib	Contains shared library files and sometimes other kernel-related files.
/boot	Contains files for booting the system.
/home	Contains the home directory for users and other accounts.
/mnt	Used to mount other temporary file systems, such as cdrom and floppy for the CD-ROM drive and floppy diskette drive, respectively

Directory	Description
/proc	Contains all processes marked as a file by process number or other information that is dynamic to the system.
/tmp	Holds temporary files used between system boots
/usr	Used for miscellaneous purposes, or can be used by many users. Includes administrative commands, shared files, library files, and others
/var	Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
/sbin	Contains binary (executable) files, usually for system administration. For example, fdisk and ifconfig utilities.
/kernel	Contains kernel files

(b) File Types

The UNIX file system contains several different types of files :

(i) Ordinary Files

- Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
- Always located within/under a directory file.
- Do not contain other files.

(ii) Directories

- Branching points in the hierarchical tree.
- Used to organize groups of files.
- May contain ordinary files, special files or other directories.
- Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
- All files are descendants of the root directory, (named /) located at the top of the tree.

(iii) Special Files

- Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations.
- Unix considers any device attached to the system to be a file - including your terminal :
- By default, a command treats your terminal as the standard input file (stdin) from which to read its input.
- Your terminal is also treated as the standard output file (stdout) to which a command's output is sent.
- Stdin and stdout will be discussed in more detail later.
- Two types of I/O : character and block.
- Usually only found under directories named /dev.

Module 5

(c) Pipes

- UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another.
- For example, to pipe the output from one command into another command :

  who | wc -l
- This command will tell you how many users are currently logged into the system. The standard output from the who command is a list of all the users currently logged into the system. This output is piped into the wc command as its standard input. Used with the -l option this command counts the numbers of lines in the standard input and displays the result on its standard output - your terminal.

5.11 OVER VIEW OF MASS STORAGE

UQ. 5.11.1 Write short note on Mass storage. MU(IT) - Q. 6(f), May 16, 5 Marks

- Hollerith cards or punch cards are mass storage device used with main frame computer and they are relatively manual storage media. Mass storage today include various types of hard disks or solid-state storage media, as well as tape containers and other types of physical data storage media.
- The idea of mass storage is also often associated with data backup and data recovery. Tape or other media can be used to provide ongoing mass storage handling. Other types of mass storage may act as an effective data storage strategy for an entire network or a set of portable remote devices.
- For example, mass storage for a tablet use of flash or USB media to back up data on a portable tablet that lacks a high amount of internal memory. Permanent peripheral storage, which is typically a magnetic tape, hard disk drive or solid state drive (SSD).
- If the term mass storage today, it means a huge amount of storage such as in a server array and not the capacity found in a single computer or server like RAID Array, Flash Memory, Magnetic Disk, Optical disk and Magnetic Tape.

5.11.1 Magnetic Storage

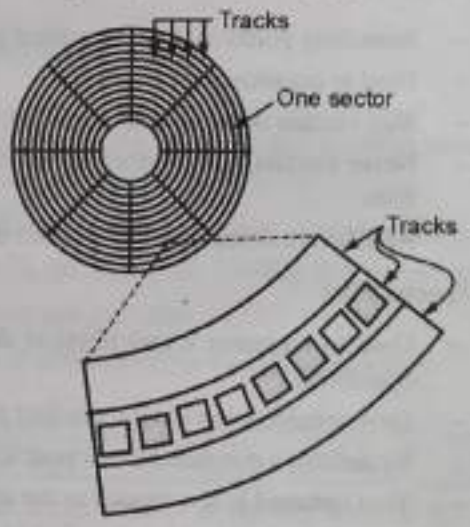
- Computer systems need to store data in digital format. One of the most widely used types of digital data storage is **magnetic storage**. This refers to any type of data storage using a magnetized medium. Digital data consists of **binary information**, which is data in the form of zero and ones. There are two types of magnetic polarities, each one used to represent either zero or one.
- Several types of magnetized media are used in computer systems, including **magnetic tape**, **floppy disks** and **hard disk drives**.

Tracks and Spots

The disk surface is divided into concentric tracks (circles within circles). The data bits are recorded as tiny magnetic spots on the tracks. The smaller the spot, the more bits per inch and the greater the storage

Sectors

Tracks are further divided into sectors, which hold a block of data that is read or written at one time; In order to update the disk, one or more sectors are read into the computer, changed and written back to disk. The operating system figures out how to fit data into these fixed spaces.



(1F15)**Fig. 5.11.1 : Tracks and Sector in Hard Disk**

Tracks and Sectors

- Tracks are concentric circles on the disk, broken up into storage units called "sectors." The sector, which is typically 512 bytes, is the smallest unit that can be read or written, shown Fig. 5.11.1. The time taken to access a particular block consists of :
  - Time to move the heads to the right track - the seek time.
  - Time waiting for the sector to come round to the head - the latency.
  - Time to actually transfer the data - the block transfer time.
- All these times are of the order of tens of milliseconds. The seek time and latency are obviously variable, depending on where the required block is relative to the current position of the heads. It's useful to put all the blocks of a file on one track, to minimise seek times and latencies in accessing the file.

Data Transfer on disk Technology

- Older disks have little electronics and just deliver a simple serial bit stream. On these disks, the controller does most of the work.
- On other disks, **IDE (Integrated Drive Electronics)** and **SATA (Serial ATA)** disks, the disk drive itself contains a microcontroller that does considerable work and allows the real controller to issue a set of higher-level commands. The controller often does track caching, bad-block remapping, and much more.
- A device feature that has important implications for the disk driver is the possibility of a controller doing seeks on two or more drives at the same time. These are known as **overlapped seeks**.
- While the controller and software are waiting for a seek to complete on one drive, the controller can initiate a seek on another drive.
- Many controllers can also read or write on one drive while seeking on one or more other drives for hard disks with integrated controllers, and in a system with more than one of these hard drives they can operate simultaneously, at least to the extent of transferring between the disk and the controller's buffer memory. Only one transfer between the controller and the main memory is possible at once, however. The ability to perform two or more operations at the same time can reduce the average access time considerably.

5.11.2 Magnetic Tape

- Magnetic tape was used as an early secondary-storage medium. It can hold permanent and large quantities of data, but its access time is slow compared with that of main memory and magnetic disk.
- Also random access to magnetic tape is about a thousand times slower than random access to magnetic disk, so tapes are not very useful for secondary storage.
- Tapes are used generally used for backup, for storage of infrequently used information, and as a medium for transferring information from one system to another. A tape is kept in a spool and is wound or rewound past a read - write head.
- Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives.
- Tape capacities vary greatly, depending on the particular kind of tape drive. Typically, they
- store from 20 GB to 200GB. Some have built-in compression that can more than double the effective storage. Tapes and their drivers are usually categorized by width, including 4,8 and 19 millimeters and 1/4 and 1/2 inch.

(a) IBM 729 (b) IBM 7330

Fig. 5.11.2 : Magnetic Tape Unit

Fig. 5.11.3 : Magnetic Tape (Spool)

Module 5

Advantages and Disadvantages of Magnetic Tapes

Advantages

1. These are inexpensive, i.e., low cost memories.
2. It provides backup or archival storage.
3. It can be used for large files.
4. It can be used for copying from disk files.
5. It is a reusable memory.
6. It is compact and easy to store on racks.

Disadvantages

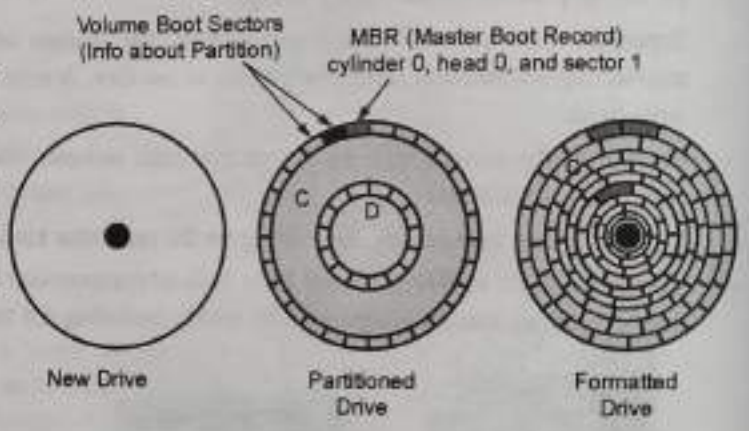
1. Sequential access is the disadvantage, means it does not allow access randomly or directly.
2. It requires caring to store, i.e., vulnerable humidity, dust free, and suitable environment.
3. It stored data cannot be easily updated or modified, i.e., difficult to make updates on data.

5.12 DISK STRUCTURE

- Disk formatting can be performed on both magnetic platter hard drives and solid-state drives. The formatting comprises low-level formatting, partitioning and high-level formatting. Low-level formatting aids in preparing the physical structure on the storage media. The partitioning process involves the division of the hard drive into logical volumes for data storage. High-level formatting helps in creating the file system format within the logical volume or within the disk partition. Disk formatting is usually done with the help of a disk formatting utility.
- Formatted and Unformatted disk are shown in Fig. 5.12.1.

(a) Low-level Disk Formatting

- Low-level formatting is the process of marking out cylinders and tracks for a blank hard disk, and then dividing tracks into multiple sectors.
- If users perform low-level formatting when data have been installed, all existing files will be erased, and it is almost impossible to recover them. Therefore, some users make such a format to avoid privacy leakage.



(1F26)**Fig. 5.12.1 : Formatting of Disk**

- To low level format a hard disk, users can make use of specific tools as well as Debug assembler languages. That's relatively complicated.

Preamble	Data	ECC

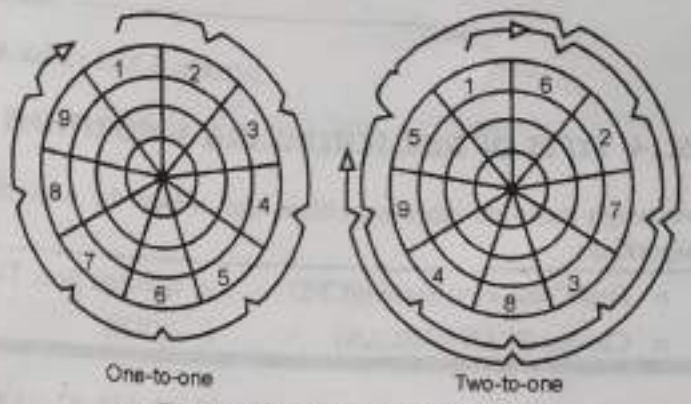
(1F27)**Fig. 5.12.2 : A Disk Sector**

(b) High-level Disk Formatting

- This is the process of writing a file system, cluster size, partition label, and so on for a newly created partition or volume. And we can also say high-level formatting just clears data on hard disk, generates boot information, initializes FAT, and labels logical bad sectors when the partition has existed. This process does no harm to hard disk in general situations, so we suggest taking such a format to fix a logically damaged partition or device, for example, Windows asks to format a SD card.

Interleave

- To arrange data in a noncontiguous way to increase performance. When used to describe disk drives, it refers to the way sectors on a disk are organized. In one-to-one interleaving, the sectors are placed sequentially around each track. In two-to-one interleaving, sectors are staggered so that consecutively numbered sectors are separated by an intervening sector shown in Fig. 5.12.3.
- The purpose of interleaving is to make the disk drive more efficient. The disk drive can access only one sector at a time, and the disk is constantly spinning beneath the read/write head. This means that by the time the drive is ready to access the next sector, the disk may have already spun beyond it. If a data file spans more than one sector and if the sectors are arranged sequentially, the drive will need to wait a full rotation to access the next chunk of the file. If instead the sectors are staggered, the disk will be perfectly positioned to access sequential sectors.



(1F28)**Fig. 5.12.3 : Sector Interleaving**

5.13 DISK SCHEDULING

GQ. 5.13.1 Discuss various disk scheduling methods.
OR Explain techniques of disk scheduling.

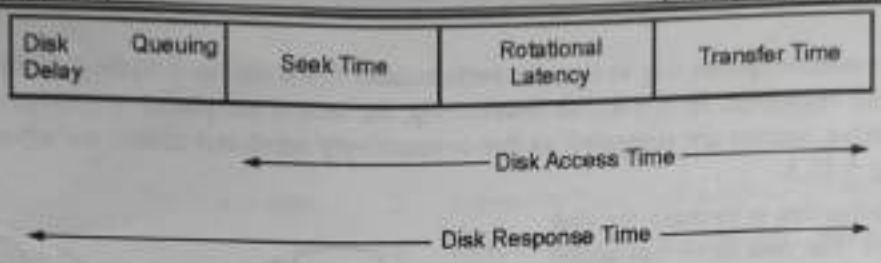
Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

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. For most disks, the seek time dominates the other two times, so reducing the mean seek time can improve system performance substantially.

Module 5



(1F33)Fig. 5.13.1 : Disk Access Time

5.14 TYPES OF DISK SCHEDULING ALGORITHMS

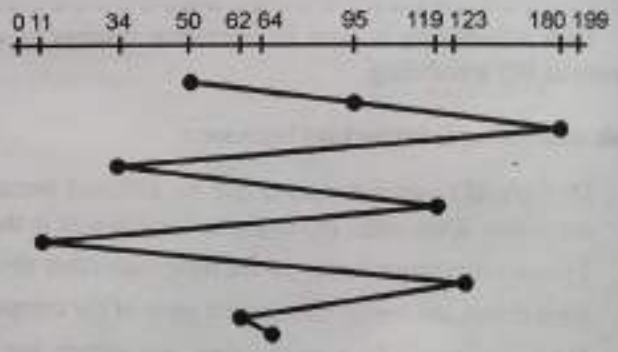
- Although there are other algorithms that reduce the seek time of all requests, following disk are the scheduling algorithms :

o First Come-First Serve (FCFS)	o Shortest Seek Time First (SSTF)	o Elevator (SCAN)
o Circular SCAN (C-SCAN)	o LOOK	o C-LOOK

- Given the following queue - 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

5.14.1 First Come -First Serve (FCFS)

- All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served.
- To determine the number of head movements simply find the number of tracks it took to move from one request to the next. For this case it moved from 50 to 95 to 180 and so on.
- From 50 to 95 it moved 45 tracks. Add up all the track to get final movement. In this example, it had a total head movement of 640 tracks.
- The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. This is the worse algorithm that one can use. FCFS is shown in Fig. 5.14.1.



(1F34)Fig. 5.14.1 : FCFS Disk Arm Scheduling

5.14.1(A) Advantages and Disadvantages

Advantages

1. Every request gets a fair chance.
2. No indefinite postponement.

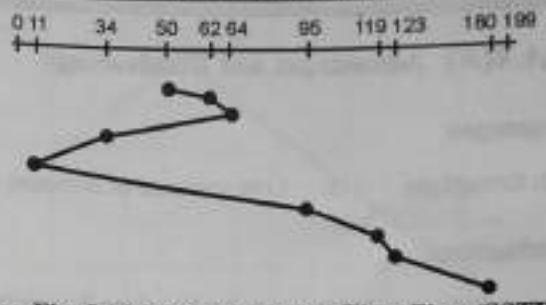
Disadvantages

1. Does not try to optimize seek time.
2. May not provide the best possible service.

5.14.2 Shortest Seek Time First (SSTF)

- In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way.

- Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place if there ate lot close request which will never handle greater distance.
- SSTF is shown in Fig. 5.14.2.



(1F35)Fig. 5.14.2 : Shortest Seek Time First (SSTF)

5.14.2(A) Advantages and Disadvantages

Advantages

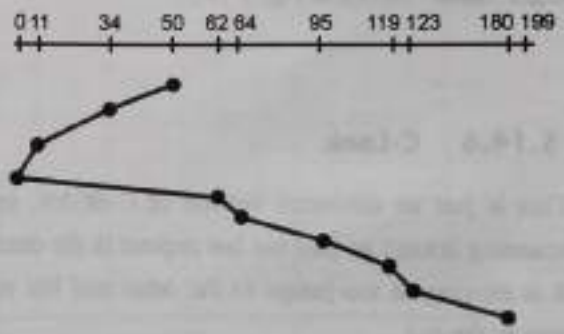
1. Average Response Time decreases
2. Throughput increases

Disadvantages

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

5.14.3 Elevator (Scan)

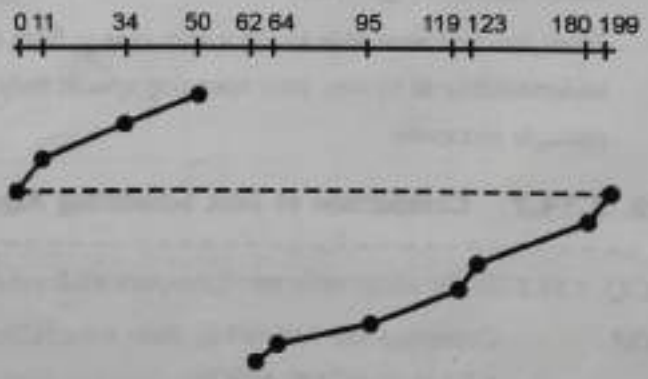
- This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down.
- If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best shown in Fig. 5.14.3.



(1F36)Fig. 5.14.3 : Elevator scan

5.14.4 Circular Scan (C-SCAN)

- Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction.
- Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the more sufficient shown in Fig. 5.14.4.



(1F37)Fig. 5.14.4 : Circular Scan

Module 5

5.14.4(A) Advantages and Disadvantage

Advantages

1. High throughput
2. Low variance of response time
3. Average response time

Disadvantage

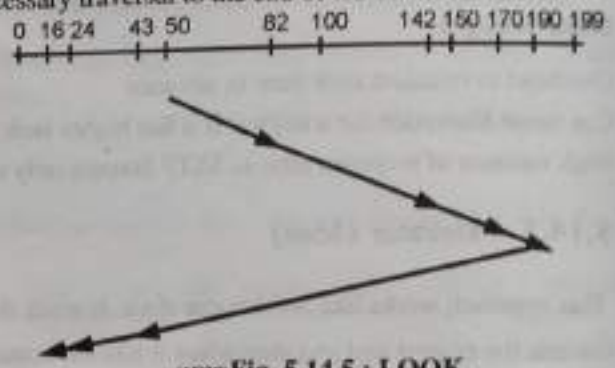
1. Long waiting time for requests for locations just visited by disk arm.

5.14.5 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

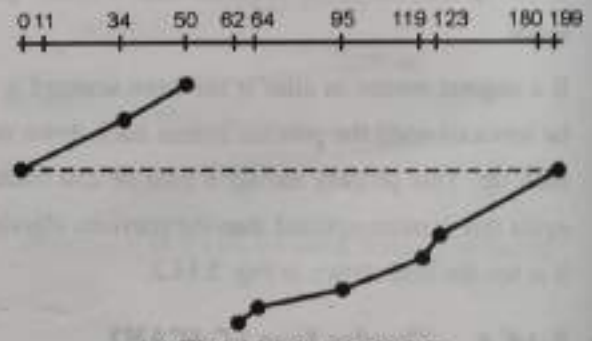
Suppose the requests to be addressed are- 82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value"** shown in Fig. 5.14.5.



(1F38)**Fig. 5.14.5 : LOOK**

5.14.6 C-Look

- This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end.
- Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks shown in Fig. 5.14.6.



(1F39)**Fig. 5.14.6 : C-LOOK**

- From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.

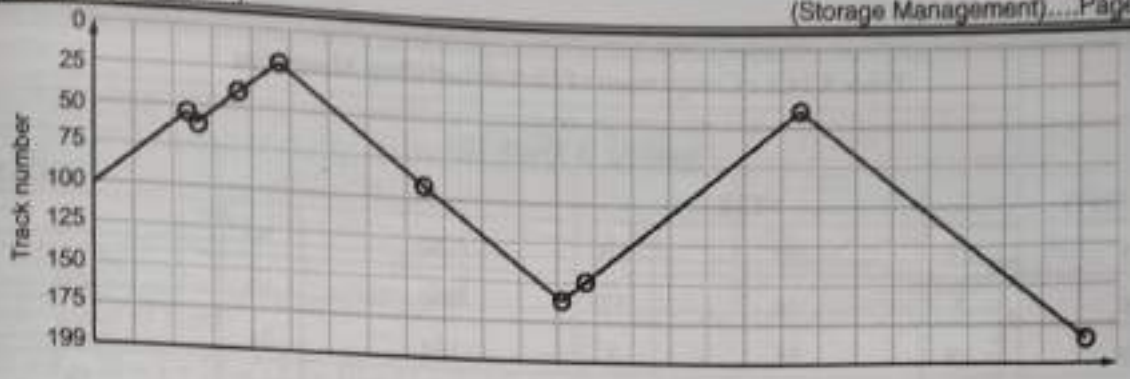
5.14.7 Comparison of Disk Scheduling Algorithm

GQ. 5.14.1 Write short note on : Compare disk scheduling algorithm.

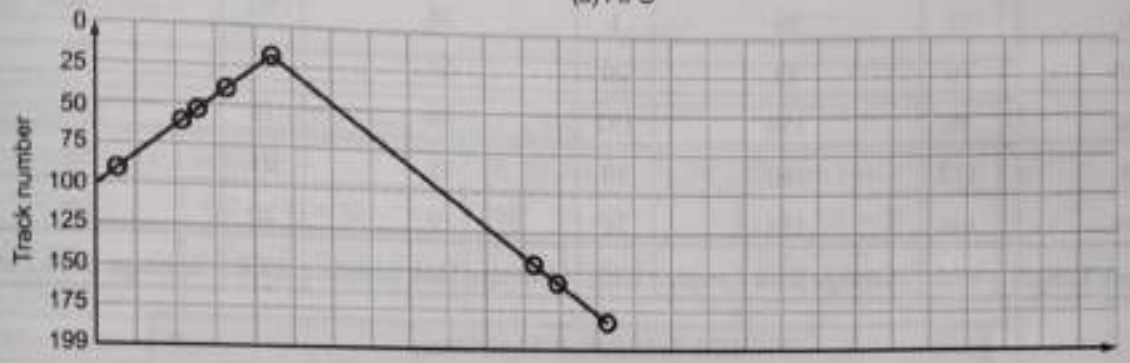
OR Compare the following disk scheduling algorithms using appropriate example – SSTF, FCFS, SCAN, C-SCAN, LOOK.

The requested tracks, in the order received by the disk scheduler, are 55, 58, 39, 18, 90, 160, 150, 38, 184.

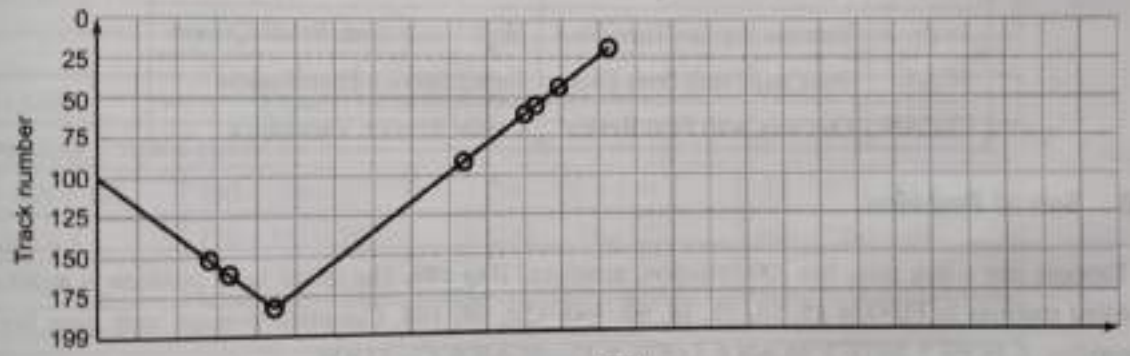
Apply the following disk scheduling algorithms, starting track is 100.



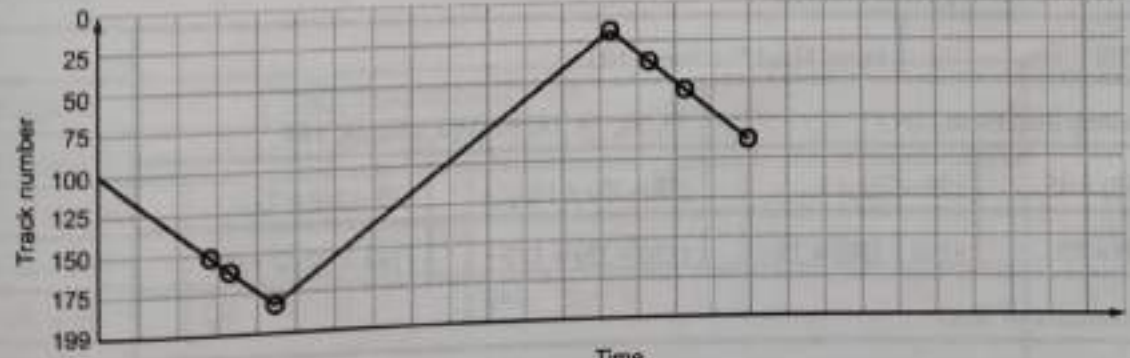
(a) FIFO



(b) SSTF



(c) SCAN



(d) C-SCAN

(1F40)**Fig. 5.14.7 : Comparison of Disk Scheduling Algorithm**

Module 5

⏭ 5.15 RAID STRUCTURE

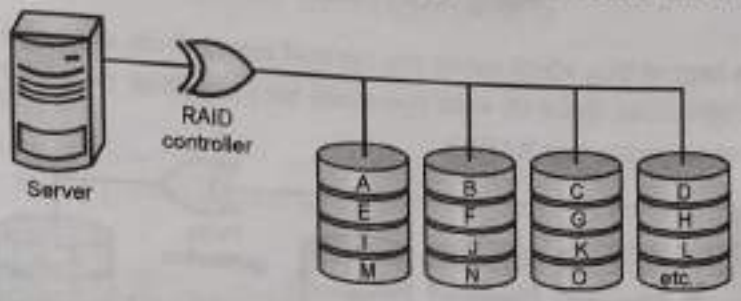
UQ. 5.15.1 What is RAID ? What are different RAID Level ? **MU(IT) - Q. 4(a), May 18, 10 Marks**

- RAID (*redundant array of independent disks*; originally *redundant array of inexpensive disks*) provides a way of storing the same data in different places (thus, redundantly) on multiple hard disks (though not all RAID levels provide redundancy). By placing data on multiple disks, input/output (I/O) operations can overlap in a balanced way, improving performance.

- Since multiple disks increase the Mean Time Between Failure (MTBF), storing data redundantly also increases fault tolerance.
- RAID arrays appear to the operating system (OS) as a single logical hard disk. RAID employs the technique of disk mirroring or disk striping, which involves partitioning each drive's storage space into units ranging from a sector (512 bytes) up to several megabytes. The stripes of all the disks are interleaved and addressed in order.

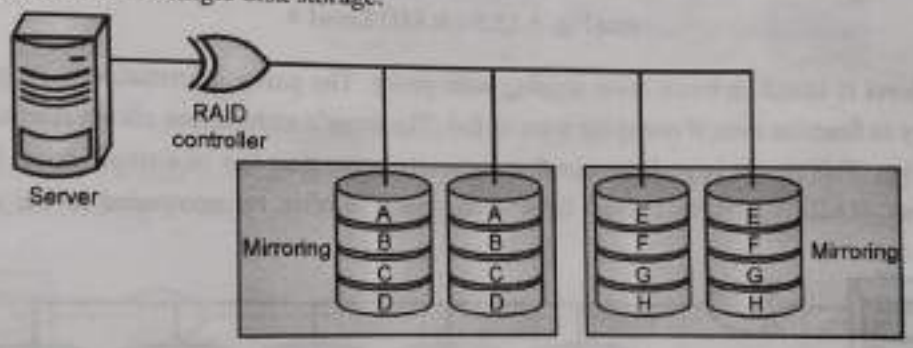
Standard Raid levels

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



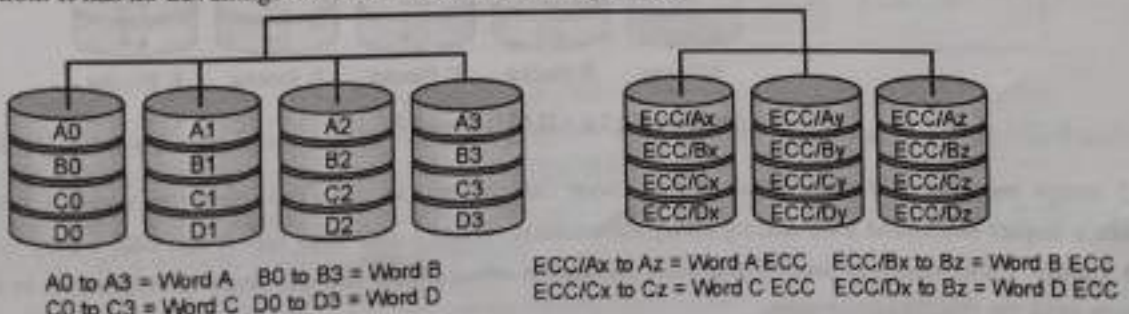
(1F16)**Fig. 5.15.1 : RAID Level 0**

- **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.



(1F17)**Fig. 5.15.2 : RAID Level 1**

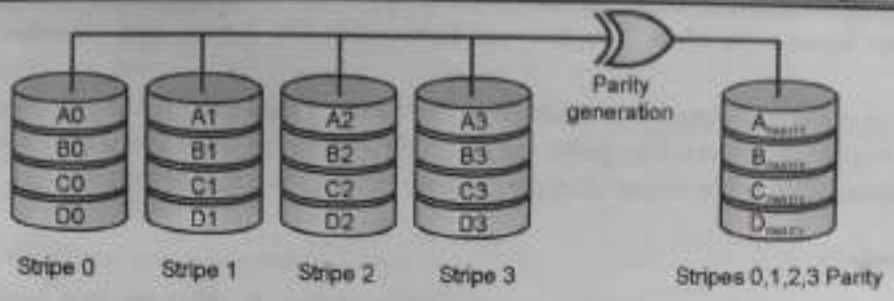
- **RAID 2 :** This configuration uses striping across disks with some disks storing Error Checking and Correcting (ECC) information. It has no advantage over RAID 3 and is no longer used.



(1F18)**Fig. 5.15.3 : RAID Level 2**

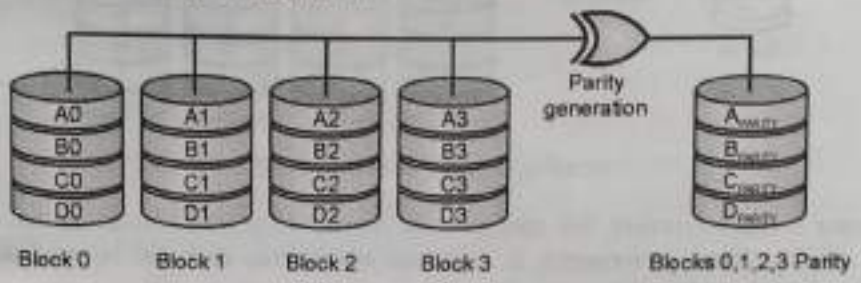
- **RAID 3 :** This technique uses striping and dedicates one drive to storing parity information. The embedded ECC information is used to detect errors. Data recovery is accomplished by calculating the exclusive OR (XOR) of the information recorded on the other drives. Since an I/O operation addresses all 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.

Module 5



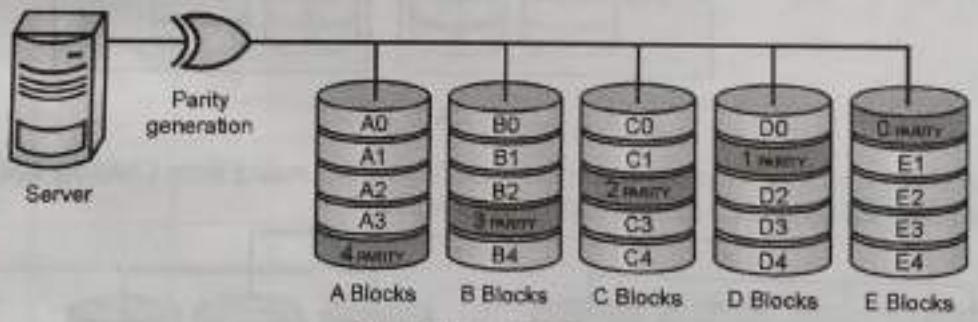
(1F18)**Fig. 5.15.4 : RAID Level 3**

- **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.



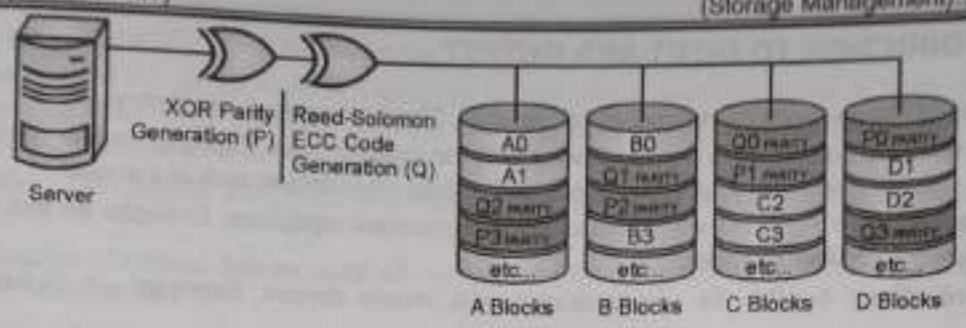
(1F20)**Fig. 5.15.5 : RAID Level 4**

- **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.



(1F21)**Fig. 5.15.6 : RAID Level 5**

- RAID 5 arrays are generally considered to be a poor choice for use on write-intensive systems because of the performance impact associated with writing parity information. When a disk does fail, it can take a long time to rebuild a RAID 5 array. Performance is usually degraded during the rebuild time and the array is vulnerable to an additional disk failure until the rebuild is complete.
- **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.

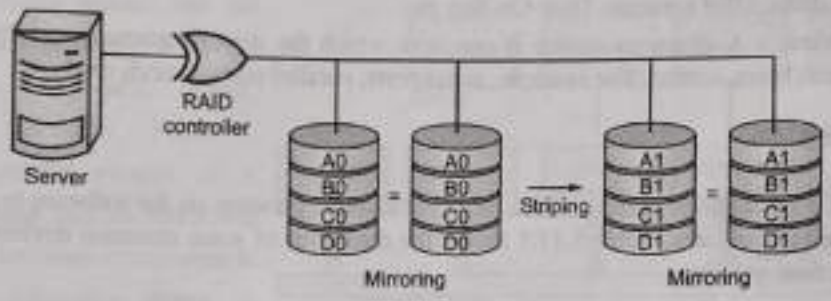


(1F22)**Fig. 5.15.7 : RAID Level 6**

Nested RAID levels

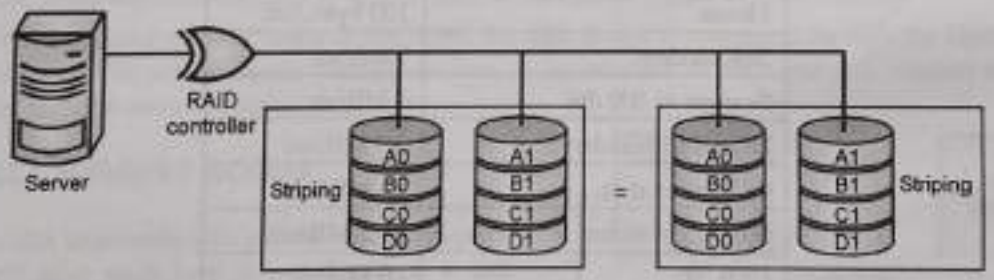
Some RAID levels are referred to as *nested RAID* because they are based on a combination of RAID levels. Here are some examples of nested RAID levels.

- **RAID 10 (RAID 1 + 0) :** Combining RAID 1 and RAID 0, this level is often referred to as RAID 10, which offers higher performance than RAID 1 but at a much higher cost. In RAID 1+0, the data is mirrored and the mirrors are striped.



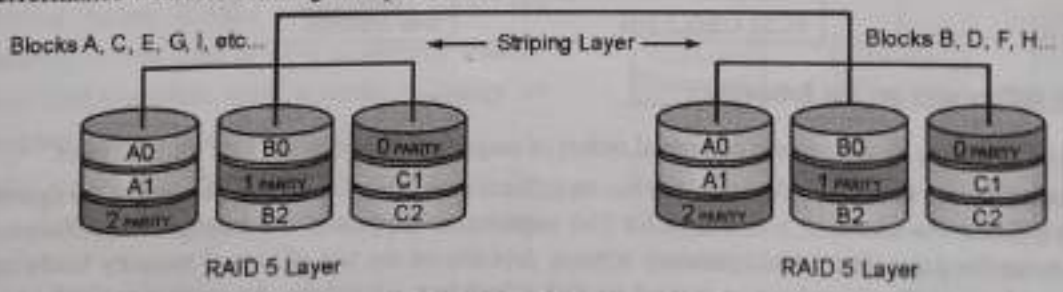
(1F23)**Fig. 5.15.8 : RAID Level 10**

- **RAID 01 (RAID 0 + 1) :** RAID 0 + 1 is very similar to RAID 1 + 0, except the data organization method is slightly different. Rather than creating a mirror and then stripping the mirror, RAID 0 + 1 creates a stripe set and then mirrors the stripe set.



(1F24)**Fig. 5.15.9 : RAID Level 01**

- **RAID 03 (RAID 0 + 3 also known as RAID 53 or RAID 5 + 3) :** This level uses striping (in RAID 0 style) for RAID 3's virtual disk blocks. This offers higher performance than RAID 3, but at a much higher cost.
- **RAID 50 (RAID 5 + 0) :** This configuration combines RAID 5 distributed parity with RAID 0 striping to improve RAID 5 performance without reducing data protection.



(1F25)**Fig. 5.15.10 : RAID Level 50**

Module 5

5.16 INTRODUCTION TO INPUT AND OUTPUT

- To perform Input/ Output operation external device are used. These are grouped into three groups :
  - **Human readable :** Suitable for communicating with the computer user. Examples include printers and terminals, the latter consisting of video display, keyboard, and perhaps other devices such as a mouse.
  - **Machine readable :** Suitable for communicating with electronic equipment. Examples are disk drives, USB keys, sensors, controllers, and actuators.
  - **Communication :** Suitable for communicating with remote devices. Examples are digital line drivers and modems.
- One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.
- An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories –
  - **Block devices :** A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
  - **Character devices :** A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc.

5.17 IO DEVICES

- I/O devices cover a wide range in speeds, which puts considerable pressure on the software to perform well over many orders of magnitude in data rates. Table 5.17.1 shows the data rates of some common devices. Most of these devices tend to get faster as time goes on.

Table 5.17.1 : IO Device and Data rates

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital camcorder	3.5 MB/sec
4x Blu-ray disc	18 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
FireWire 800	100 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
SCSI Ultra 5 bus	640 MB/sec

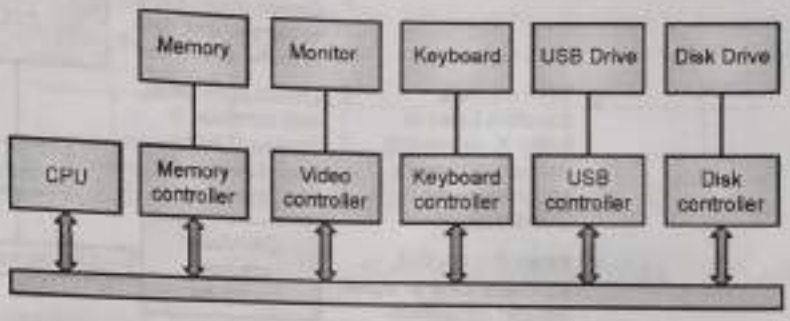
Among the key differences are the following :

- **Data rate :** There may be differences of several orders of magnitude between the data transfer rates.
- **Application :** The use to which a device is put has an influence on the software and policies in the operating system and supporting utilities. For example, a disk used for files requires the support of file management software. A disk used as a backing store for pages in a virtual memory scheme depends on the use of virtual memory hardware and software. Furthermore, these applications have an impact on disk scheduling algorithms. As another example, a terminal may be used by an ordinary user or a system administrator.

- These uses imply different privilege levels and perhaps different priorities in the operating system.
- **Complexity of control :** A printer requires a relatively simple control interface. A disk is much more complex. The effect of these differences on the operating system is filtered to some extent by the complexity of the I/O module that controls the device, as discussed in the next section.
- **Unit of transfer :** Data may be transferred as a stream of bytes or characters (e.g., terminal I/O) or in larger blocks (e.g., disk I/O).
- **Data representation :** Different data encoding schemes are used by different devices, including differences in character code and parity conventions.
- **Error conditions :** The nature of errors, the way in which they are reported, their consequences, and the available range of responses differ widely from one device to another.

5.18 DEVICE CONTROLLER

- Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.
- The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, scanner, camcoder, Pendrive, projector etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller shown in Fig. 5.18.1.

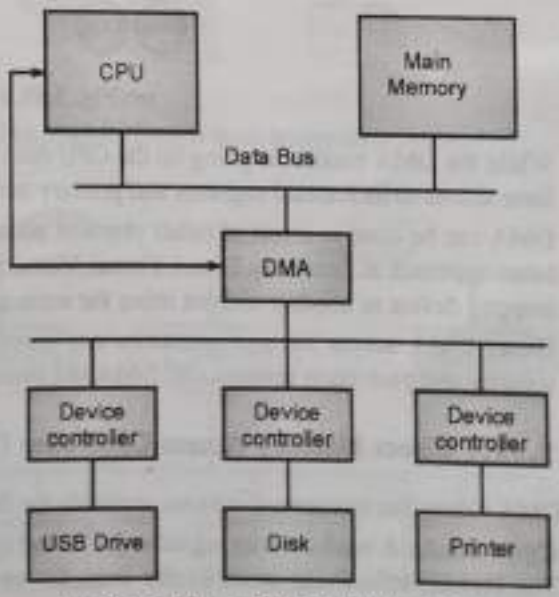


(1F1)**Fig. 5.18.1 : Device Controller**

- The controller's job is to convert the serial bit stream into a block of bytes and perform any error correction necessary. The block of bytes is typically first assembled, bit by bit, in a buffer inside the controller. After its checksum has been verified and the block has been declared to be error free, it can then be copied to main memory.
- There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

5.19 DIRECT MEMORY ACCESS

- Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So, a typical computer uses direct memory access (DMA) hardware to reduce this overhead shown in Fig. 5.19.1.
- Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device.
- CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.



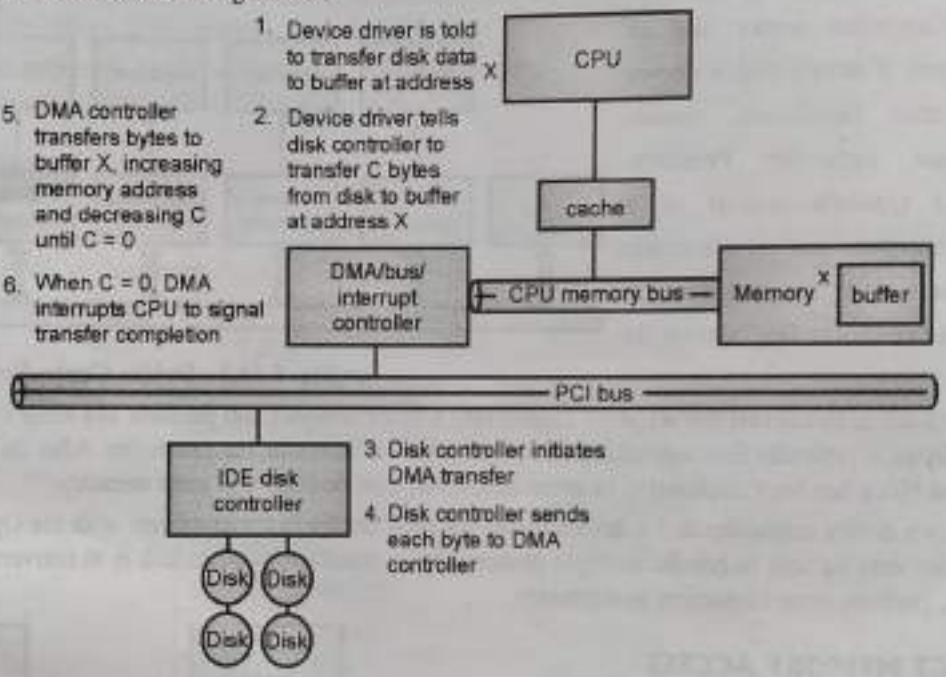
(1F4)**Fig. 5.19.1 : Direct Memory Access**

Module 5

- Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus.
- The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

Operation

- For devices that transfer large quantities of data (such as disk controllers), it is wasteful to tie up the CPU transferring data in and out of registers one byte at a time. Instead this work can be off-loaded to a special processor, known as the Direct Memory Access, DMA, Controller. The host issues a command to the DMA controller, indicating the location where the data is located, the location where the data is to be transferred to, and the number of bytes of data to transfer. The DMA controller handles the data transfer, and then interrupts the CPU when the transfer is complete.
- A simple DMA controller is a standard component in modern PCs, and many bus-mastering I/O cards contain their own DMA hardware. Handshaking between DMA controllers and their devices is accomplished through two wires called the DMA-request and DMA-acknowledge wires.



(IFS)**Fig. 5.19.2 : Steps in a DMA transfer**

- While the DMA transfer is going on the CPU does not have access to the PCI bus (including main memory), but it does have access to its internal registers and primary and secondary caches.
- DMA can be done in terms of either physical addresses or virtual addresses that are mapped to physical addresses. The latter approach is known as Direct Virtual Memory Access, DVMA, and allows direct data transfer from one memory-mapped device to another without using the main memory chips.
- Direct DMA access by user processes can speed up operations, but is generally forbidden by modern systems for security and protection reasons. (IC DMA is a kernel-mode operation.) Fig. 5.19.2 illustrates the DMA process.

5.19.1 Direct Memory Access Controller (DMAC) Options for Data Transfer

The DMA Controller has several options available for the transfer of data. They are :

1. **Cycle Steal :** A read or write signal is generated by the DMAC, and the I/O device either generates or latches the data. The DMAC effectively steals cycles from the processor in order to transfer the byte, so single byte transfer is also known as cycle stealing.

2. **Burst Transfer :** To achieve block transfers, some DMAC's incorporate an automatic sequencing of the value presented on the address bus. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.

3. **Hidden DMA :** It is possible to perform hidden DMA, which is transparent to the normal operation of the CPU. In other words, the bus is grabbed by the DMAC when the processor is not using it. The DMAC monitors the execution of the processor, and when it recognises the processor executing an instruction which has sufficient empty clock cycles to perform a byte transfer, it waits till the processor is decoding the op code, then grabs the bus during this time. The processor is not slowed down, but continues processing normally. Naturally, the data transfer by the DMAC must be completed before the processor starts.

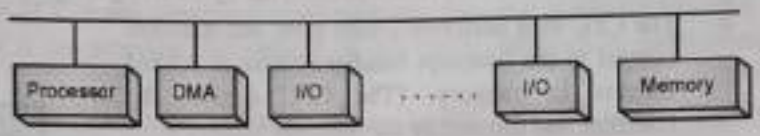
5.19.2 Configurations of DMA

DMA mechanism can be configured in a variety of ways, which are :

1. Single-bus, detached DMA 2. Single-bus, integrated DMA-I/O 3. I/O bus

1. **Single-bus, detached DMA :** All modules share the same system bus. The DMA module is acting as a surrogate processor, which uses programmed I/O to exchange data between memory and an I/O module through the DMA module.

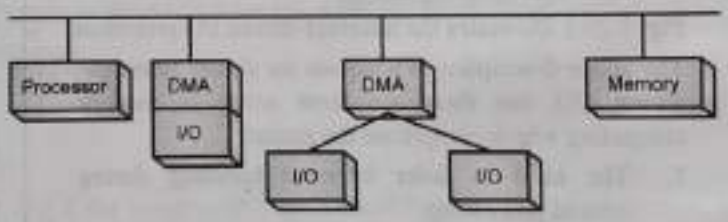
   This configuration is inexpensive, but is inefficient. This is because each transfer of a word consumes two bus cycles shown in Fig. 5.19.3.



(1F6)**Fig. 5.19.3 : Single-bus, detached DMA**

2. **Single-bus, integrated DMA :** In this configuration, there is a path between the DMA module and one or more I/O module that does not include the system bus. The DMA logic can be a part of an I/O module, or a separate module that controls one or more I/O modules. Therefore, the number of required bus cycles can be cut substantially.

   The system bus that the DMA module shares with the processor and memory is used by the DMA module only to exchange data with memory. The exchange of data between the DMA and I/O modules takes place off the system bus shown in Fig. 5.19.4.

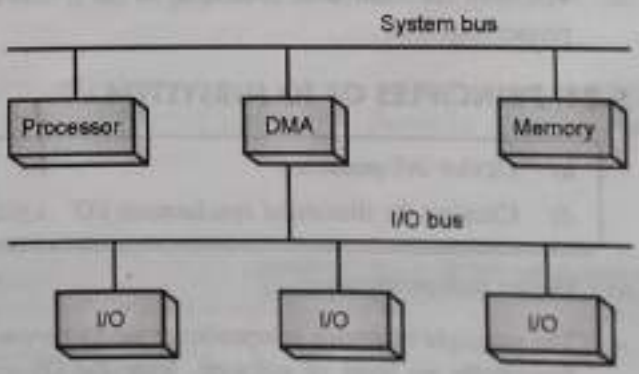


(1F7)**Fig. 5.19.4 : Single-bus, integrated DMA**

3. **I/O bus :** In this configuration, the concept is further improves from the previous configuration, which is single-bus, integrated DMA. I/O modules are connected to the DMA module using an I/O bus.

   This can reduce the number of I/O interfaces in the DMA module to one and provides for an easily expandable configuration.

   The system bus that the DMA module shares with the processor and memory is used by the DMA module only to exchange data with memory. The exchange of data between the DMA and I/O modules takes place off the system bus shown in Fig. 5.19.5.

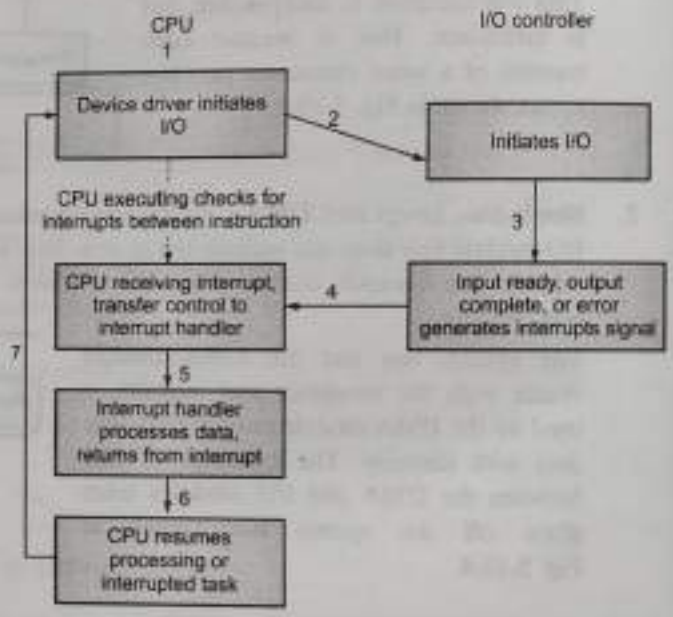


(1F8)**Fig. 5.19.5 : I/O bus**

Module 5

5.20 INTERRUPTS

- **Polling v/s Interrupts I/O :** A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as **polling** and **interrupts**. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.
- **Polling I/O :** Polling is the simplest way for an I/O device to communicate with the processor the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.
- Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls. Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.
- **Interrupts I/O :** An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention. Interrupts allow devices to notify the CPU when they have data to transfer or when an operation is complete, allowing the CPU to perform other duties when no I/O transfers need its immediate attention.
- The CPU has an interrupt-request line that is sensed after every instruction.
  - A device's controller raises an interrupt by asserting a signal on the interrupt request line.
  - The CPU then performs a state save, and transfers control to the interrupt handler routine at a fixed address in memory. (The CPU catches the interrupt and dispatches the interrupt handler.)
  - The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a return from interrupt instruction to return control to the CPU. (The interrupt handler clears the interrupt by servicing the device.)
- Fig. 5.20.1 illustrates the interrupt-driven I/O procedure.
- The above description is adequate for simple interrupt-driven I/O, but there are three needs in modern computing which complicate the picture :
  1. The need to defer interrupt handling during critical processing,
  2. The need to determine which interrupt handler to invoke, without having to poll (Search) all devices to see which one needs attention, and
  3. The need for multi-level interrupts, so the system can differentiate between high- and low-priority interrupts for proper response.



(1F9)**Fig. 5.20.1 : Interrupt-driven I/O**

5.21 PRINCIPLES OF IO SUBSYSTEM

a) Device independence	b) Uniform naming	c) Error handling
d) Creating the illusion of synchronous I/O	e) Buffering	f) Sharable vs dedicated devices

a) Device independence

- The principle of device independence works very well with respect to random access information storage devices. Essentially no level of software, from the OS upward cares what kind of storage device it's reading from/ writing to.

- It works fairly well in equating character input devices (e.g. keyboard) with disks and character output devices (e.g. a character-oriented window) with disk; which is why redirection can be done.
- For the most part, but with significant exceptions, software that reads files from disk can be applied to take its input from the keyboard and vice versa.
- For other devices "device independence" works with respect to some "systems" features such as naming the device, handling the interrupts, and protection.
- There are some devices that don't look like any other device at any level. The interface to the monitor is through video RAM; entirely unlike the interface to any other output device. The interface to the clock consists wholly of the interrupt handler; again, there's no resemblance to any other device.

b) Uniform naming : Recall that we discussed the value of the name space implemented by file systems. There is no dependence between the name of the file and the device on which it is stored. So, a file called IAmStoredOnAHardDisk might well be stored on a floppy disk.

c) Error handling : There are several aspects to error handling including: detection, correction (if possible) and reporting.

- Detection should be done as close to where the error occurred as possible before more damage is done (fault containment). This is not trivial.
- Correction is sometimes easy, for example ECC (error correcting code) memory does this automatically (but the OS wants to know about the error so that it can schedule replacement of the faulty chips before unrecoverable double errors occur).
- Other easy cases include successful retries for failed ethernet transmissions. In this example, while logging is appropriate, it is quite reasonable for no action to be taken.
- Error reporting tends to be awful. The trouble is that the error occurs at a low level but by the time it is reported the context is lost. Unix/Linux in particular is horrible in this area.

d) Creating the illusion of synchronous I/O

- I/O *must* be asynchronous for good performance. That is the OS *cannot* simply wait for an I/O to complete. Instead, it proceeds with other activities and responds to the notification when the I/O has finished.
- Users (mostly) want no part of this. The code sequence :

  Read X
  Y <-- X+1
  Print Y

  should print a value one greater than that read. But if the assignment is performed before the read completes, the wrong value is assigned.
- Performance junkies sometimes do want the asynchrony so that they can have another portion of their program executed while the I/O is underway. That is they implement a mini-scheduler in their application code.

e) Buffering : Often needed to hold data for examination prior to sending it to its desired destination. But this involves copying and takes time.

f) Sharable vs dedicated devices : For devices like printers and tape drives, only one user at a time is permitted. These are called serially reusable devices. Devices like disks and Ethernet ports can be shared by processes running concurrently.

Module 5

5.22 ORGANIZATION OF THE I/O FUNCTION

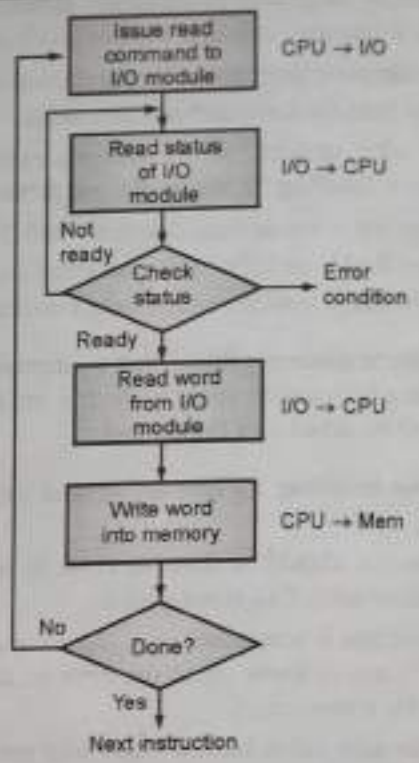
UQ. 5.22.1 Explain Interrupt driven and Programmed IO. MU(IT) - Q. 5(c), Dec. 17, 10 Marks

There are three fundamentally different ways that I/O can be performed.

1. Programmed IO 2. Interrupt Driven IO 3. IO using DMA

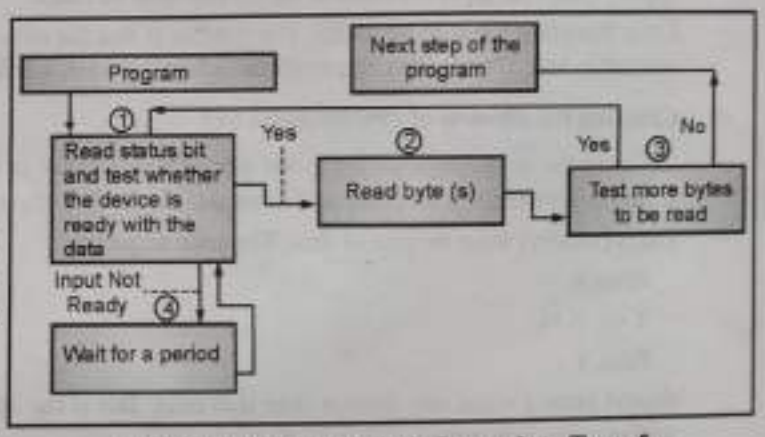
5.22.1 Programmed IO

- Programmable I/O is one of the I/O technique other than the interrupt-driven I/O and direct memory access (DMA). The programmed I/O was the most simple type of I/O technique for the exchanges of data or any types of communication between the processor and the external devices. With programmed I/O, data are exchanged between the processor and the I/O module.
- Programmed I/O (PIO) refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device.
- The CPU issues a command then waits for I/O operations to be complete. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.
- The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as Polling. As a result, the level of the performance of the entire system is severely degraded.



(a) Programmed I/O Mode Input Data Transfer

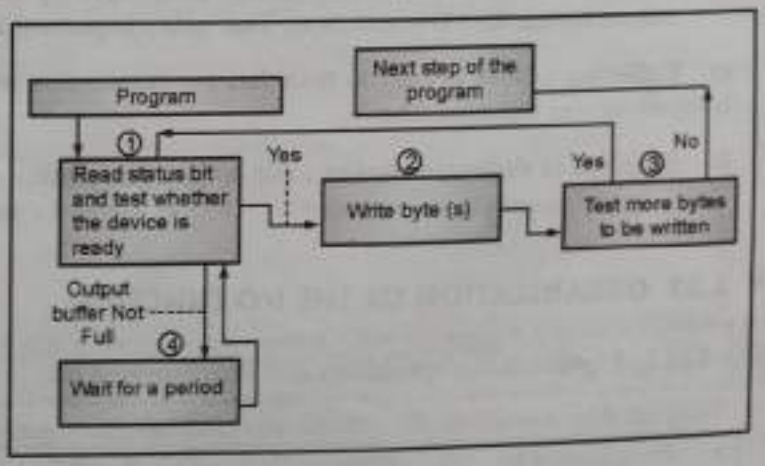
- Each input is read after first testing whether the device is ready with the input (a state reflected by a bit in a status register) shown in Fig. 5.22.1.
- The program waits for the ready status by repeatedly testing the status bit and till all targeted bytes are read from the input device. The program is in busy (non-waiting) state only after the device gets ready else in wait state.



(1F10)**Fig. 5.22.1 : IO Mode Input data Transfer**

(b) Programmed I/O Mode Output Data Transfer

- Each output written after first testing whether the device is ready to accept the byte at its output register or output buffer is empty shown in Fig. 5.22.2.
- The program waits for the ready status by repeatedly testing the status bit(s) and till all the targeted bytes are written to the device. The program in busy (non-waiting) state only after the device gets ready else wait state.



(1F11)**Fig. 5.22.2 : Programmed IO Output data Transfer**

(c) I/O Commands

To execute an I/O-related instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command. There are four types of I/O commands that an I/O module may receive when it is addressed by a processor :

- **Control :** Used to activate a peripheral and tell it what to do. For example, a magnetic-tape unit may be instructed to rewind or to move forward one record. These commands are tailored to the particular type of peripheral device.
- **Test :** Used to test various status conditions associated with an I/O module and its peripherals. The processor will want to know that the peripheral of interest is powered on and available for use. It will also want to know if the most recent I/O operation is completed and if any errors occurred.
- **Read :** Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer. The processor can then obtain the data item by requesting that the I/O module place it on the data bus.
- **Write :** Causes the I/O module to take an item of data (byte or word) from the data bus and subsequently transmit that data item to the peripheral.

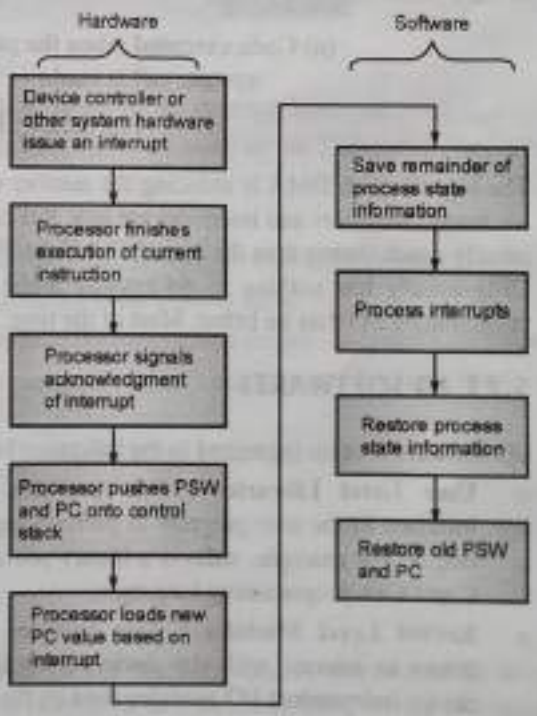
The main limitation of programmed I/O is given below :

- It used only in some low-end microcomputers.
- It has single input and single output instruction.
- Each instructions selects one I/O device (by number) and transfers a single character (byte)
- Example : Microprocessor controlled video terminal.
- Four registers : Input status and character, output status and character.

5.22.2 Interrupt Driven IO

GQ. 5.22.1 Explain Interrupt driven IO and discuss the advantages of Interrupt driven IO over programmed IO.

- Interrupt driven I/O is an alternative scheme dealing with I/O. Interrupt I/O is a way of controlling input/output activity whereby a peripheral or terminal that needs to make or receive a data transfer sends a signal. This will cause a program interrupt to be set. The interrupt system, the processors enter an interrupt service routine.
- The function of the routine will depend upon the system of interrupt levels and priorities that is implemented in the processor.
- Fig. 5.22.3 shows the simple interrupt processing.
- For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports or memory mapping.
- For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.



(IF12)**Fig. 5.22.3 : Interrupt Driven IO**

Module 5

a) Basic Operations of Interrupt

- CPU issues read command.
- I/O module gets data from peripheral whilst CPU does other work.
- I/O module interrupts CPU.
- CPU requests data.
- I/O module transfers data.

b) The main Limitations Interrupt-driven I/O

- Primary disadvantage of programmed I/O is that CPU spends most of its time in a tight loop waiting for the device to become ready. This is called busy waiting.
- With interrupt-driven I/O, the CPU starts the device and tells it to generate an interrupt when it is finished.
- Done by setting interrupt-enable bit in status register.
- Still requires an interrupt for every character read or written.
- Interrupting a running process is an expensive business (requires saving context).
- Requires extra hardware (DMA controller chip).
- All these limitation can be overcome by the Introduction of DMA (Direct Memory Access)

5.22.3 IO using DMA (Direct Memory Access)

- An obvious disadvantage of interrupt-driven I/O is that an interrupt occurs on every character. Interrupts take time, so this scheme wastes a certain amount of CPU time. A solution is to use DMA. Here the idea is to let the DMA controller feed the characters to the printer one at time, without the CPU being bothered. In essence, DMA is programmed I/O, only with the DMA controller doing all the work, instead of the main CPU. This strategy requires special hardware (the DMA controller) but frees up the CPU during the I/O to do other work. An outline of the code is given in Fig. 5.22.4.

```
Copy_from_user(buffer,p,count);
Set_up_DMA_controller();
Scheduler();
```

(a) Code executed when the print system call is made

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

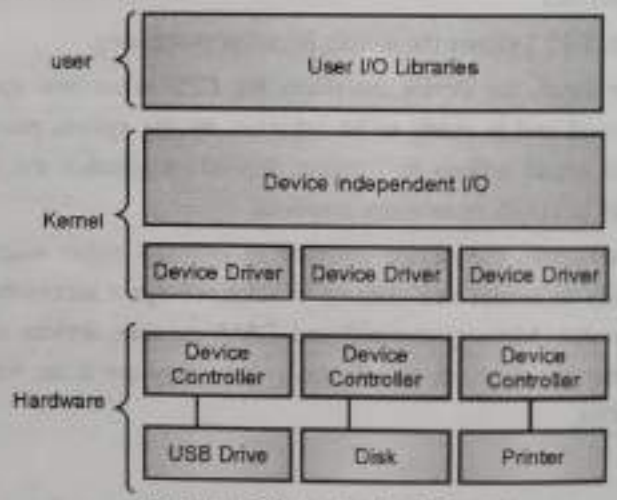
(b) Interrupt-service procedure

Fig. 5.22.4 : Printing a string using DMA

- The big win with DMA is reducing the number of interrupts from one per character to one per buffer printed. If there are many characters and interrupts are low, this can be a major improvement. On the other hand, the DMA controller is usually much slower than the main CPU. If the DMA controller is not capable of driving the device at full speed, or the CPU usually has nothing to do anyway while waiting for the DMA interrupt, then interrupt-driven I/O or even programmed I/O may be better. Most of the time, though, DMA is worth it.

5.23 IO SOFTWARES

- I/O software is often organized in the following layers :
  - **User Level Libraries :** This provides simple interface to the user program to perform input and output. For example, **stdio** is a library provided by C and C++ programming languages.
  - **Kernel Level Modules :** This provides device driver to interact with the device controller and device independent I/O modules used by the device drivers.
  - **Hardware :** This layer includes actual hardware and hardware controller which interact with the device drivers and makes hardware alive.



(1F13)**Fig. 5.23.1 : Device Drivers**

- A key concept in the design of I/O software is that it should be device independent where it should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.

5.23.1 Device Drivers

- Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.
- A device driver performs the following jobs :
  - To accept request from the device independent software above to it.
  - Interact with the device controller to take and give I/O and perform required error handling.
  - Making sure that the request is executed successfully.
- How a device driver handles a request is as follows : Suppose a request comes to read a block N. If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the new request in the queue of pending requests.

Interrupt handlers

- An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.
- When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.
- The interrupt mechanism accepts an address - a number that selects a specific interrupt handling routine/function from a small set. In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.
- The steps that do occur may be in a different order on some machines.
  1. Save any registers (including the PSW) that have not already been saved by the interrupt hardware.
  2. Set up a context for the interrupt service procedure. Doing this may involve setting up the TLB, MMU and a page table.
  3. Set up a stack for the interrupt service procedure.
  4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
  5. Copy the registers from where they were saved (possibly some stack) to the process table.
  6. Run the interrupt service procedure. It will extract information from the interrupting device controller's registers.
  7. Choose which process to run next. If the interrupt has caused some high-priority process that was blocked to become ready, it may be chosen to run now.
  8. Set up the MMU context for the process to run next. Some TLB setup may also be needed.
  9. Load the new process registers, including its PSW.
  10. Start running the new process.

Module 5

5.23.2 Device-Independent I/O Software

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. Though it is difficult to write completely device independent software but we can write some modules which are common among all the devices. Following is a list of functions of device-independent I/O Software :

- Uniform interfacing for device drivers.

- Device naming - Mnemonic names mapped to Major and Minor device numbers.
- Device protection.
- Providing a device-independent block size.
- Buffering because data coming off a device cannot be stored in final destination.
- Storage allocation on block devices.
- Allocation and releasing dedicated devices.
- Error Reporting.

The device-independent code does most of the functionality, but not necessarily most of the code since there can be *many* drivers all doing essentially the same thing in slightly different ways due to slightly different controllers.

- **Naming :** Again an important OS functionality. Must offer a consistent interface to the device drivers.
  - In Unix this is done by associating each device with a (special) file in the /dev directory.
  - The i-node for these files contains
  - . an indication that these are special files and also contain so called major and minor device numbers.
  - The major device number gives the number of the driver. (These numbers are rather ad hoc, they correspond to the position of the function pointer to the driver in a table of function pointers.)
  - The minor number indicates for which device (e.g., which scsi cdrom drive) the request is intended
- **Protection :** A wide range of possibilities are actually done in real systems. Including both extreme examples of everything is permitted and nothing is permitted (directly).
  - In ms-dos any process can write to any file.
  - In IBM and other mainframe OS's, normal processors do not access devices. Indeed the main CPU doesn't issue the I/O requests. Instead an I/O channel is used and the mainline constructs a channel program and tells the channel to invoke it.
  - Unix uses normal rwx bits on files in /dev (I don't believe x is used).
- **Buffering :** Is necessary since requests come in a size specified by the user and data is delivered in a size specified by the device. Enforce exclusive access for non-shared devices like tapes.

5.23.3 User-Space I/O Software

- These are the libraries which provide richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers. Most of the user-level I/O software consists of library procedures with some exception like spooling system which is a way of dealing with dedicated I/O devices in a multiprogramming system.
- I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW. For example putchar(), getchar(), printf() and scanf() are example of user level I/O library stdio available in C programming.

5.23.4 Kernel I/O Subsystem:

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

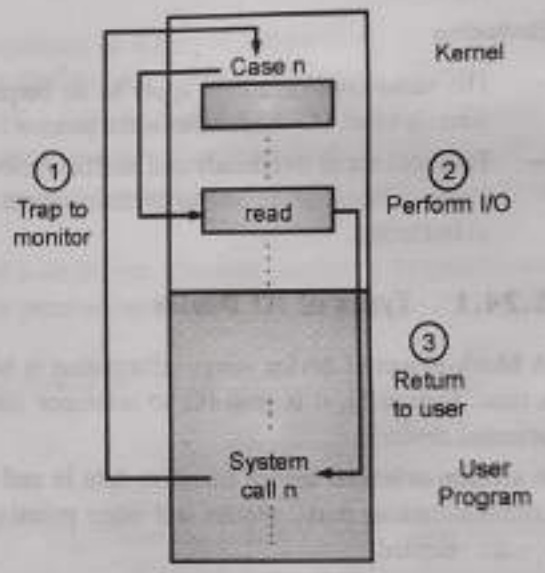
- **Scheduling :** Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.
- **Buffering :** Kernel I/O Subsystem maintains a memory area known as **buffer** that stores data while they are transferred between two devices or between a device with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.

- **Caching :** Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.
- **Spooling and Device Reservation :** A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in kernel thread.
- A spool (Simultaneous Peripheral Operations On-Line) buffers data for (peripheral) devices such as printers that cannot support interleaved data streams.
- If multiple processes want to print at the same time, they each send their print data to files stored in the spool directory. When each file is closed, then the application sees that print job as complete, and the print scheduler sends each file to the appropriate printer one at a time.
- Support is provided for viewing the spool queues, removing jobs from the queues, moving jobs from one queue to another queue, and in some cases changing the priorities of jobs in the queues.
- Spool queues can be general (any laser printer) or specific (printer number 42).
- OSes can also provide support for processes to request / get exclusive access to a particular device, and/or to wait until a device becomes available.

Error Handling

An operating system that uses protected memory can guard against many kinds of hardware and application errors.

- I/O requests can fail for many reasons, either transient (buffers overflow) or permanent (disk crash).
- I/O requests usually return an error bit (or more) indicating the problem. UNIX systems also set the global variable errno to one of a hundred or so well-defined values to indicate the specific error that has occurred. (See errno.h for a complete listing, or man errno.)
- Some devices, such as SCSI devices, are capable of providing much more detailed information about errors, and even keep an on-board error log that can be requested by the host.
- The I/O system must protect against either accidental or deliberate erroneous I/O.
- User applications are not allowed to perform I/O in user mode - All I/O requests are handled through system calls that must be performed in kernel mode.



(1F14)**Fig. 5.23.2 : Use of a system call to perform I/O**

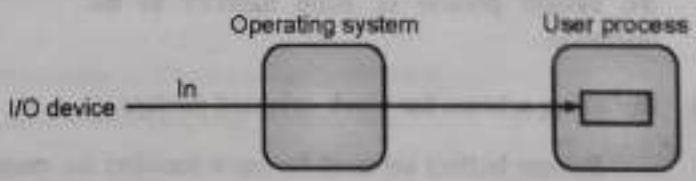
5.24 IO BUFFERING

UQ. 5.24.1 Write short on IO buffering. MU(IT) - Q. 5(c), May 16, 5 Marks

Module 5

Let say a user process wishes to read blocks of data from a disk one at a time, with each block having a length of 512 bytes.

The data are to be read into a data area within the address space of the user process at virtual location 1000 to 1511 The simplest way would be to execute an I/O command to the disk unit and then wait for the data to become available. The waiting could either be busy waiting (continuously test the device status) or, more practically, process suspension on an interrupt.



(1F29)**Fig. 5.24.1 : No Buffering**

(a) Issues with these Approach

- The program is hung up waiting for the relatively slow I/O to complete.
- Another problem is that this approach to I/O interferes with swapping decisions by the operating system.

(b) Solution to Above Problems

- Virtual locations 1000 to 1511 must remain in main memory during the course of the block transfer. Otherwise, some of the data may be lost.
- If paging is being used, at least the page containing the target locations must be locked into main memory. Thus, although portions of the process may be paged out to disk, it is impossible to swap the process out completely, even if this is desired by the operating system.

(c) Risk of single-process deadlock

- If a process issues an I/O command, is suspended awaiting the result, and then is swapped out prior to the beginning of the operation, the process is blocked waiting on the I/O event, and the I/O operation is blocked waiting for the process to be swapped in.
- To avoid this deadlock, the user memory involved in the I/O operation must be locked in main memory immediately before the I/O request is issued, even though the I/O operation is queued and may not be executed for some time.

(d) Buffering

- The same considerations apply to an output operation. If a block is being transferred from a user process area directly to an I/O module, then the process is blocked during the transfer and the process may not be swapped out.
- To avoid these overheads and inefficiencies, it is sometimes convenient to perform input transfers in advance of requests being made and to perform output transfers sometime after the request is made. This technique is known as buffering.

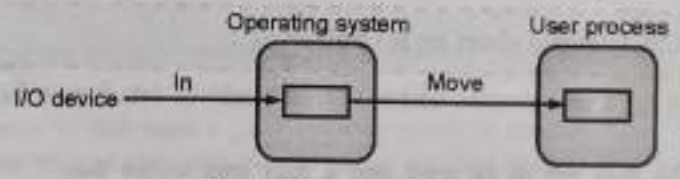
5.24.1 Types of IO Device

- A **block-oriented** device stores information in blocks that are usually of fixed size, and transfers are made one block at a time. Generally, it is possible to reference data by its block number. Disks and USB keys are examples of block-oriented devices.
- A **stream-oriented** device transfers data in and out as a stream of bytes, with no block structure. Terminals, printers, communications ports, mouse and other pointing devices, and most other devices that are not secondary storage are stream oriented.
- Some of the buffering schemes that are supported by operating systems to improve the performance of the system. The various approaches to buffering are :

1. Single buffer	2. Double Buffer	3. Circular Buffer

1. Single Buffer

It is the simplest type single buffering shown in Fig. 5.24.2. 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.



(1F30)**Fig. 5.24.2 : Single Buffering**

(i) Buffering scheme for block-oriented devices

- System buffers are used for input transfers are made to the system buffer. The process moves the block into user space and immediately requests another block as soon as transfer is complete,.
- This is called reading ahead, or anticipated input; it is done thinking that the block will be needed in future. For many types of computation, this assumption is good many times because data are usually accessed sequentially.

- But block be read in unnecessarily at the end of a sequence of processing.This approach provide a speedup operation in the system to the no system buffering. The user process can be processing one block of data till the process is over the next block is being read in.
- Swap the process out is possible both the operations are performed in different memory like the input operation is performed in system memory rather than user process memory.
- Since the operating system has to keep track of the assignment of system buffers to user processes this make the logic complicated.

(ii) Swapping logic affected

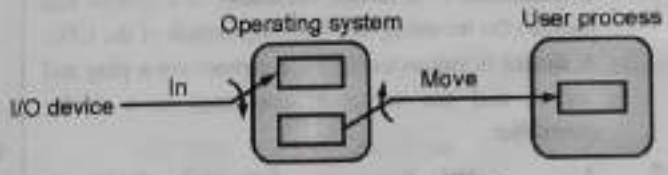
- If the I/O operation involves the same disk that is used for swapping, it hardly makes sense to queue disk writes to the same device for swapping the process out.
- This attempt to swap the process and release main memory will itself not begin until after the I/O operation finishes, at which time swapping the process to disk may no longer be appropriate.
- Similar considerations apply to block-oriented output. When data are being transmitted to a device, they are first copied from the user space into the system buffer, from which they will ultimately be written. The requesting process is now free to continue or to be swapped as necessary.

(iii) Buffering scheme for stream-oriented I/O

- The single buffering scheme can be used in a line-at-a time fashion or a byte-at-a-time fashion. Line-at-a-time operation is appropriate for scroll-mode terminals (sometimes called dumb terminals).
- With this form of terminal, user input is one line at a time, with a carriage return signaling the end of a line, and output to the terminal is similarly one line at a time. A line printer is another example of such a device.
- Byte-at-a-time operation is used on forms-mode terminals, when each keystroke is significant, and for many other peripherals, such as sensors and controllers.
- In the case of line-at-a-time I/O, the buffer can be used to hold a single line. The user process is suspended during input, awaiting the arrival of the entire line. For output, the user process can place a line of output in the buffer and continue processing.
- It need not be suspended unless it has a second line of output to send before the buffer is emptied from the first output operation.

2. Double Buffer

An improvement over single buffering can be had by assigning two system buffers to the operation (Fig. 5.24.3). A process now transfers data to (or from) one buffer while the operating system empties (or fills) the other. This technique is known as **double buffering** or **buffer swapping**.



(1F31)**Fig. 5.24.3 : Double Buffering**

(i) Double Buffering scheme for Block oriented devices

There are two buffers in the system :

- One buffer is used by the driver or controller to store data while waiting for it to be taken by higher level of the hierarchy. Other buffer is used to store data from the lower level module.
- Double buffering is also known as buffer swapping.
- A major disadvantage of double buffering is that the complexity of the process get increased.
- If the process performs rapid bursts of I/O, then using double buffering may be deficient.

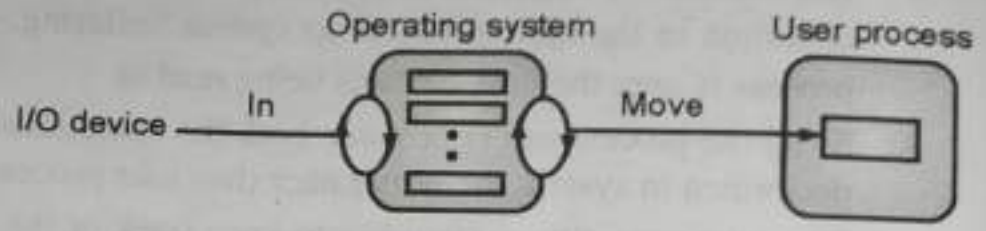
(ii) Double Buffering scheme for Stream oriented devices

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

Module 5

3. Circular Buffer

- A double-buffer scheme should smooth out the flow of data between an I/O device and a process. If the performance of a particular process is the focus of our concern, then we would like for the I/O operation to be able to keep up with the process.



(1F32)**Fig. 5.24.4 : Circular Buffering**

- Double buffering may be inadequate if the process performs rapid bursts of I/O. In this case, the problem can often be alleviated by using more than two buffers.
- When more than two buffers are used, the collection of buffers is itself referred to as a circular buffer (Fig. 5.24.4), with each individual buffer being one unit in the circular buffer.

4. The Utility of Buffering

- Buffering is a technique that smoothes out peaks in I/O demand. However, no amount of buffering will allow an I/O device to keep pace with a process indefinitely when the average demand of the process is greater than the I/O device can service.
- Even with multiple buffers, all of the buffers will eventually fill up and the process will have to wait after processing each chunk of data.
- However, in a multiprogramming environment, when there is a variety of I/O activity and a variety of process activity to service, buffering is one tool that can increase the efficiency of the operating system and the performance of individual processes.