UNIT - V

* I/O Management And Disk Scheduling *

* I/O devices :-

External devices that engage in I/O with computer systems can be roughly grouped into three categories :

*** 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.

There are great differences across classes and even substantial differences within each class. Among the key differences are the following :

1) Data rate -

There may be differences of several orders of magnitude between the data transfer rates.

2) Application -

The use to which a device is put has an influence on the software and policies in the operating system and supporting utilities.

3) 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.

4) 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).

5) Data representation –

Different data encoding schemes are used by different devices, including differences in character code and parity conventions.

6] 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.

* Organization of the I/O Function –

1> Programmed I/O –

The processor issues an I/O command, on behalf of a process, to an I/O module; that process then busy waits for the operation to be completed before proceeding.

Table – I/O Techniques –

	No interrupts	Use of interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

2> Interrupt-driven I/O –

The processor issues an I/O command on behalf of a process. There are two possibilities. If the I/O instruction from the process is nonblocking, then the processor continue

to execute instructions from the process that issued the I/O command. If the I/O instruction is blocking, then the next instruction that the processor executes is from the OS, which will put the current process in a blocked state and schedule another process.

3] Direct memory access (DMA):-

A DMA module controls the exchange of data between main memory and an I/O module. The processor sends a request for the transfer of a block of data to the DMA module & is interrupted only after the entire block has been transferred.

* Direct Memory Access –

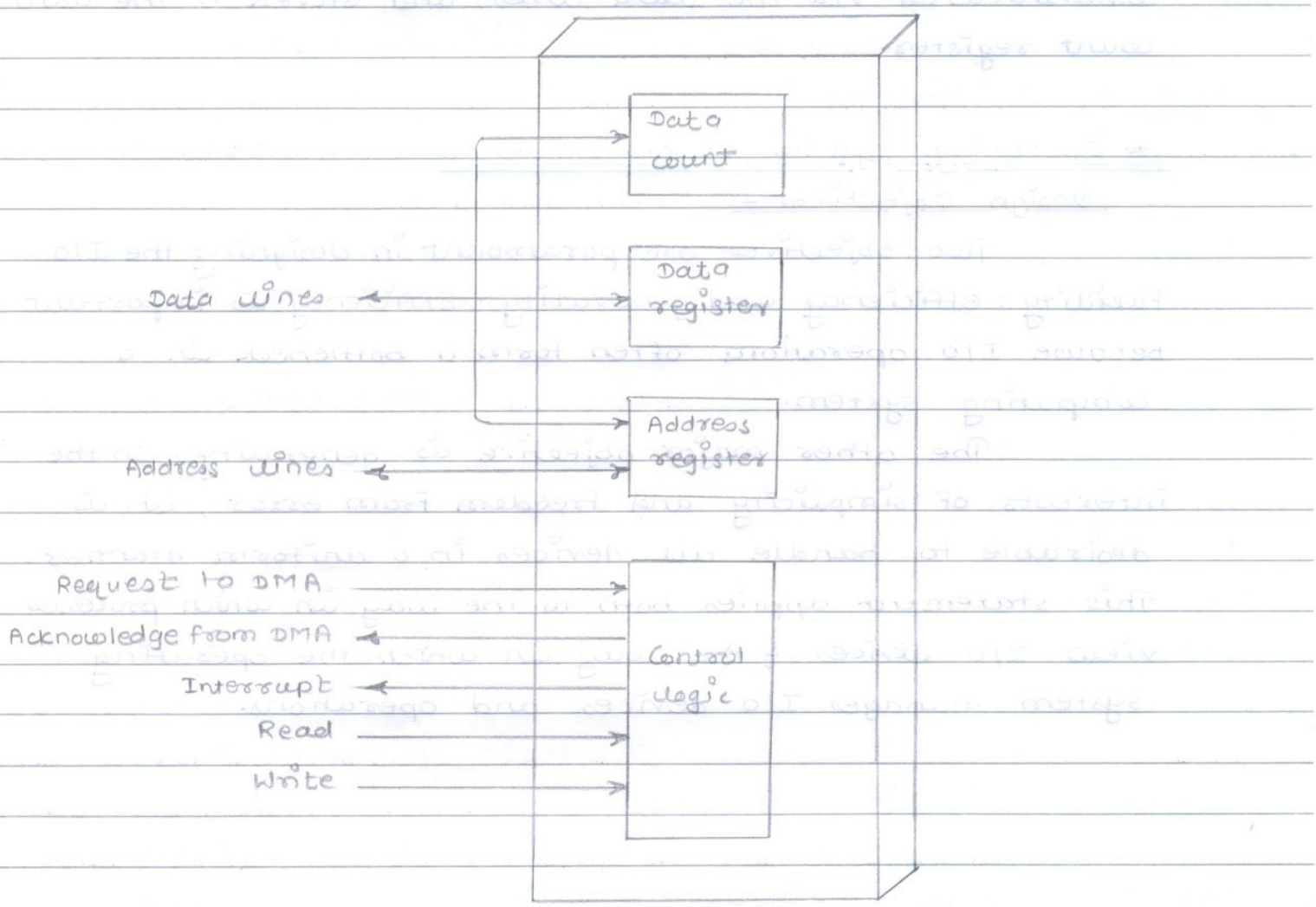


Fig- Typical DMA Block Diagram

The DMA technique works as follows. When the processor wishes to read or write a block of data, it issues a command to the DMA module by sending to the DMA module the following information:

- Whether a read or write is requested, using the read or write control line between the processor and the DMA module.
- The address of the I/O device involved, communicated on the data lines.
- The starting location in memory to read from or write to, communicated on the data lines & stored by the DMA module in its address register.
- The number of words to be read or written, again communicated via the data lines and stored in the data count register.

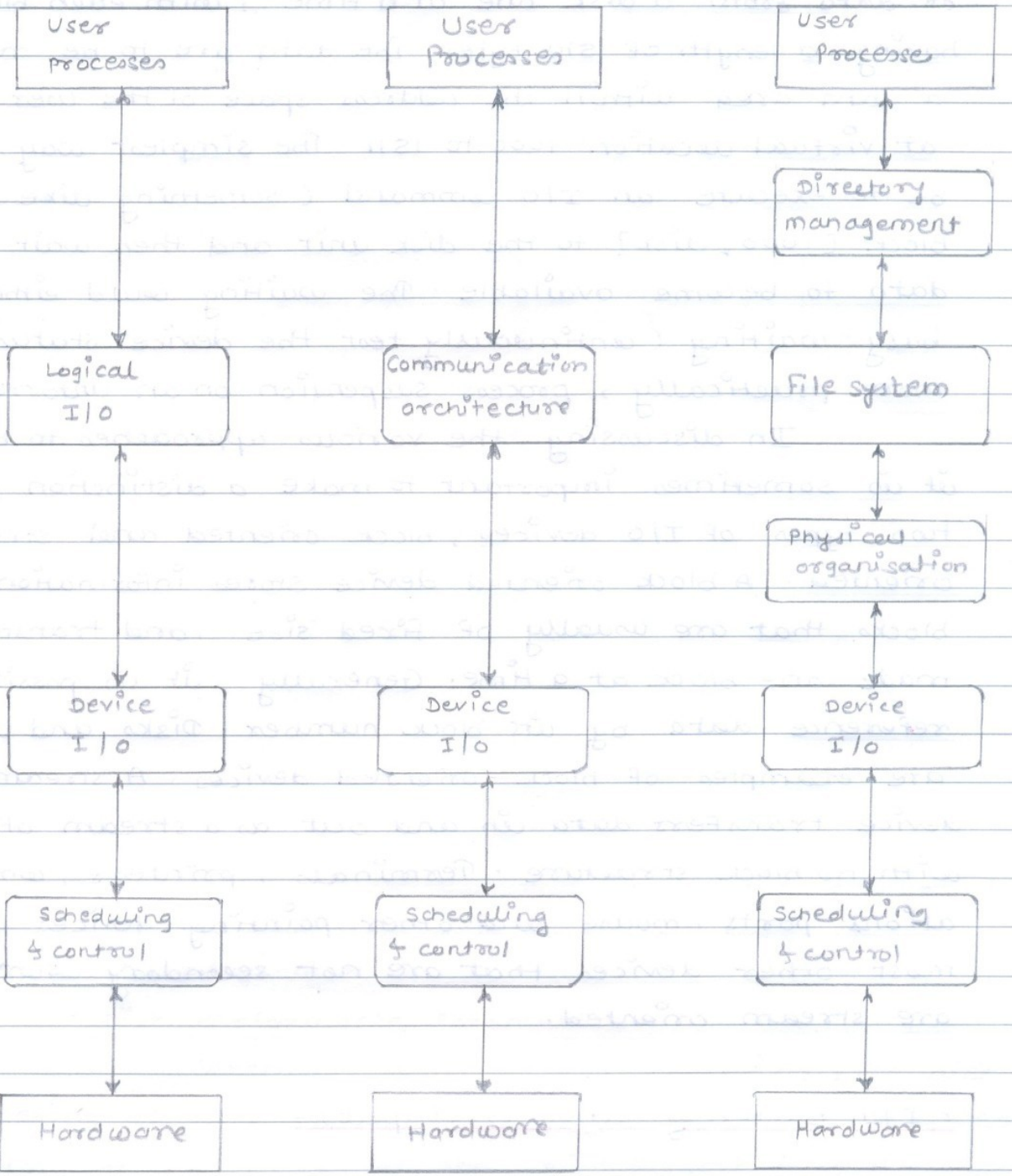
* Operating System Design Issues –

Design Objectives –

Two objectives are paramount in designing the I/O facility: efficiency and generality. Efficiency is important because I/O operations often form a bottleneck in a computing system.

The other major objective is generality. In the interests of simplicity and freedom from error, it is desirable to handle all devices in a uniform manner. This statement applies both to the way in which processes view I/O devices & the way in which the operating system manages I/O devices and operations.

* Logical Structure of the I/O Function –



a] Local peripheral device b] Communications port c) Filesystem

Fig - A model of I/O organization

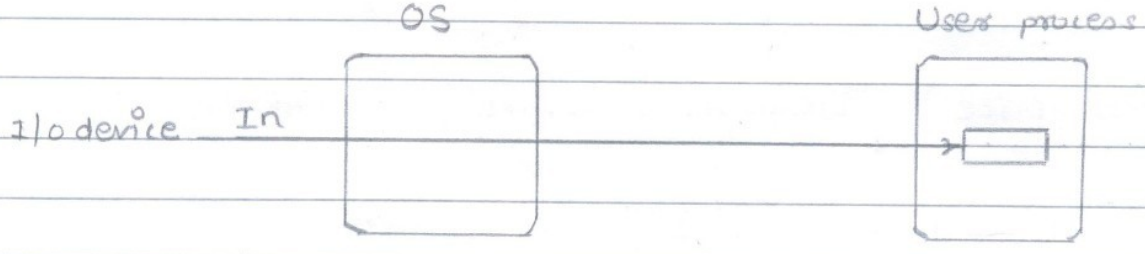
DATE :

* I/O Buffering –

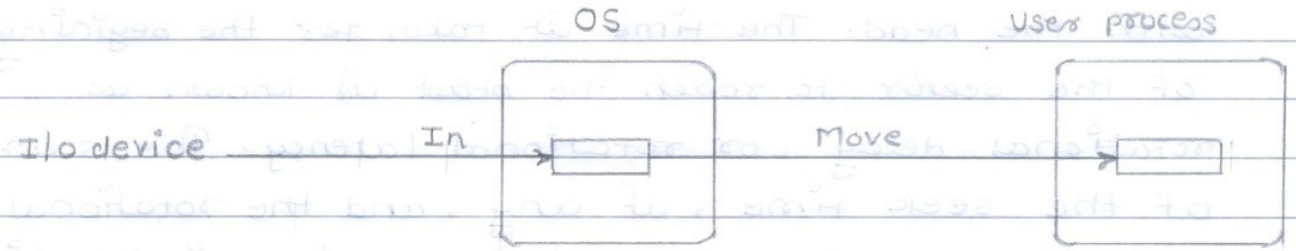
Suppose that a user process wishes to read blocks of data from a disk one at a time, with each block having a length of 512 bytes. The data are to be read into a data area within the address space of the user process at virtual location 1000 to 1511. The simplest way would be to execute an I/O command (something like Read-Block [1000, disk] 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.

In discussing the various approaches to buffering, it is sometimes important to make a distinction between two types of I/O devices; block oriented and stream oriented. 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.

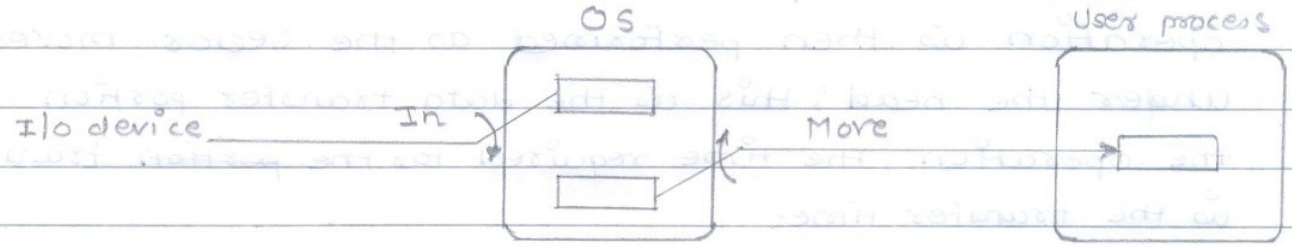
* I/O Buffering Schemes (input) –



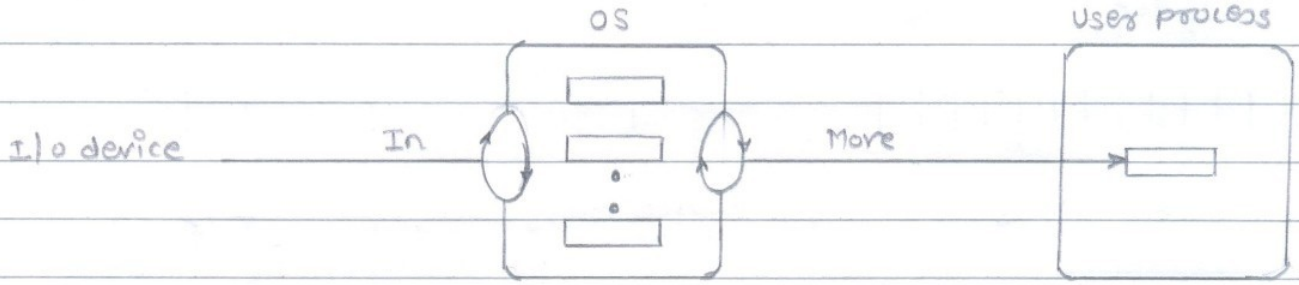
a] no buffering



b) Single buffering



c) Double buffering



d) Circular buffering

Fig - I/o Buffering Schemes (input)

* Disk Scheduling *

* Disk Performance Parameters -

When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired track & at the beginning of the desired sector on that track. Track selection involves moving the head in a movable head system or electronically selecting one head on a fixed head system.

On a movable head system, the time it takes to position the head at the track is known as seek time. In either case, once the track is selected, the disk controller waits until the appropriate sector rotates to line up

DATE :

with the head. The time it takes for the beginning of the sector to reach the head is known as rotational delay, or rotational latency. The sum of the seek time, if any, and the rotational delay equals the access time, which is the time it takes to get into position to read or write. Once the head is in position, the read or write operation is then performed as the sector moves under the head; this is the data transfer portion of the operation; the time required for the transfer is the transfer time.

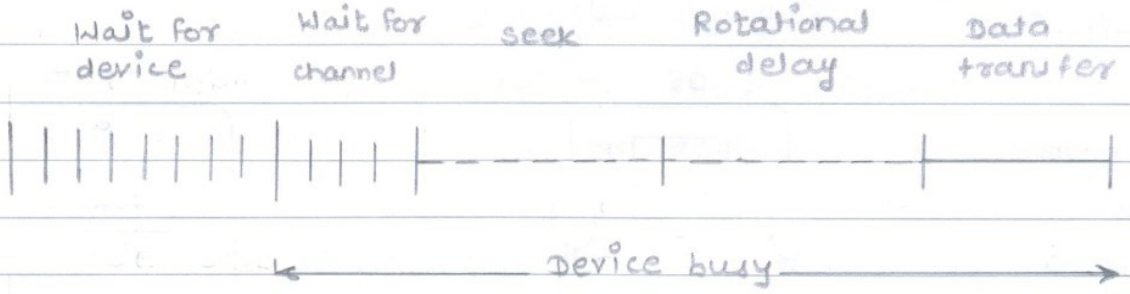
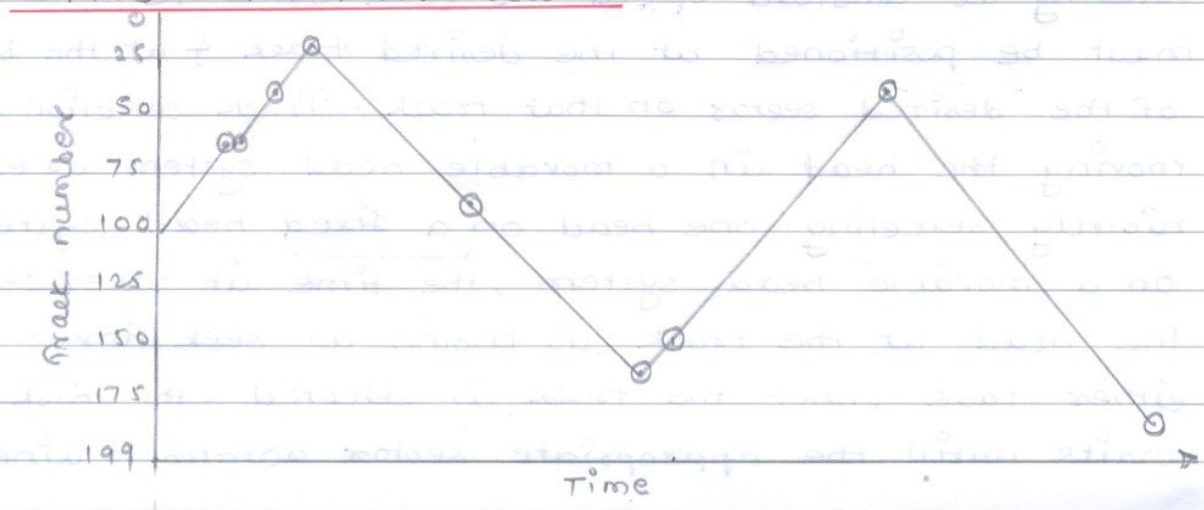


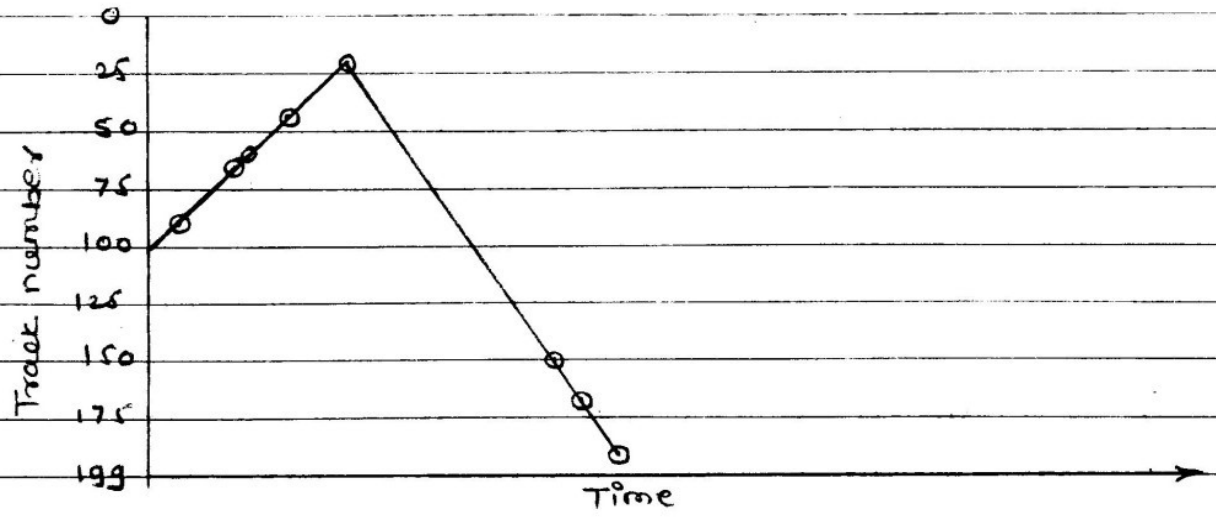
Fig - Timing of a Disk I/O Transfer

* Disk Scheduling Policies -

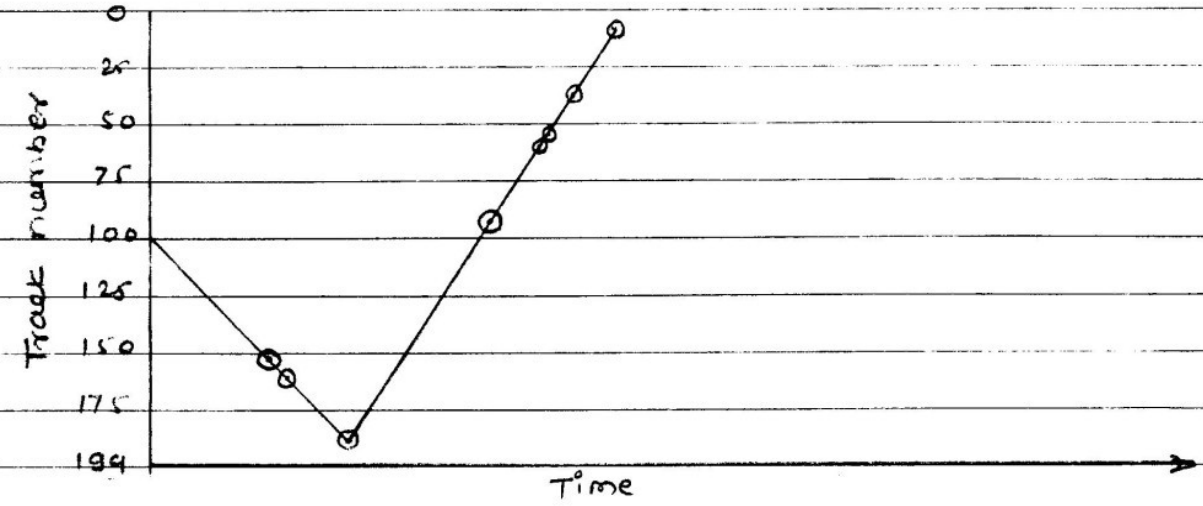
The requested tracks, in the order received by the disk scheduler, are 55, 58, 39, 18, 90, 160, 150, 38, 184.

* First - in - First - out -

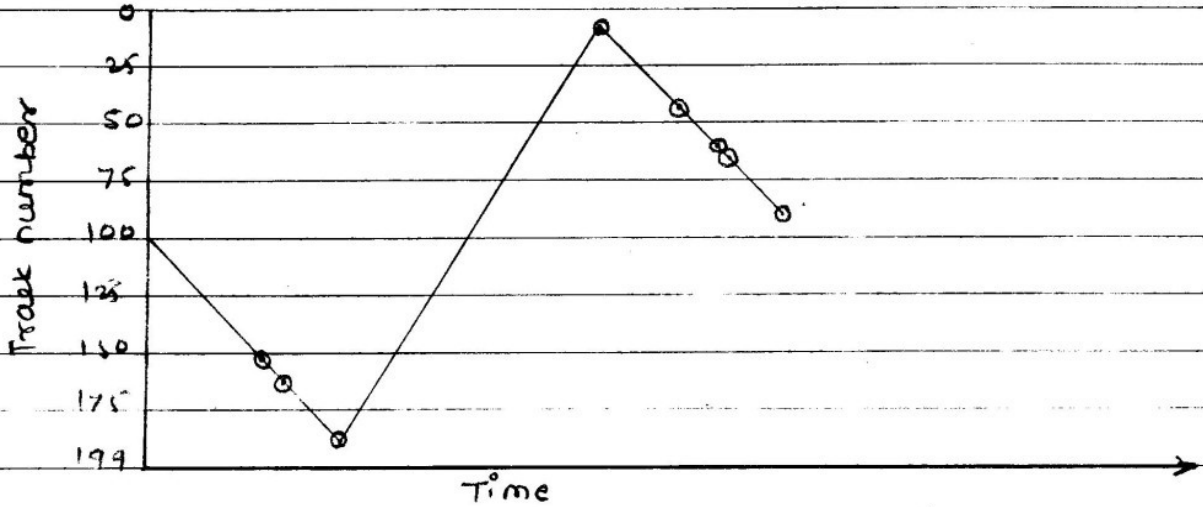




b) SSTF



c) SCAN



d) C-SCAN

Table – Comparison of Disk scheduling Algorithms –

a] FIFO (starting at track 100)		b] SSTF (starting at track 100)		c] SCAN (starting at track 100, in the direction of increasing track number)		d] C-SCAN (starting at track 100, in the direction of increasing track numbers)	
Next track accessed	No. of tracks traversed	Next track accessed	No. of tracks traversed	Next track accessed	No. of tracks traversed	Next track accessed	No. of tracks traversed
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Avg. Seek length	55.3	Avg. seek length	27.5	Avg. seek length	27.8	Avg. seek length	35.8

* Table – Disk scheduling algorithm

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis & simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management.
LIFO	Last in first out	Maximize locality & resource utilization.

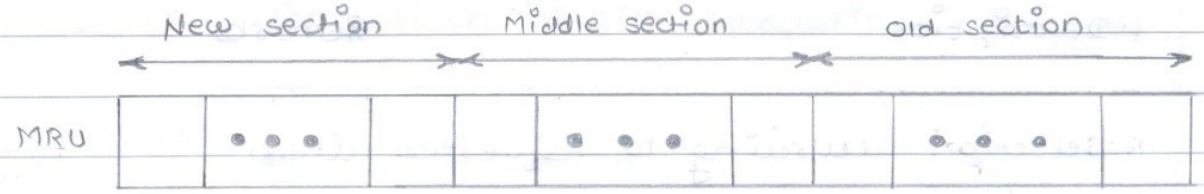
Name	Description	Remarks
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	N-step-SCAN with N= queue size at begining of SCAN cycle.	Load sensitive

* Disk Cache —

The term cache memory is usually used to apply to a memory that is smaller and faster than main memory and that is interposed between main memory and the processor : such a cache memory reduces average memory access time by exploiting the principle of locality.

* Design Considerations —

a] FIFO



b] Use of three sections

* Linux I/O –

In general terms, the Linux I/O kernel facility is very similar to that of other UNIX implementation, such as SVR4. The Linux kernel associates a special file with each I/O device driver. Block, character, and network devices are recognized.

* Disk Scheduling –

The default disk scheduler in Linux 2.4 is known as the Linus Elevator, which is a variation on the LOOK algorithm. For Linux 2.6, the Elevator algorithm has been augmented by two additional algorithms: the deadline I/O scheduler and the anticipatory I/O scheduler [LOVE 04].

* The Elevator Scheduler –

The elevator scheduler maintains a single queue for disk read and write requests and performs both sorting and merging functions on the queue. In general terms, the elevator scheduler keeps the list of requests sorted by block number. Thus, as the disk requests are handled, the drive moves in a single direction, satisfying each request as it is encountered.

* Deadline Scheduler –

* Deadline Scheduler -

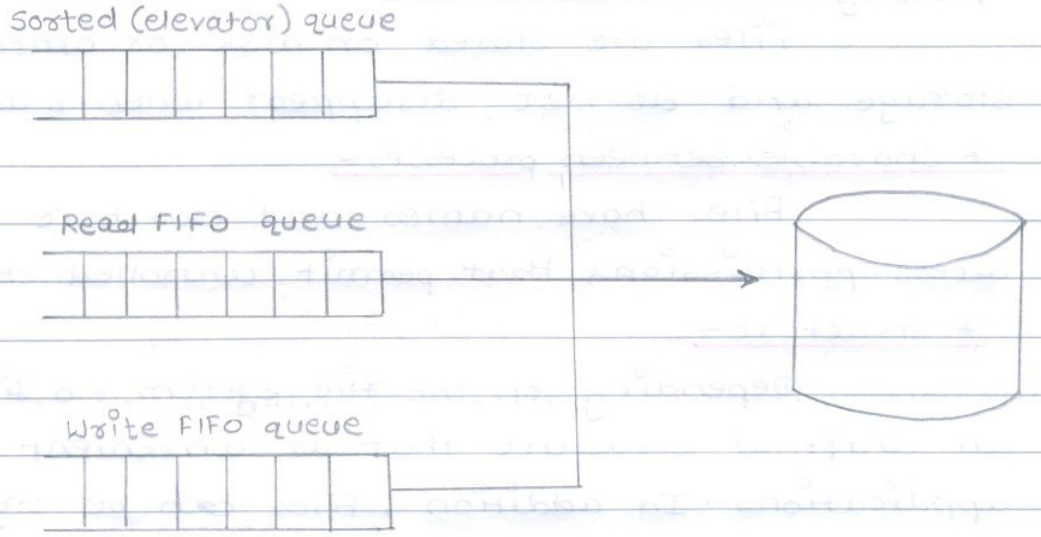


Fig- The Linux Deadline I/O Scheduler

* Anticipatory I/O Scheduler -

The original elevator scheduler & the deadline scheduler both are designed to dispatch a new request as soon as the existing request is satisfied, thus keeping the disk as busy as possible.

* Linux Page Cache -

The kernel maintained a page cache for reads and writes from regular file system files and for virtual memory pages, and a seperate buffer cache for block I/O. For Linux 2.4 and later, there is a single unified page cache that is involved in all traffic between disk and main memory.

* Files and File systems -

From the user's point of view, one of the most important parts of an operating system is the file system. 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 existance -

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 Structure -

Four terms are in common use when discussing files.

1) Field -

A field is a basic element of data. An individual field contains a single value, such as an employee's last name, a date or the value of a sensor reading.

2) Record -

A record is a collection of related ~~files~~ fields that can be treated as a unit by some application program.

3) File -

A file is a collection of similar records. The file is treated as a single entity by users and applications and may be referenced by name.

4) Database -

A database is collection of related data.

* File Management Systems -

A file management system is that set of system software that provides services to users and applications in the use of files. [GROS 86] suggests the following objectives for a file management system:

- To meet the data management needs and requirements of the user, which include storage of data and the ability to perform the aforementioned operations.
- To guarantee, to the extent possible, that the data in the file are valid.
- 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 a variety of storage device types.
- To minimize or eliminate the potential for lost or destroyed data.
- To provide standaradized set of I/O interface routine to user processes.

* File system Architecture -

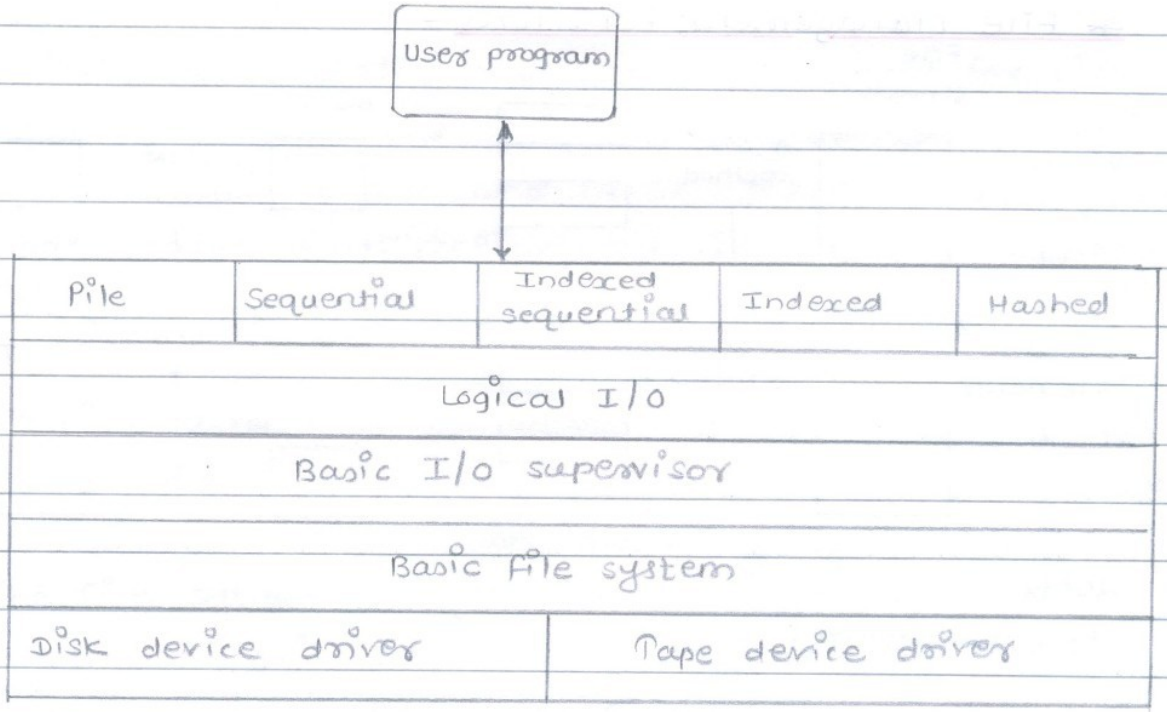


Fig - File system Software Architecture.

At the lowest level, device drivers communicate directly with peripheral devices or their controllers or channels. A device driver is responsible for starting I/O operations on a device and processing the completion of an I/O request.

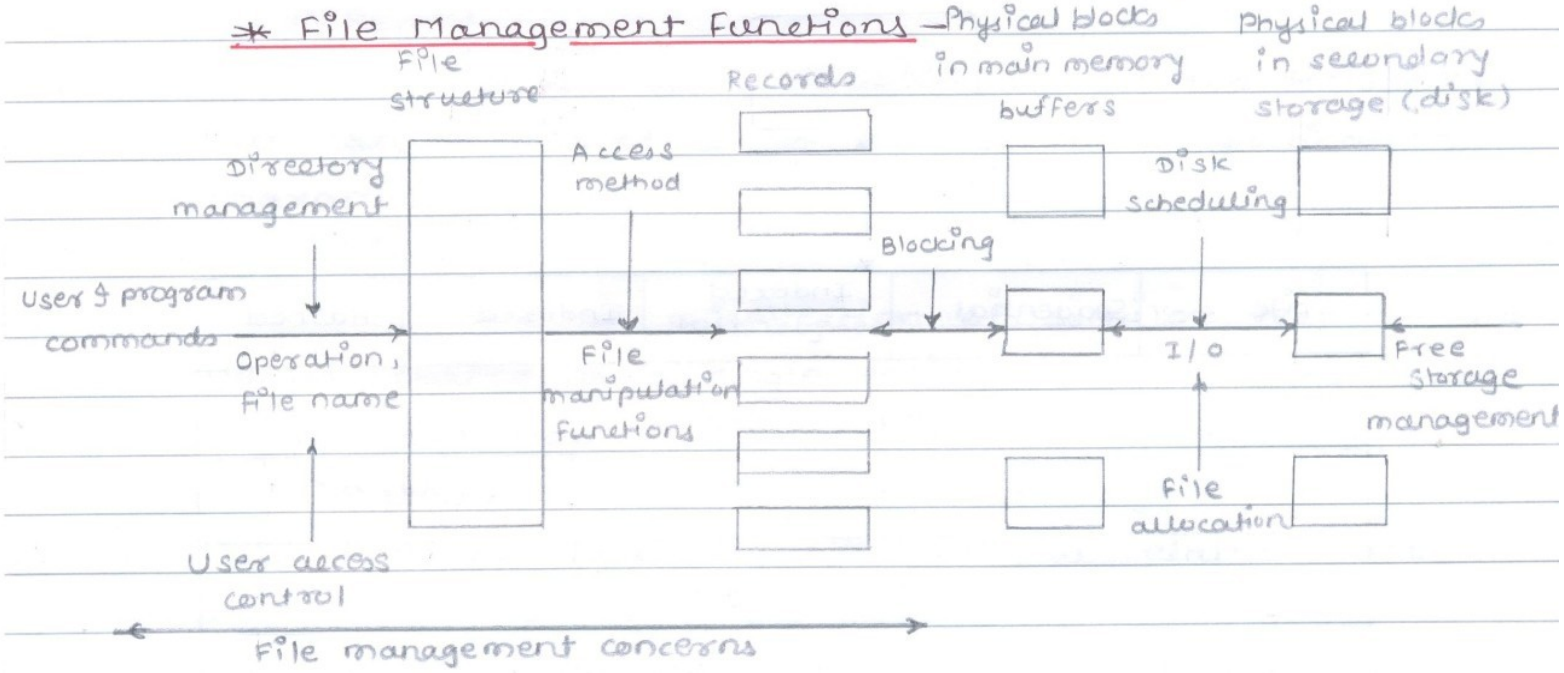
The next level is referred to as the basic file system, or the physical I/O level. This is the primary interface with the environment outside of the computer system.

The basic I/O supervisor is responsible for all file I/O initiation and termination. At this level, control structures are maintained that deal with device I/O, scheduling, and file status.

Logical I/O enables users and applications to access records. Thus, whereas the basic file system deals with blocks of data, the logical I/O module deals with the file records.

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.

* File Management Functions



* File Organization And Access -

In this section, we use term file organization to refer to 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, issues dealt with later.

In choosing a file organization, several criteria are important:

- Short access time
- Ease of update
- Economy of storage
- Simple maintenance
- Reliability.

A file stored on CD-ROM will never be updated, and so ease of update is not an issue.

* The Pile -

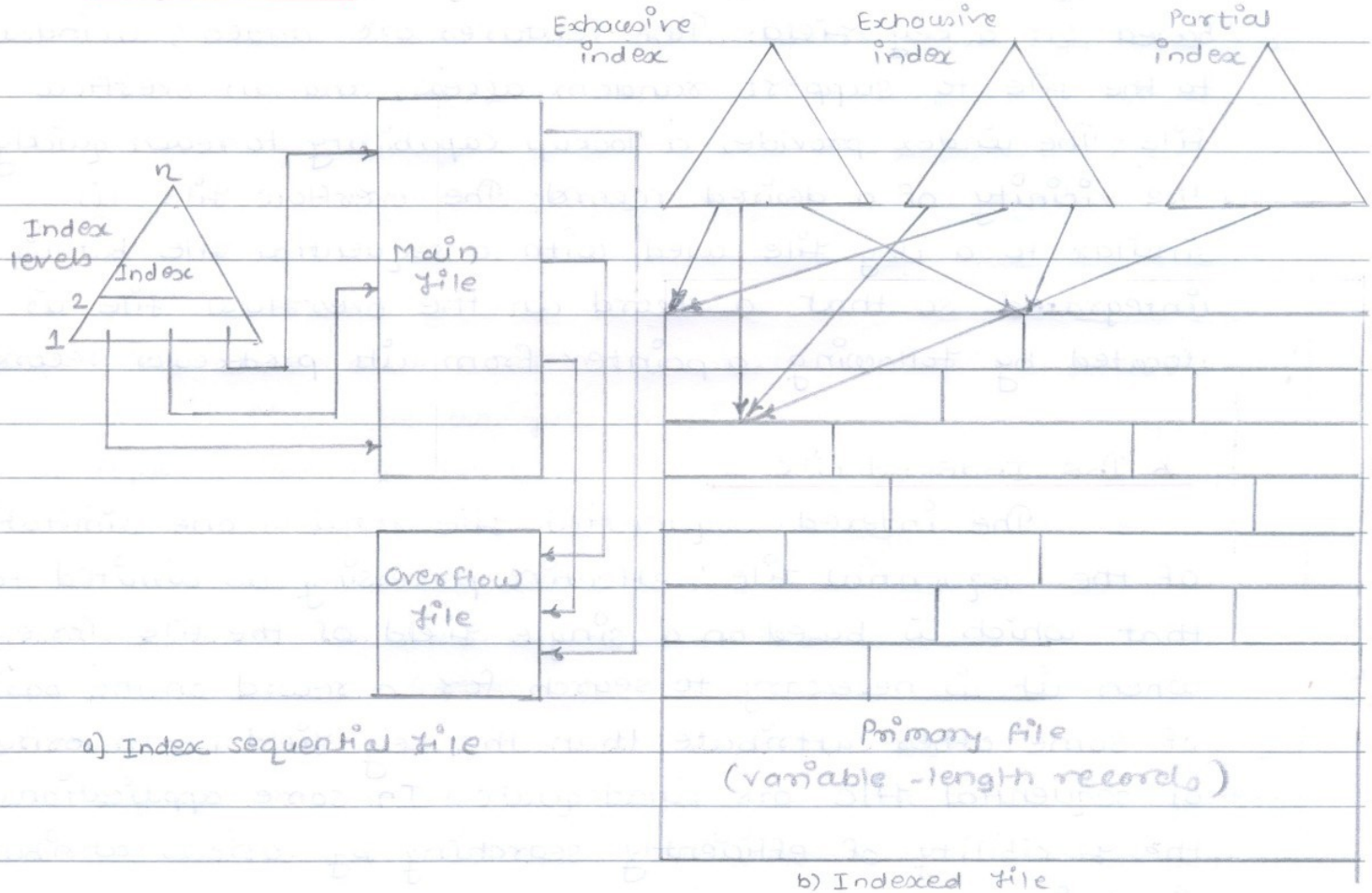


Fig- Common File Organizations

* The Sequential File -

The most common form of file structure is the sequential file. In this type of a file, a fixed format is used for records. All records are of the same length, consisting of the same number of fixed-length fields in a particular order. Because the length & position of each field are known, only the values of fields need to be stored; the field name and length for each field are attributes of the file structure.

An alternative is to organize the sequential file physically as a linked list.

* The Index sequential File -

A popular approach to overcoming the disadvantages of the sequential file is the indexed sequential file. The indexed sequential file maintains the key characteristic of the 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 reach quickly the vicinity of a desired record. The overflow file is similar to a 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.

* The Indexed File -

The indexed sequential file retains one limitation of the sequential file: effective processing is limited to that which is based on a single field of the file, for e.g. when it is necessary to search for a record on the basis of some other attribute than the key field, both forms of sequential file are inadequate. In some applications, the flexibility of efficiently searching by various attributes is desirable.

* The 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.

The direct file makes use of hashing on the key value.

* File Directories –

Contents –

Associated with any file management system and collection of files is a file directory. The directory contains information about the files, including attributes, location, and ownership. Much of this information, especially that concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines. Although some of the information in directories is available to users and applications, this is generally provided indirectly by system routines.

Table – Information Elements of a File Directory.

Basic Information
File name – Name as chosen by creator (user or program). Must be unique within a specific directory.
File type – For e.g. – text, binary, load module, etc.
File organisation – For system 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 & block no. on disk)

Size used – Current size of the file in bytes, words, or blocks
Size Allocated – The maximum size of the file.

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 & 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 Back up – 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 / processes that have the file open, whether it is locked by a process, & whether the file has been updated in main memory but not yet on disk.

* Structure -

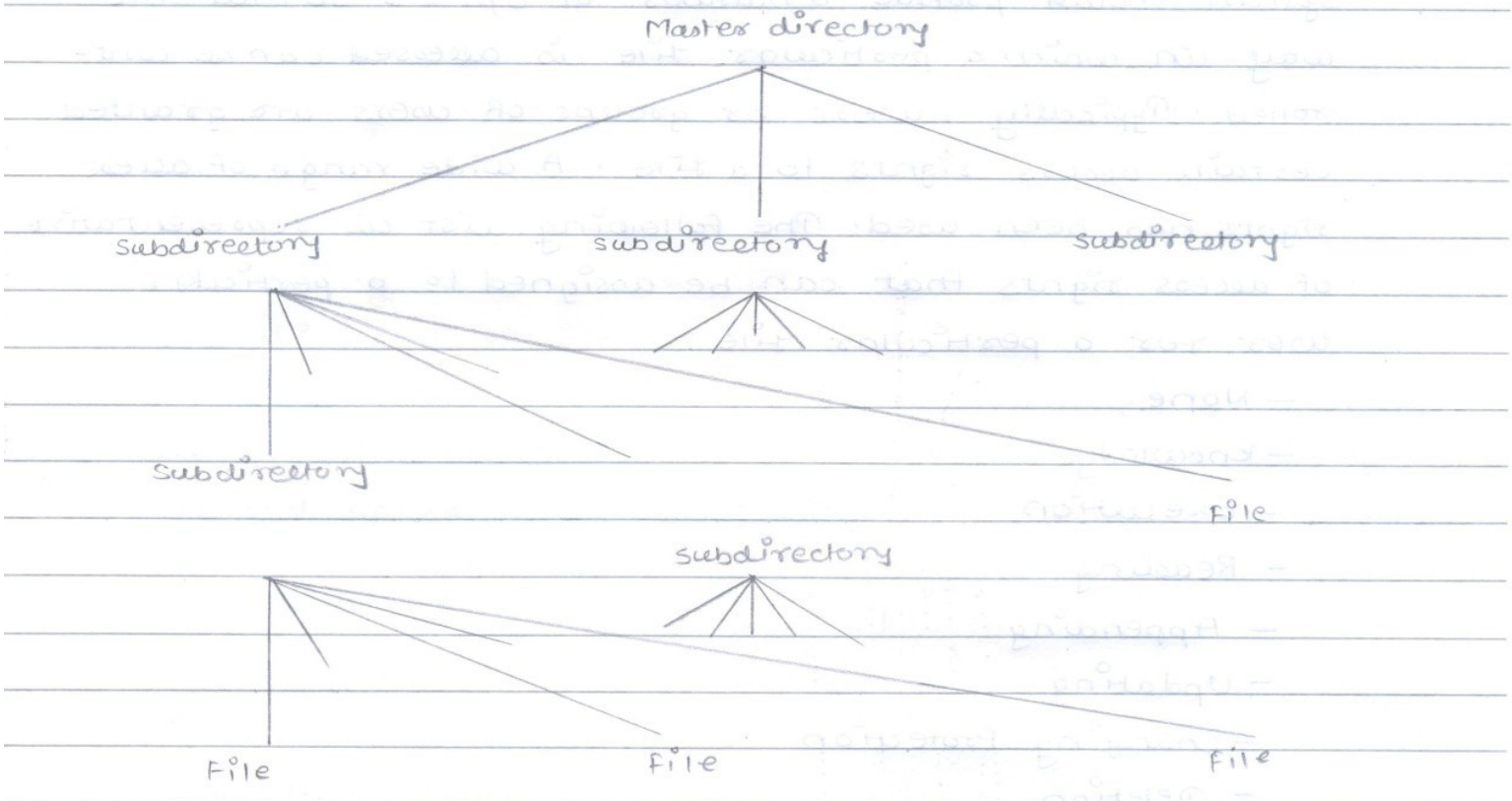


Fig - Tree - structured Directory

* 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. On the other hand, it is an unacceptable burden on users to require that they provide unique names, especially in a shared system.

* File Sharing -

In a multiuser system, there is almost always a requirement for allowing files to be shared among a number of users. Two issues arise: access rights and the management of simultaneous access.

Access Rights -

The file system should provide a flexible tool for

allowing extensive file sharing among users. The file system should provide a number of options so that the way in which a particular file is accessed can be controlled. Typically, users or groups of users are granted certain access rights to a file. A wide range of access rights has been used. The following list is representative of access rights that can be assigned to a particular user for a particular file:

- None
- Knowledge
- Execution
- Reading
- Appending
- Updating
- Changing Protection
- Deletion

Access can be provided to different classes of users:

* Specific user:

Individual users who are designated by user ID.

* 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.

* All:

All users who have access to this system. These are public files.

* 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.

* Record Blocking –

Records are the logical unit of access of a structured file, whereas blocks are the unit of I/O with secondary storage. For I/O to be performed, records must be organized as blocks.

Given the size of a block, there are three methods of blocking that can be used.

Fixed blocking –

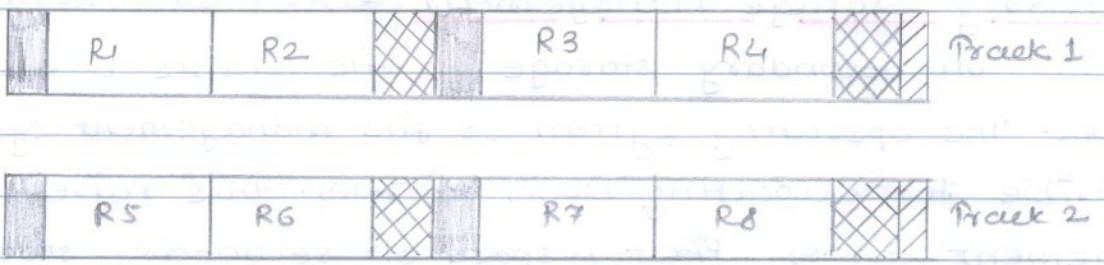
Fixed-length records are used, and an integral number of records are stored in a block. There may be unused space at the end of each block. This is referred to as internal fragmentation.

Variable-length spanned blocking –

Variable-length records are used and are packed into blocks with no unused space. Thus, some records must span two blocks, with the continuation indicated by a pointer to the successor block.

Variable-length unspanned blocking –

Variable-length records are used, but spanning is not employed. There is wasted space in most blocks because of the inability to use the remainder of a block if the next record is larger than the remaining unused space.



Fixed blocking

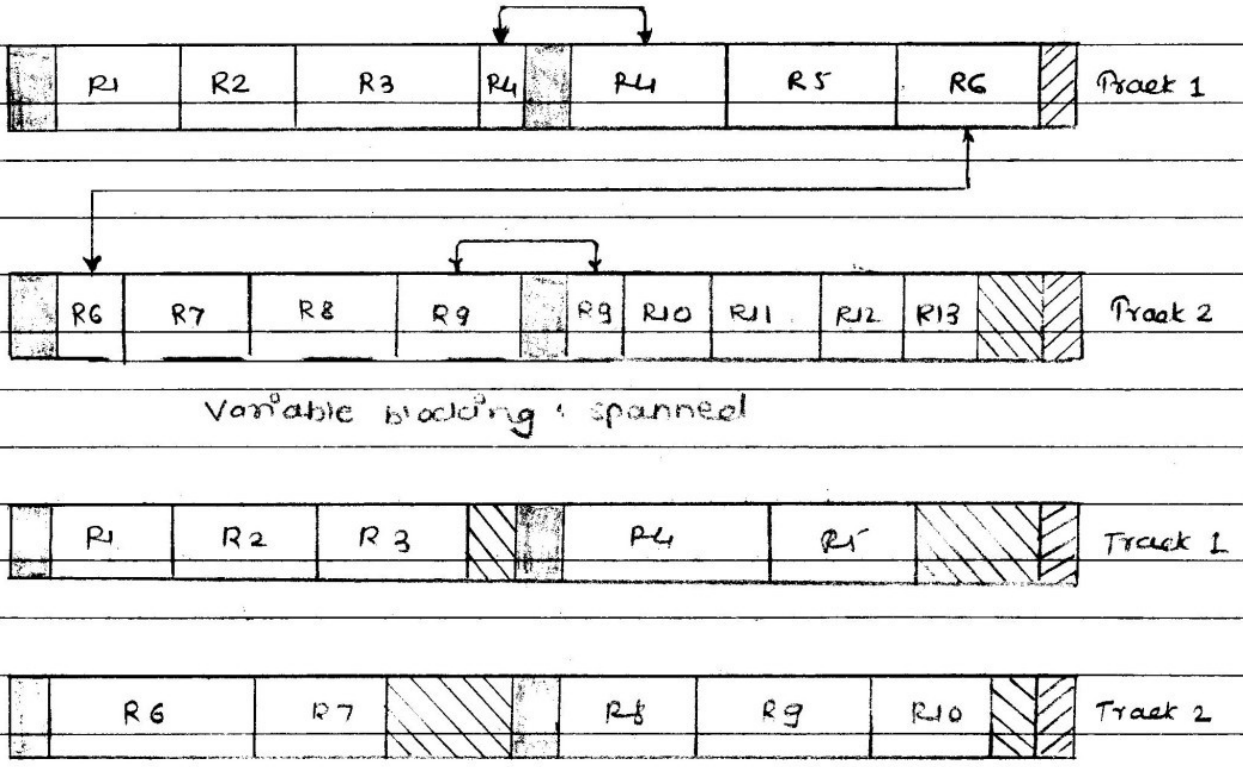


Fig - Record Blocking Methods [MIED87]

* Secondary Storage Management -

On secondary storage, a file consists of a collection of blocks. The operating system or file management system is responsible for allocating blocks to files. This raises two management issues. First, space on secondary storage must be allocated to files and second, it is necessary to keep track of the space available for allocation. We will see that these two tasks are related; that is, the approach taken for free space management. Further, we will see that there is an interaction between file structure

* File Allocation –

Several issues are involved in file allocation:

1] When a new file is created, is the maximum space required for the file allocated at once?

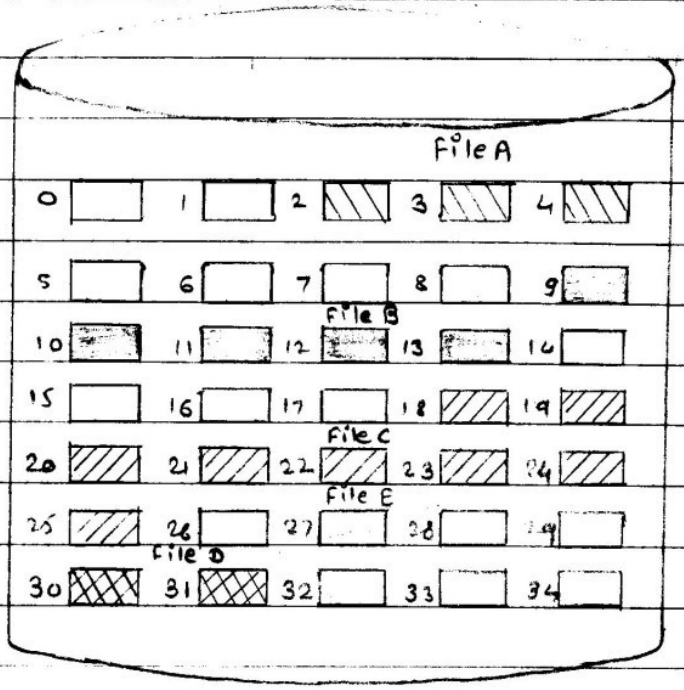
2] Space is allocated to a file as one or more contiguous units, which we shall refer to as portions. That is, a portion is a contiguous set of allocated blocks. The size of a portion can range from a single block to the entire file. What size of portion should be used for file allocation?

3] What sort of data structure or table is used to keep track of the portions asigned to a file? An example of such a structure is a file allocation table (FAT), found on DOS and some other systems.

Table – File Allocation methods –

	Contiguous	Chained	Indexed	
Pre allocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	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 size	one entry	one entry	Large	Medium

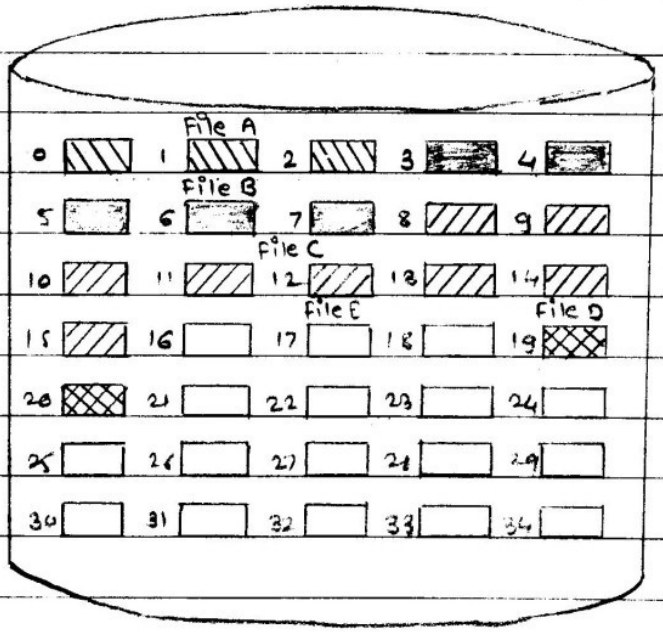
File Allocation Methods –



File Allocation Table

File name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

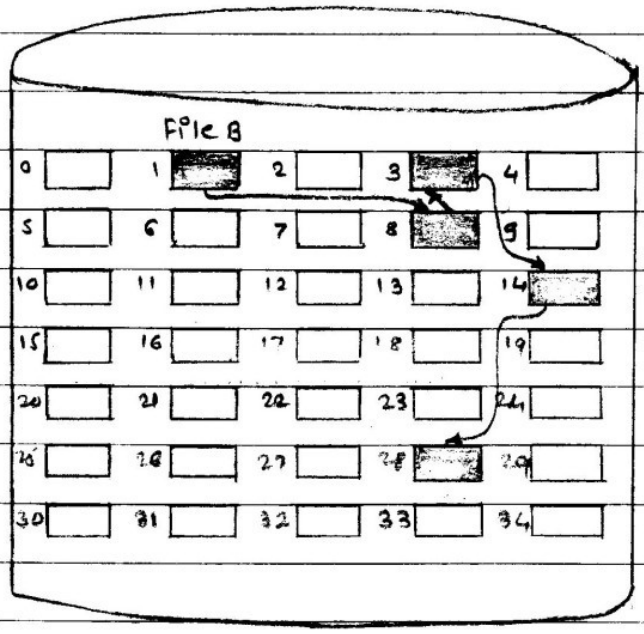
Fig (A) – Contiguous File Allocation



File Allocation Table

File name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

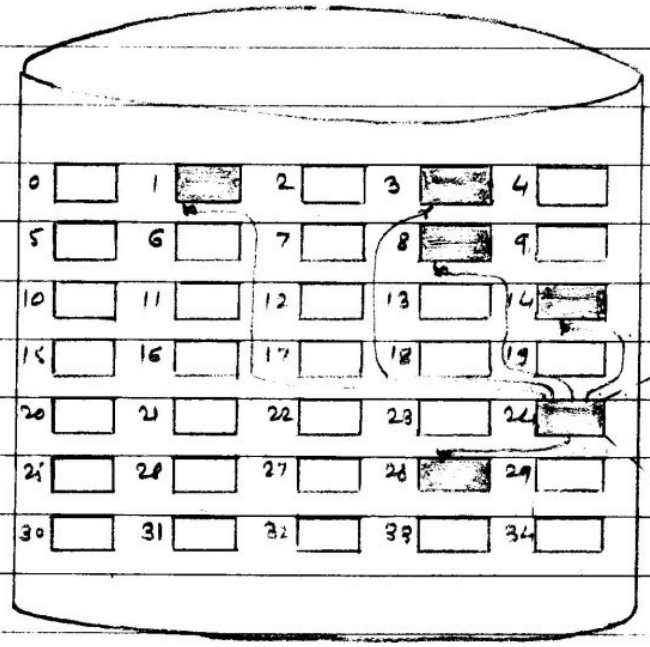
Fig – Contiguous File Allocation (After compaction)



File Allocation Table -

File name	start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Fig – Chained Allocation



File name	Index Block
• • •	• • •
File B	24
• • •	• • •

1
8
9
14
24

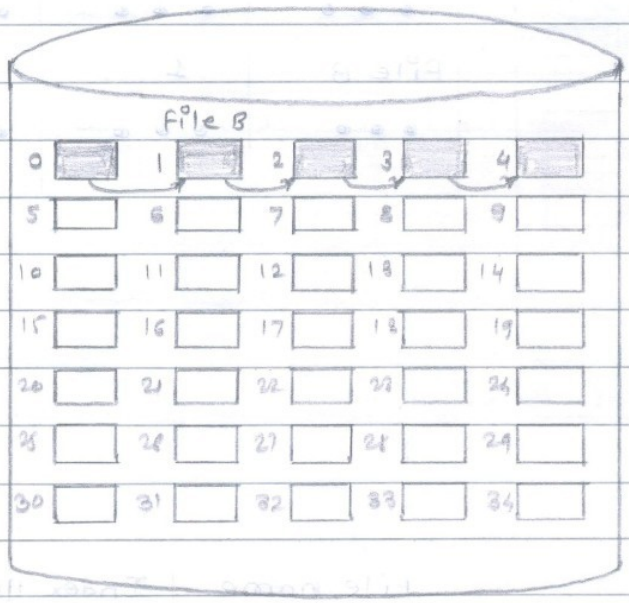
Fig - Indexed Allocation with Block Portions

* Bit Tables –

This method uses a vector containing one bit for each block on the disk. Each entry of a 0 corresponds to a free block, and each 1 corresponds to a block in use. For e.g. for the disk layout of fig. A, a vector of length 35 is needed and would have the following value:

00111000 11111000 11111111111011000

A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks.



File Allocation Table

File name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Fig - Chained Allocation (After Consolidation)

*** Chained Free Portions -**

The free portions may be chained together by using a pointer and length value in each free portion. This method has negligible space overhead because there is no need for a disk allocation table, merely for a pointer to the beginning of the chain & the length of the first portion.

*** Indexing -**

The indexing approach treats free space as a file and uses an index table as described under file allocation.

*** Free Block List -**

In this method, each block is assigned a number sequentially & the list of numbers of all free blocks is maintained in a reserved portion of the disk. Depending

on the size of the disk, either 24 or 32 bits will be needed to store a single block number, so the size of the free block list is 24 or 32 times the size of the corrosponding bit table and thus must be stored on disk rather than in main memory.

* Volumes —

The term volume is used somewhat differently by different operating systems and file management systems, but in essence a volume is a logical disk.

* Linux Virtual File system —

Linux includes a versatile and powerful file handling facility, designed to support a wide variety of file management systems and file structures. The approach taken in Linux is to make use of a virtual file system (VFS), which presents a single, uniform file system interface to user processes. The VFS defines a common file model that is capable of representing any conceivable file system's general feature & behaviour. The VFS assumes that files are objects in a computer's mass storage memory that share basic properties regardless of the target file system or the underlying processor hardware.

VFS is an object-oriented scheme. Because it is written in C, rather than a language that supports object programming (such as C++ or Java), VFS objects are implemented simply as C data structures. Each object contains both data and pointers to file-system-implemented functions that operate on data. The four primary object types in VFS are as follows:

1) Superblock object : Represents a specific mounted file system.

2) Inode object : Represents a specific file.

3) Dentry object : Represents a specific directory entry.

4] File object - Represents an open file associated with a process.

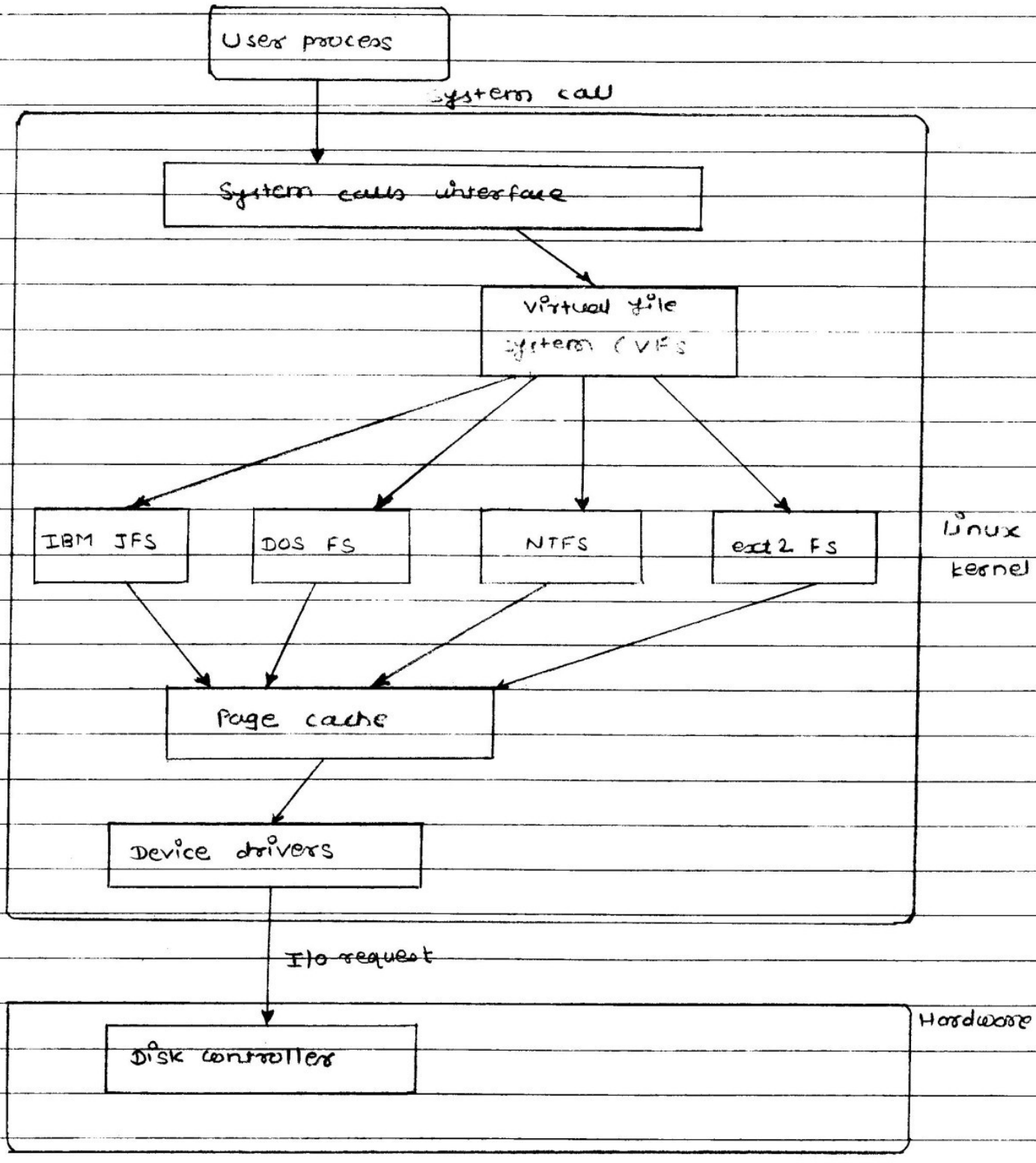


Fig - Linux Virtual File System Context

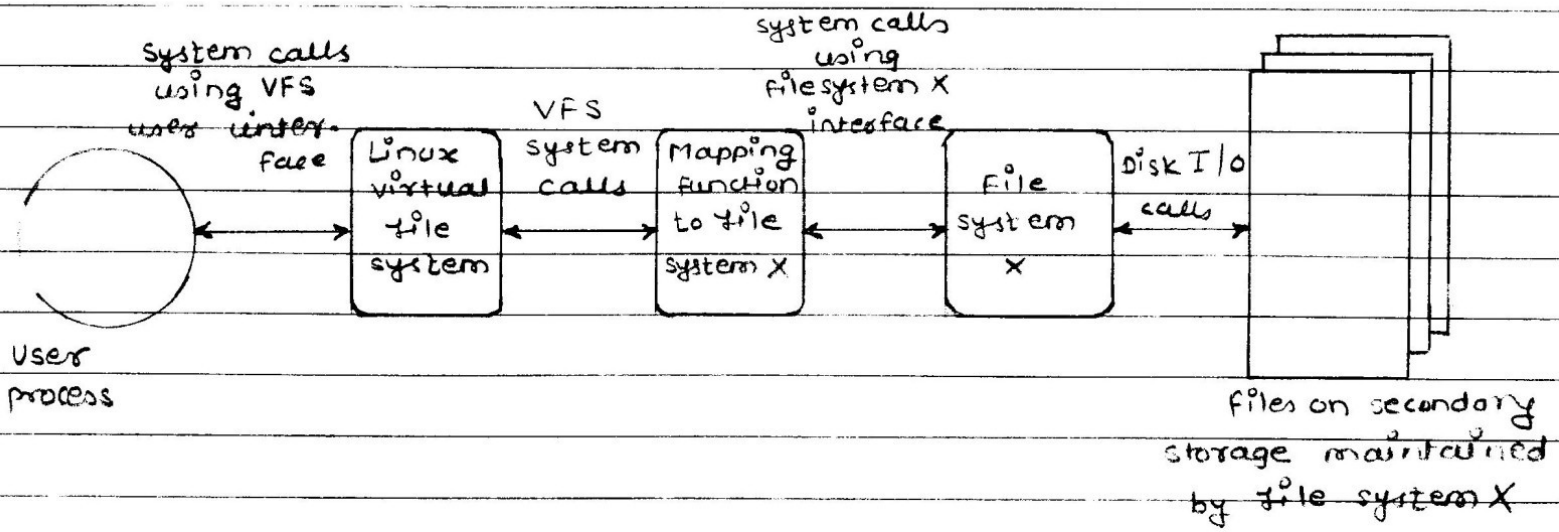


Fig – Linux Virtual File System Concept

END