File Structures

18IS61

Module-1 Notes

Introduction to the Design and Specification of File Structures.

Definition:

- A file structures is the organization of data in files, and files in secondary storage and operations for accessing the data.
- A file structure allows application to read, write and modify data. It might also support finding the data that matches some search criteria or reading through the data in some particular order.

The heart of the file structures.

→ Disks are slow compared to RAM or ROM present in the system. On the other hand they provide enormous capacity to hold data at much lesser cost. They are non-volatile and preserve the information even when the system is turned off.

→ The main driving force behind the file structure design is relatively slow access time of disk and its enormous, non-volatile capacity.

→ Good file structures ~~degn~~ design will give us accen to all the capacity of disk

→ A file structure is the representation of organizing the data on the secondary memory in a particular fashion, so that retrieving data becomes easy.

→ An improvement in file structures design may make an application hundreds of times faster.

A short History of file structure design

General Goals.

→ Get the information we need with one access to the disk.

→ If that's not possible, then get information with as few accesses as possible

→ Group information so that we are likely to get everything we need with only one trip to the disk.

I Early Work

→ Early work assumed that files were on tape.

→ Access was sequential and the cost of access grew in direct proportion to the size of the file.

II The emergence of Disks and Indexes

→ Disks allowed for direct access

→ Indexes made it possible to keep a list of keys and pointers in a small file that could be searched very quickly.

→ With the key and pointer, the user had direct access to the large, primary file.

III The emergence of Tree Structures.

→ The idea of using tree structures to manage the index emerged in the early 60's.

→ As indexes also have a sequential flavour, when they grew too much, they also became difficult to manage.

→ However, trees can grow very unevenly as records are added and deleted, resulting in long searches requiring many disk accesses to find a record.

IV Balanced Trees

→ In 1963, researchers came up with the idea of AVL trees for data in memory.

→ In the 1970's came the idea of B-Trees which require an $O(\log_k N)$ access time where N is the number of entries in the file and k, th number of entries indexed in a single block of the B-Tree structure.

V Hash Tables

→ Retrieving entries in 3 or 4 accesses is good, but it does not reach the goal of accessing data with a single request.

→ Hashing was a good way to reach this goal with files that do not change size greatly over time.

→ Extendible Dynamic Hashing - one or atmost two disk to access the file no matter how big it becomes.

Fundamental File Operations.

* Physical Files and logical Files

Physical file	logical file.
→ Resides on secondary memory.	→ Resides on primary memory.
→ Change are not done directly by the application	→ Change are done directly to logical file by the application
→ Change are updated when file is saved.	→ Change are updated when application changes the values
→ Can be accessed by any no. of applicants.	→ Can be accessed by only application which handle it.
→ Secondary Memory	→ Primary memory.

* Opening files.

Opening a file can be done in 2 ways.

1. Opening a existing file
2. Create a new file.

Opening - A read/write pointers are positioned at the beginning of the file & are ready to read/write.

Create - A new file opens the file as it doesn't have any contents.

Syntax of open function.

fd = open (filename, flags [, pmode]);

where, fd - is the file descriptor

, filename - type char*, physical filename.

flags - controls the operation of the open function.

Different flags.

O_APPEND, O_CREAT, O_EXCL, O_RDONLY, O_TRUNC, O_RDWR, O_WRONLY

pmode - it is protection mode - specifies - read, write & execute permission for owner, & group also others.

Ex: fd = open (filename, O_RDWR|O_CREAT, 0751);

* Closing files:

Files are usually closed automatically by the operating system when a program terminates normally. The execution of close statement within a program is needed to protect it against data loss.

"closecs" function is used to close the file.

* Reading and Writing.

Reading and writing are the fundamental file processing operations, a read function call requires three pieces of information.

Syntax of Read function.

Read (source_file, Destination_addr, size) - read from file.

Syntax of write function

write (distination_file, source_addr, size) - write to file

File handling.

1) Using standard C function defined in header file stdio.h - C streams
2) Using stream classes defined in iostream.h & fstream.h - C++ streams

Seeking:

the action of moving directly to a certain position in a file is called seeking.

seek() function requires atleast 2 pieces of information

seek (source-file, offset)

source-file : is the logical filename in which seek operation will occur.

offset : no. of positions to be moved from the start of the file.

Eg: seek (data, 370);

Seeking using C stream.

pos = fseek (file, byte_offset, origin)

where, pos - moved position value of read/write pointer after fseek() function.

file - file descriptor of the file.

byte_offset - the no. of bytes to be moved from origin in file.

origin - value that specifies the starting position from which the byte-offset must be taken

3 values.

SEEK_SET (0), SEEK_CUR (1), SEEK_END (2).

↓ Beginning ↓ Current position ↓ end of file.

Eg: pos = fseek (data, 370L, 0)

Using C++ stream

Syntax: file.seekg (byte_offset, origin);

file.seekp (byte-offset, origin);

Special Characters.

By creating the file structures we may encounter some difficulty with extra, unexpected characters.

Examples for encounter.

→ On computers we find Control-z (ASCII value of 26)

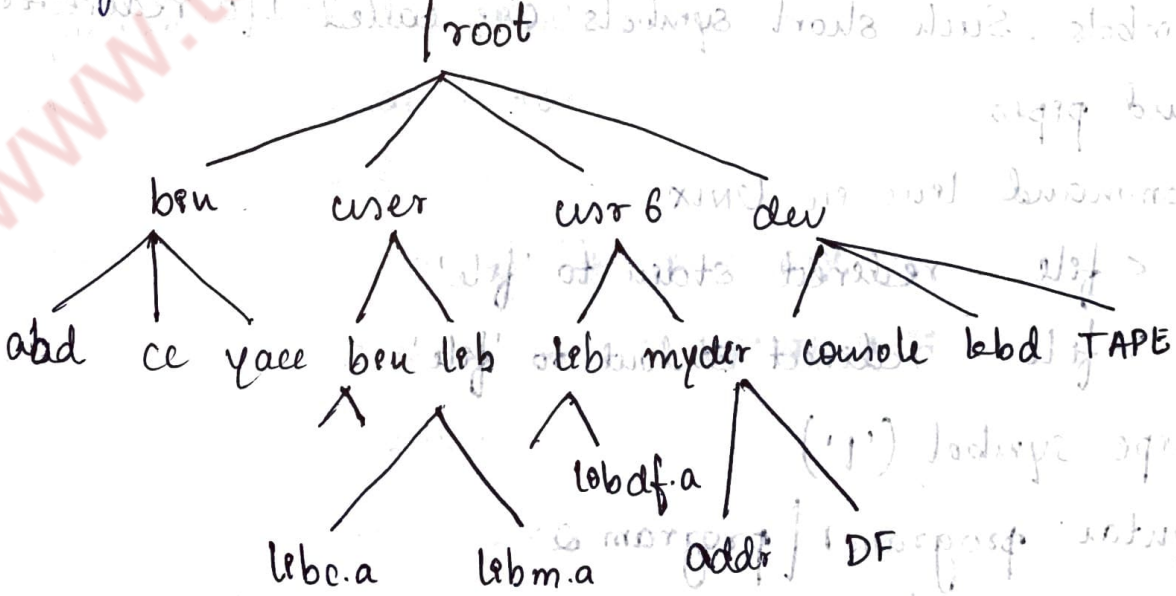
→ End-to-end line in a text file is as a pair of characters consisting of a Carriage return (CR: ASCII value of 13) and a line feed (LF: ASCII value of 10)

→ Users of larger systems, such as VMS, may find that they have just the opposite problem.

The UNIX Directory Structure:

There are hundreds or thousands of files in a computer. To have a convenient access to these files, they are organized in a simple way. In UNIX it is called file system.

It is a tree-structured organization of directories with root of the tree.



UNIX contains only 2 kinds of file.

① Regular file

② Directory, containing files or directories.

Physical devices & logical files:

Physical devices like keyboard, monitor, hard disk etc. Directories and logical files everything considered as files.

Data can be read from a file using the file descriptor and then displayed on the console.

```
file = fopen ("newfile.c", "r");
while (fread (&ch, 1, 1, file) != 0)
        fwrite (&ch, 1, 1 stdout);
```

I/O Redirection pipes:

the output of a program can be sent to a regular file rather than to stdout or the output can be sent as the input to other program, by using some short symbols. Such short symbols are called I/O redirection and pipes.

Command line in UNIX

< file - redirect stdin to 'file'

> file - redirect stdout to 'file'

pipe symbol ('|')

Syntax: program 1 | program 2

the result of program 1 is sent as input to program 2.

File-Related header files:

The special values like end-of-file (EOF) access permissions while opening a file etc are defined in particular header files.

In UNIX headers files are present in /usr/include file.

Header files required for file handling problems are -

iostream.h, fstream.h, fcntl.h, file.h.

C++ streams - iostream.h & fstream.h

UNIX operation - fcntl.h and file.h

Flags - O_RDONLY, O_WRONLY & O_RDWR found in file.h.

UNIX file system commands:

UNIX provides many commands for manipulating files.

Some of them are listed below.

-> cat filename - print the contents of the named file

-> tail filename - print the last ten lines of text file.

-> cp f1 f2 - copy the contents of f1 to f2

-> mv f1 f2 - move (rename) f1 to f2

-> rm filename - remove (delete) file

-> chmod mode f1 - change the protection mode on the named file.

-> ls - list the contents of the directory

-> mkdir name - create a new directory with the name given.

-> rmdir - remove the named directory.

Secondary storage and system software.

Disks:

Organization of desks

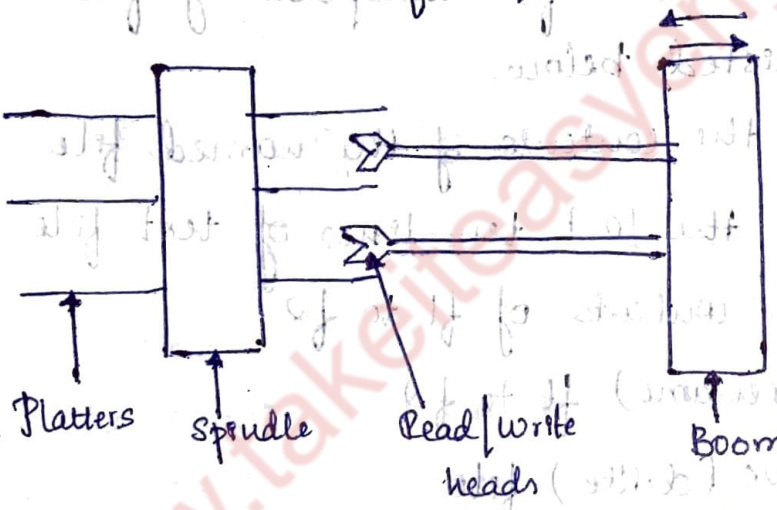
→ Magnetic desks come in many forms, like hard desk, floppy desk, removable desk.

→ Hard disks offers high capacity and low cost per bit.

→ The information stored on a disk is stored on the surface of one or more platters.

→ The information stored in successive tracks on the surface of the desk.

→ The read/write head assembly is used to read & write the data from/to the desk.



★ Schematic illustration of disk drive.

Surface of desk:

→ Each track is often divided into a number of sectors.

→ A sector is the smallest addressable portion of desk.

→ When a read statement calls for a particular byte from a desk file, the OS finds the correct surface, track and sector. OS copies the entire content of the sector into a special area in memory called the buffer.

→ A desk drive has a number of platters. The corresponding tracks that are below and above one another of different platters from a cylinder.

→ The information of cylinder can be accessed without moving read/write head. Moving the arm holding R/W head is called seeking.

→ Each platter has 2 surface, so the no. of tracks/cylinder is twice the no. of platters.

→ No. of cylinders is same as the no. of tracks on the desk surface & each track has the same capacity.

→ Capacity of desk = no. of cylinders, no. of tracks per cylinder & capacity of the track.

Cylinder - group of tracks

Track - group of sectors

Sector - group of bytes.

So, Track capacity = no. of sectors/track * bytes/sector.

Cylinder capacity = no. of tracks/cylinder * track capacity

Drive capacity = no. of cylinders * cylinder capacity.

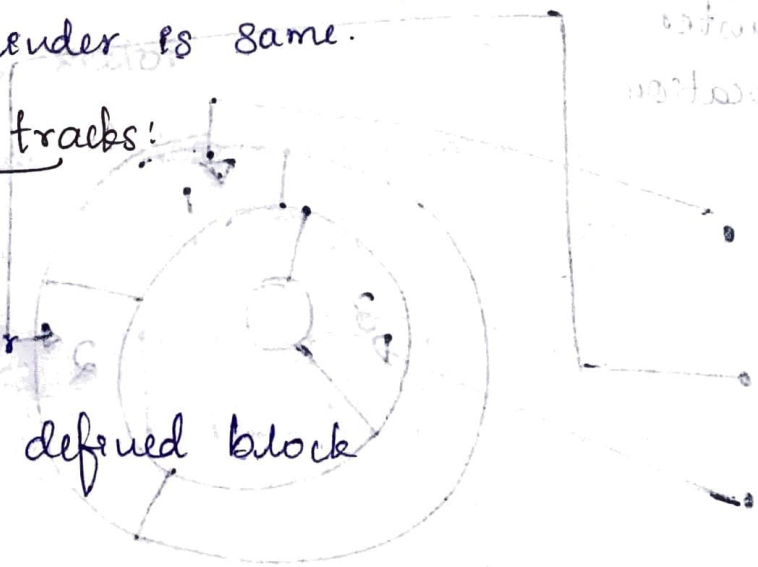
Size of each cylinder is different, the capacity of each cylinder is same.

Organizing tracks:

2 ways.

① By sector

② By user defined block



The physical placement of sector.

→ The sectors can be organized on a track in several track. The simplest among them is that sectors are adjacent, fixed-sized segments of a track that hold a file. This is a perfect way to view a file, but not good way to store sectors physically.

→ When we read the data of a file continuously, which are stored in adjacent sectors one after the other, reading cannot be done easily. To read the next sector, the disk must be revolved once. Hence only one sector can be read per revolution of the disk.

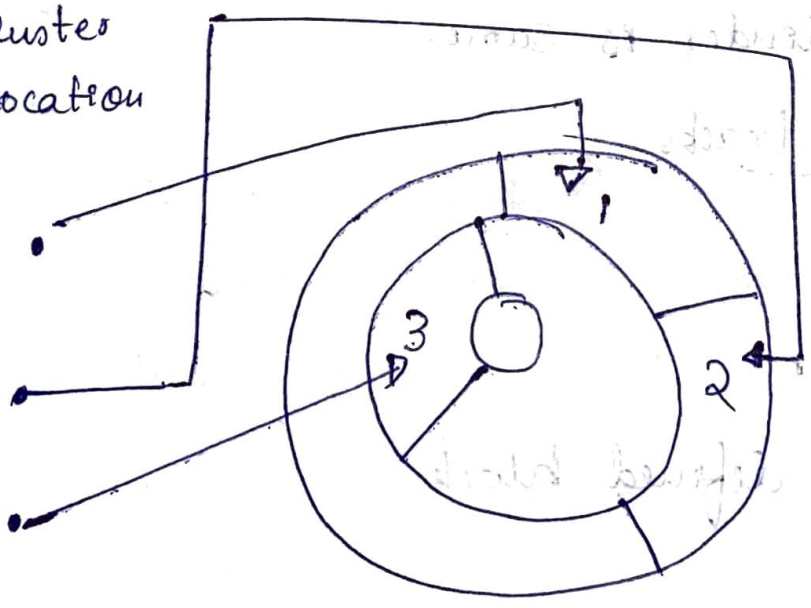
Clusters:

→ Cluster is a fixed no. of contiguous sector. Once a given cluster has been found on a disk, all sector in that cluster can be accessed without requiring an additional seek.

→ A part of the OS called the file manager creates a file allocation table (FAT) - list of clusters

FAT

Cluster no.	Cluster location
1	
2	
3	



Extents:

→ These are the parts of a file which are stored in contiguous clusters so that the no. of seek to access a file reduces.

→ It is preferable to store the entire file in one extent. It is not possible because of less storage space, errors. So the file divided into 2 or more extents.

→ As the no. of extents in a file increase, the file becomes more spread out on the disk & the amount of seeking required to process the file increases.

Fragmentation:

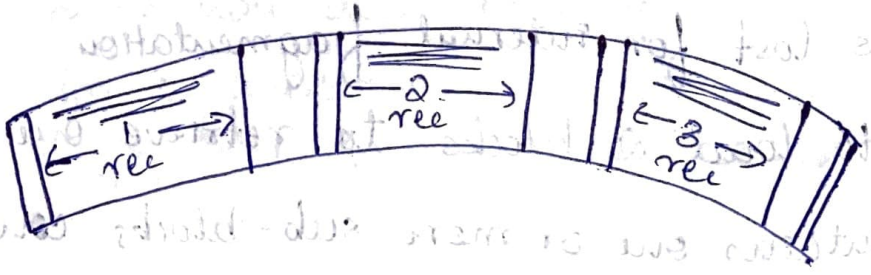
→ Since unit of disk is one sector, the data is stored in terms of a multiple sectors & new file is started in a new sector, this leads to unused (waste) disk space resulting in fragmentation.

2 ways:

① One record per sector:

Advantage: records can be retrieved by retrieving just one sector.

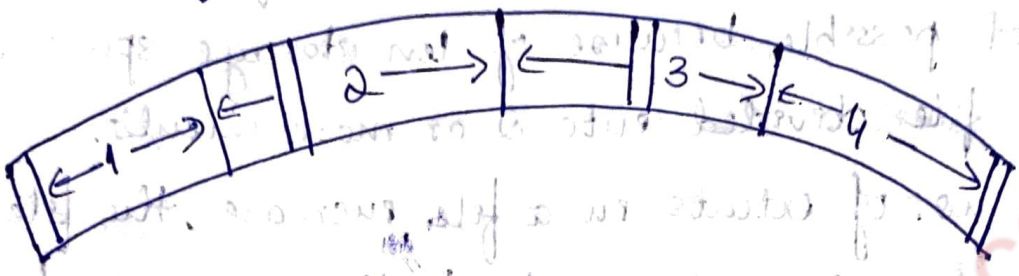
Disadvantage: It leaves enormous amount of unused space within each sector. - Internal fragmentation.



② Records spanned over sectors:

Advantage: no loss of space due to internal fragmentation.

Disadvantage: some records can be accessed only by accessing 2 sectors



Organizing tracks by block:

→ Sometimes disk tracks are devided into integral no. of user-defined blocks, where size can vary.

→ As the blocks are of varying size, (to fit the data) there is no sector spanning and fragmentation problems

→ The term blocking factor is used to indicate the no. of records that can be stored in each block.

1111.....1111	22.....22	333.....33	4444
block 1	block 2	block 3	block 4

If the file is with record size 300 bytes, then the block size will be multiple of 300 bytes.

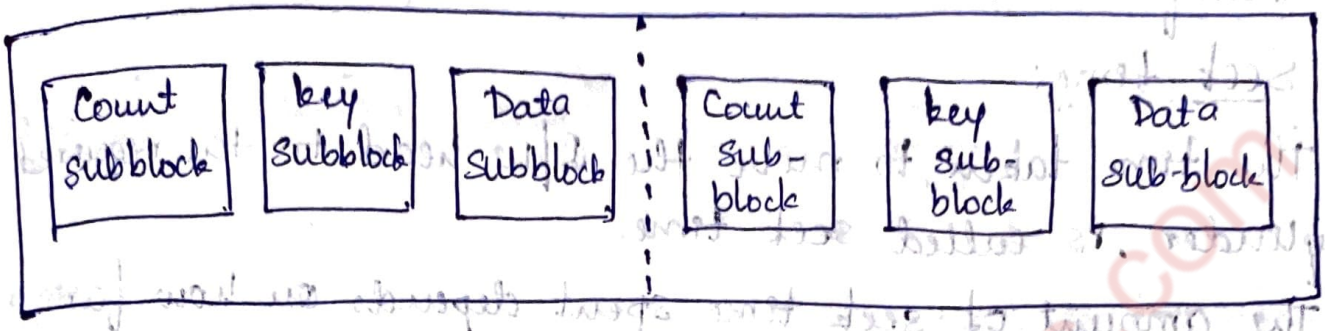
Advantages:

- No space is lost for internal fragmentation
- No need to load 2 blocks to retrieve one record

Each block contains one or more sub-blocks containing extra information about the data block.

The subblock may be:

- Count sub-block :- contains no. of bytes in the data block
- key sub-block :- key for the last record in the data block.



Non-data overhead:

→ Both blocks and sectors organization of track, certain amount of space is taken up from the disk to store some information about storage data, this is called non-data overhead.

→ On sector-addressable disks, non-data overhead contains information such as sector address, block address, sector condition, synchronization marks between fields of info etc. This non-data overhead is of no concern to the programmer.

→ On a block-addressable disk, some non-data overhead is considered by the programmer. Since sub-block and inter-block gaps have to be provided in every block, there is more non-data overhead with blocks than with sector organization.

Note: If blocking factor is large, more no. of records can be stored. But larger blocking factor may lead to fragmentation within a track.

The cost of disk access:

To access a file from the disk, the total amount of time needed can be devided based on 3 operations. The time needed is seek time + rotational delay + transfer time.

(a) Seek time:

→ The time taken to move the R/W head to the required cylinder, is called seek time.

→ The amount of seek time spent depends on how far the R/W head has to move.

→ Most hard disks available today have an average seek time of less than 10 milliseconds.

→ Since the seek time required for each file operation various, usually the average seek is determined.

(b) Rotational delay:

→ The time taken for a disk to rotate and bring the required sector under read/write head is called rotational delay.

→ On average, this is considered to be half of the time taken for one rotation. If a hard disk rotates at 5000rpm, then it takes 12msec to complete one rotation, so rotational delay is 6 msec.

(c) Transfer time:

The time required to transfer one byte of data from track to R/W head or vice versa, is called Transfer time.

the transfer time is given by

Transfer time = (no. of bytes transferred ÷ no. of bytes on a track) * rotation time.

where, rotation time is time taken for one rotation

Data transfer speed = no. of bytes transferred / transfer time.

Transfer time (in sectors) = $\frac{\text{no. of sectors transferred}}{\text{no. of sectors / track}}$ x rotation time.

Note:

① Sequential access of file takes less time to transfer data when compared to random access.

② Seek time is more in random access.

Disk as Bottleneck

→ The disk transfer time is much slower when compared to the network & computer CPU. So the network or CPU has to wait for long time for the disk to transmit data.

→ A no. of techniques are used to solve this problem. One among those is multiprogramming, in which the CPU works on other jobs while waiting for the data to arrive from the disk.

→ Another technique is striping. Disk striping involves splitting the parts of a file on several different drives. This improves the throughput of the disk.

→ Another approach to solve the disk bottleneck is to avoid accessing the disk.

→ 2 ways of using memory instead of secondary storage are memory disks & disk caches.

→ A RAM disk is a large part of memory configured to simulate the behavior of mechanical disk, other than speed & validity.

→ the data in RAM disk can be accessed must faster than disks i.e., without a seek or relational delay.

→ A Disk cache is a large block of memory configured to contain pages of data from a disk when data is requested by a program, the file manager first looks into the disk cache to check if it contains the page with the requested data.

RAM disks & cache memory are examples of buffering.

Magnetic Tape

→ Magnetic tape is a device that provides no direct (random) accessing facility, but can provide very rapid sequential access to data.

→ Tapes are compact, works in different environmental conditions, are easy to store & transport and are less expensive than disks. Tapes are commonly used as a backup device.

Types of tape system:

There has been tremendous improvement in tape technology in the past years. There are now a variety of tape formats with prices ranging from $150 to $150,000 per tape drive.

Ex: 9-track, Digital linear tape, HP Colorado T8000, storage Tek Redwood

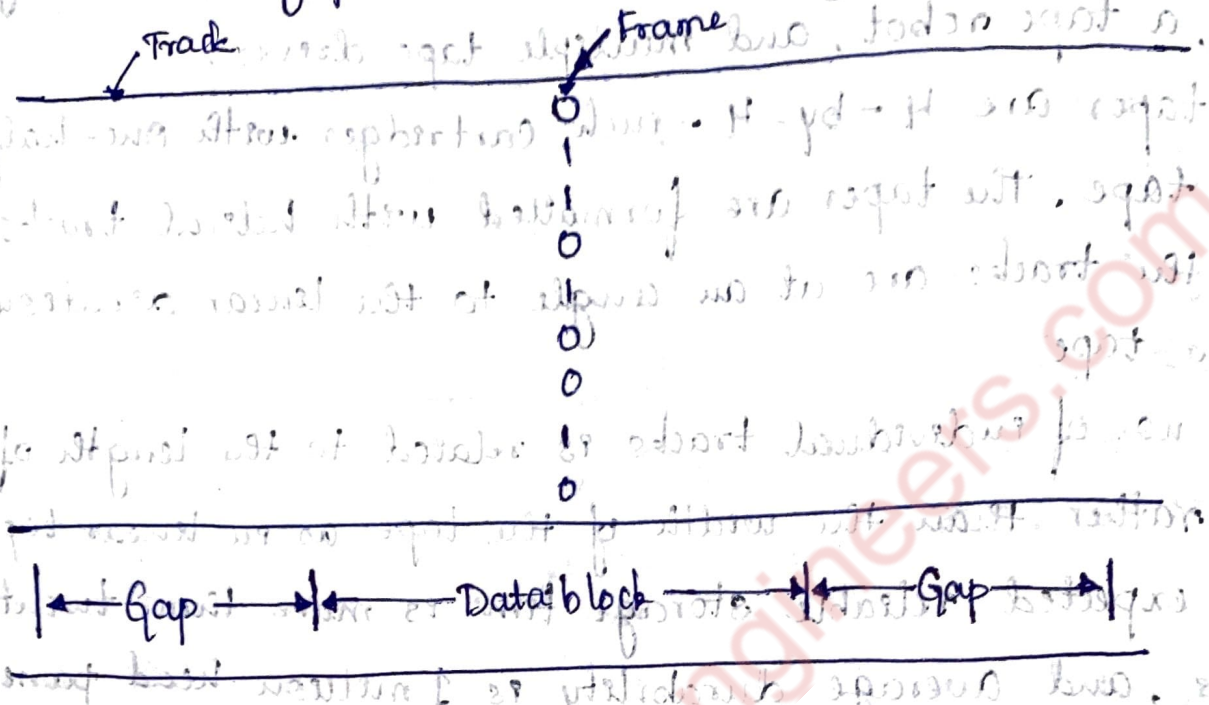
An Example of a High-Performance Tape System.

→ The StorageTek Redwood SD3 is one of the highest-performance tape systems available in 1997.
→ It is usually configured in a silo that contains storage racks, a tape robot, and multiple tape drives.
→ The tapes are 4-by-4-inch cartridges with one-half inch tape. The tapes are formatted with helical tracks. i.e., the tracks are at an angle to the linear direction of the tape.
→ The no. of individual tracks is related to the length of the tape rather than the width of the tape as in linear tapes.
→ The expected reliable storage time is more than twenty years, and average durability is 1 million head panes.

Organization of Data on Nine-Track tapes.

→ On tapes, there is no need of addresses to identify the location of data, as the data can be accessed sequentially only.
→ On tapes, the logical position of a byte with in a file corresponds directly to its physical position from start of the file.
→ The surface of a typical tape is set of parallel tracks, each of which is a sequence of bits. If there are nine tracks, the nine bits, at the corresponding position in the nine tracks are considered as 1 byte, plus a parity bit.
→ The parity bit is not part of data, but is used to check the validity of the data.

→ Frames are grouped into data blocks whose size can vary from a few bytes to many kilobytes, depending on the needs of the user. Blocks are separated by inter block gaps, which does not contain any information.



Nine-track tape

The performance of tape drives is measured in terms of 3 quantities:

(1) Tape density - in bits per inch (bpi) per track, commonly 800, 1600, 6250 or 3000

(2) Tape Speed - in inches per second (ips), commonly 30 to 200 ips.

(3) Size of inter block gap - in inches commonly between 0.3 - 0.75 inches.

The space required to store a file in tape is calculated using the formula,

$$S = n \times (b+g)$$

where, S - space required.
n - No. of data blocks (1 or more record)
b - Physical length of the data block
g - length of an inter block gap.

$$b = \frac{\text{data block size}}{\text{Tape density}}$$

Estimating Data Transmission Times

Transmission rate is affected by tape density and tape speed. Transmission rate can be calculated using;

transmission rate = tape density (bpi) x tape speed (ips)

Effective recording density of tape = $\frac{\text{No. of bytes/block}}{\text{No. of inches req. to store a block (btg)}}$

Disk vs Tape

→ In past, both disk and tapes were used for secondary storage. Disks were proffered for random access and tape for sequential access.

→ Now, disks have over much of secondary storage because of decreased cost of disks & memory storage taps are used as tertiary storage.

CD-ROM:

Introduction

→ CD-ROM is an acronym for compact disk Read-Only memory. A single disk can hold more than 600 MB of data. CD-ROM is read only

i.e, it is publishing medium rather than a data storage & retrieval like magnetic desks.

CD-ROM Strengths

High storage capacity, inexpensive price, durability.

CD-ROM weakness

Extremely slow performance, this makes intelligent file structure difficult.

A short history of CD-ROM

→ CD-ROM is used to store any kind of digital info.

→ CD-ROM is the offspring of videodisc technology develop in late 1960's.

→ The laser vision format support recording in both a constant linear velocity (CLV) format - that maximizes storage capacity and a constant angular velocity (CAV) format - that enables fast seek performances.

→ The Philips & Sony worked on a way to store music on optical disc in digital data format rather than the analog format used earlier.

→ Thus the CD audio appeared in 1984. CD-ROM was built using the same technology of CD audio.

Physical Organization of CD-ROM

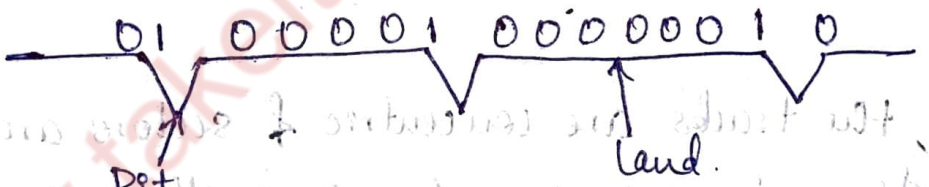
→ CD-ROM is descendant of CD-Audio, in which data was accessed sequentially.

Reading Pits & Lands

→ CD-ROM's are stamped from a glass master disk which has coating that is changed by the laser beam. When the coating is developed, the areas hit by the laser beam turn into pits & the smooth, unchanged areas between the pits are called lands.

→ When the stamped copy of the disk is read, we focus a beam of laser light on the track. The pits scatters the light, but the land reflects back most of the light. This alternating pattern of high & low intensity the original digital info.

→ 1's are represented by the transition from land to pit and back again. 0's are represented by the amount of time b/w transitions. The longer b/w transitions, the more 0's we have.



→ The encoding scheme used is such that, it is not possible to have two adjacent 1's. 1's are always separated by 2 or more 0's.

→ The encoding scheme called EFM encoding (8 to 14 Modulation) which turn original 8-bit of data into 14 expanded bits. - represented as pits & lands on the disk.

Decimal value	Original bits	Translated bits
0	00000000	010010000100000
1	00000001	100000100000000
2	00000010	100100000100000
3	00000011	100100001000000
4	00000100	010000100000000
5	00000101	000010000100000
6	00000110	000100001000000
7	00000111	001001000000000
8	00001000	010010010000000

EFM encoding table

CLV instead of CAV

→ Data on a CD-ROM is stored on spiral tracks. Each track is divided into sectors & data is R/W sector wise.

→ In CAV, the tracks are concentric & sectors are pie-shaped. It write data less densely in the outer tracks than in the center tracks, as there is equal amount of data in all the sectors. This leads to wasting of storage capacity in outer tracks but have the advantage of being able to spin the disk at the same speed for all positions of read/write head.

→ In CLV, sector towards the center of the disk takes the same amount of space as a sector towards the outer edge of disk. Hence data can be stored in max density in all the sectors.

→ CLV is used in CD-ROM instead of CAV because CLV gives much better storage capacity even through varying speed movements of R/W head may cause some difficulties.

CAV	CLV
→ Disk is divided into pie shaped sector.	→ Disk is divided into sectors of equal physical size.
→ Data is densely packed at center & loosely packed at the ends.	→ Data is densely packed all over the disk.
→ Wastage of disk space at outer edge	→ No wastage of space.
→ Disk spins at a constants speed.	→ Disk spins more slowly when reading at outer edge.
→ Lesser storage space than archetecture	→ Much better storage space.

CD-ROM Strength and weakness.

① Seek Performance: The chief weakness of CD-ROM is the random-access performance. Current magnetic disk has an average random access time of about 30 msec. On CD-ROM this access takes 500 msec.

② Data Transfer Rate: A CD-ROM drive reads seventy sectors or 1540 kB of data/sec. It is about 5 times faster than the transfer rate of floppy disks. The transfer rate is faster when compared to the CD-ROM's seek performance.

(3) Storage capacity: A CD-ROM holds more than 600MB of data. This is a huge amount of memory for storage. Many text databases and document collections stored on CD-ROM use only a fraction memory.

(4) Read-Only Access: CD-ROM is publishing memory, a storage device that cannot be changed after manufacturer develops it. The advantage of this is that the user need not worry about the updating. This simplifies some of the file structures.

(5) Asymmetric writing & Reading: with CD-ROM, we create files are placed on the disk once and access the file content thousands or millions of times.

Storage as a Hierarchy

There are diff types of storage devices of diff speed, capacity & cost. The users can select the device depending on their need.

Types of Bytes	Devices & media	Access time (sec)	Capacity (bytes)	cost (cost/byte)
(primary) Registers, ram desk, memory, desk cache.	Semiconductors	$10^{-9} - 10^{-5}$	$10^{0} - 10^{9}$	$10^{9} - 10^{-3}$
(secondary) -Direct access, -serial	-Magnetic desk -Tape & mass storage	$10^{-3} - 10^{-1}$ $10^{1} - 10^{2}$	$10^{4} - 10^{9}$ $10^{0} - 10^{11}$	$10^{-2} - 10^{-5}$ $10^{-5} - 10^{-7}$
(offline) Archeve & Back	Removable, magnetic desks, optical desks & tapes	$10^{0} - 10^{2}$	$10^{4} - 10^{12}$	$10^{-5} - 10^{-7}$

A journey of a Byte

How a byte is stored from a program,

write (textfile, ch, 1); // write value of ch to hard disk.

calls the OS. The OS invokes the file manager, an OS program which deals with the file-related matter & I/O devices.

The file manager does the following tasks when write operation is requested.

→ Checks whether operation requested write is permitted.

→ Locates the physical location where the bytes has to be stored (i.e, locate drive, cylinder, track & sector).

→ Finds out whether the sector to store the character ('ch') is already in memory, if not call I/O Buffer.

→ Puts 'ch' in the buffer.

→ keeps the sector in memory to see if more operations are to be done in the same sector.

The I/O Buffer

To R/W data to a text file, stored in secondary storage, the sector where the data has to be stored must be in the buffer in primary memory. If such sector is not found in the buffer, the file manager finds an I/O Buffer space and reads the sector from the disk.

User's Program	File I/O System
Write (textFile, ch, 1)	If necessary, load the sector from next # the system output buffer. Move 'P' into the buffer.

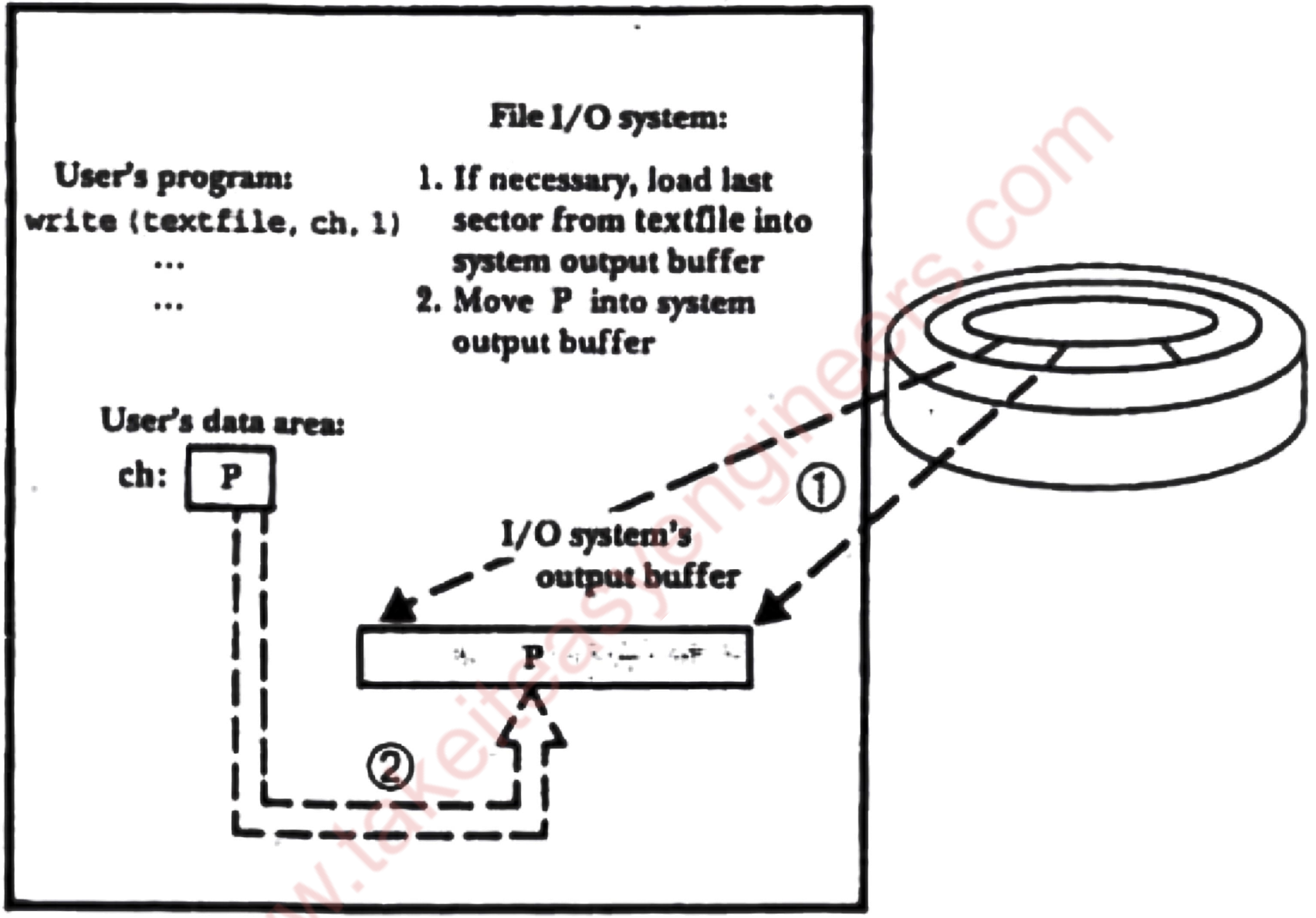


Figure 3.20 The file manager moves *P* from the program's data area to a system output buffer where it may join other bytes headed for the same place on the disk. If necessary, the file manager may have to load the corresponding sector from the disk into the system output buffer.

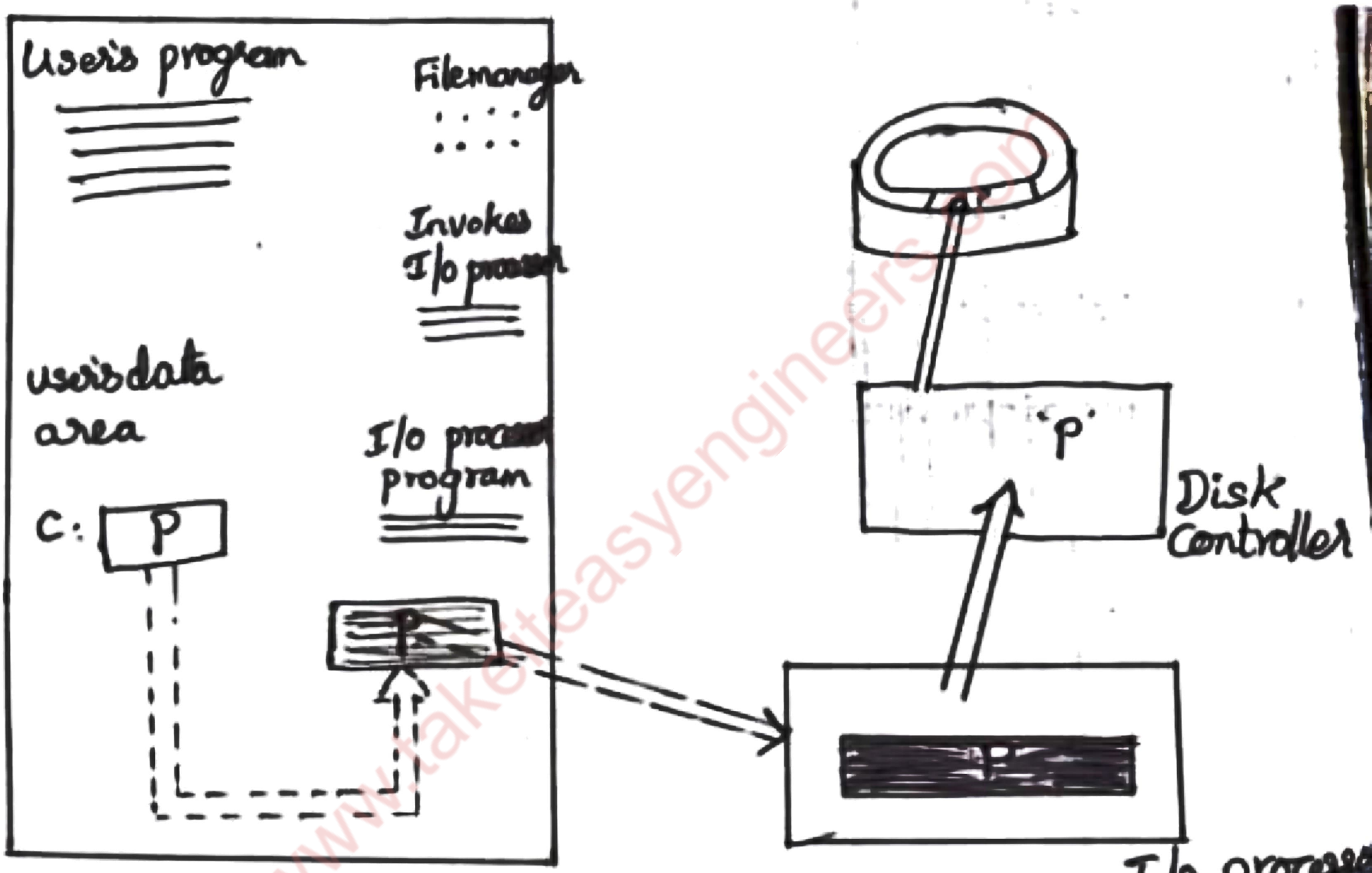


Fig: For write operation, the I/O processor gets the data from the buffer, prepares it for storing on the disk and then sends it to the disk controller, which deposits it on the surface of the disk.

Logical

1. The program asks the operating system to write the contents of the variable *c* to the next available position in TEXT.

2. The operating system passes the job on to the file manager.

3. The file manager looks up TEXT in a table containing information about it, such as whether the file is open and available for use, what types of access are allowed, if any, and what physical file the logical name TEXT corresponds to.

4. The file manager searches a file allocation table for the physical location of the sector that is to contain the byte.

5. The file manager makes sure that the last sector in the file has been stored in a system I/O buffer in RAM, then deposits the '*P*' into its proper position in the buffer.

6. The file manager gives instructions to the I/O processor about where the byte is stored in RAM and where it needs to be sent on the disk.

7. The I/O processor finds a time when the drive is available to receive the data and puts the data in proper format for the disk. It may also buffer the data to send it out in chunks of the proper size for the disk.

8. The I/O processor sends the data to the disk controller.

9. The controller instructs the drive to move the read/write head to the proper track, waits for the desired sector to come under the read/write head, then sends the byte to the drive to be deposited, bit-by-bit, on the surface of the disk.

Physical

9. **The controller instructs the drive to move the read/write head to the proper track, waits for the desired sector to come under the read/write head, then sends the byte to the drive to be deposited, bit-by-bit, on the surface of the disk.**

Physi

Figure 3.19 Layers of procedures involved in transmitting a byte from a program's data area to a file called *textfile* on disk.

The I/O Processor

There is large difference in transmission speed & data path width between the CPU & external devices. The processing disassembling & assembling groups of byte for transmission to & from external devices is done by a special purpose device called I/O processor.

Disk Controller

The job of controlling the operation of the disk is done by disk controller.

→ The I/O processor asks the disk controller if the disk drive is available for waiting.

→ It instructs the disk drive to move its read/write head to the right track & right sector.

→ Disk spins to right location & byte is written.

Buffer management

Uses of Buffer - Buffering involves working with large chunks of data in memory, so that the no. of access to secondary storage can be reduced.

Buffer Bottleneck

→ Assume that the system has a single buffer and is performing input & output on one character at a time alternatively.

→ In this case, the sector containing the character to be read is constantly overwritten by the sector containing spot where the character has to be written and vice versa.

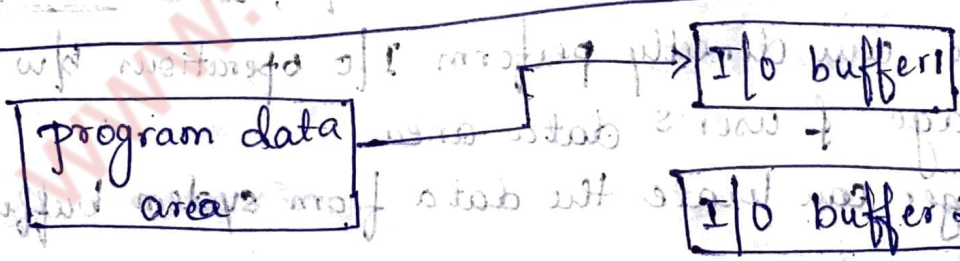
→ In such a case, the system needs more than one buffer.
→ Moving data to & from disk is very slow & programs may become I/O bound. ∴ we need to find better strategies to avoid this problem.

Buffering strategies

→ **Multiple Buffering:**
Reading / writing from a disk is time consuming. To utilize CPU efficiently buffers are maintained. If 2 buffers are used the CPU can be filling one buffer while the contents of the other are being transmitted to disk.

- **Double Buffering:**
When both the tasks are finished, the roles of the buffers can be exchanged. This method of swapping the roles of 2 buffers after each operation is called double buffering.

(a)



(b)

In (a) the contents of system I/O buffer 1 are sent to disk while I/O buffer 2 is being filled & (b) the contents of buffer 2 are sent to disk while I/O buffer 1 is being filled.

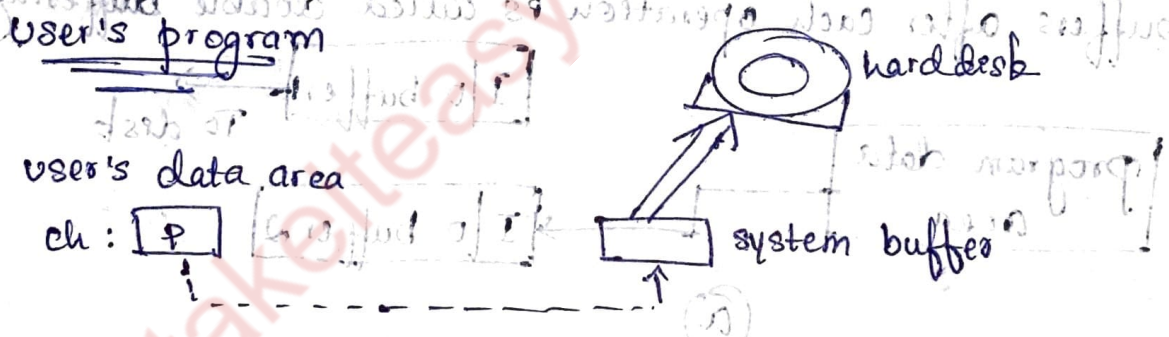
• Buffer pooling.

→ When system buffer is needed, it is taken from a pool of available buffers and used.

→ When the system receives a request to read a certain sector or block, it search to find if any buffer in the block contains that sector or block. If no buffer contains it, the system finds a free buffer from the pool & loads the sector or block into it.

→ Move Mode and Locate Mode.

For a program to R/W contents a file, the data from user's data area has to move to the system buffer (or vice versa). This method of moving chunks of data from one place to another in memory, before accessing data, is called move mode.



The move mode can be avoided in 2 ways, called locate mode

→ The file manager can directly perform I/O operations b/w secondary storage & user's data area.

→ The file manager can locate the data from system buffer using pointers.

→ Scatter / Gather I/O

Each block in memory consists of header followed by data. In order to process the data, the headers of each block & data of each block is moved to 2 separate buffers. The technique of separating the headers & data block called 'Scatter'

The reverse of scatter input is gather output. The several data & header block are arranged separately in gather technique.

I/O in Unix:

→ The process of transmitting data from a program to an external device. The topmost layer deals with data in logical terms.

→ The below layer's carryout the task of turning the logical object into a collection of bits on a physical device. This layer is called kernel

→ The top layer consists of processes, associated with solving some problems using shell commands (like cat, tail, ls etc), User programs that operate on files, and library routines like scanf and so on. Below this layer is the Unix kernel, which consists of all the rest of the layers.

→ In UNIX all the operations below the top layer are independent of applications.

→ When this system call is executed kernel is invoked immediately. This system call instructs the kernel to write a character to a file.

→ The kernel I/o system connects the file descriptor (fd) in program to some file in the file systems. It does this by proceeding through a series of four tables that enables the kernel to find the file in the disk.

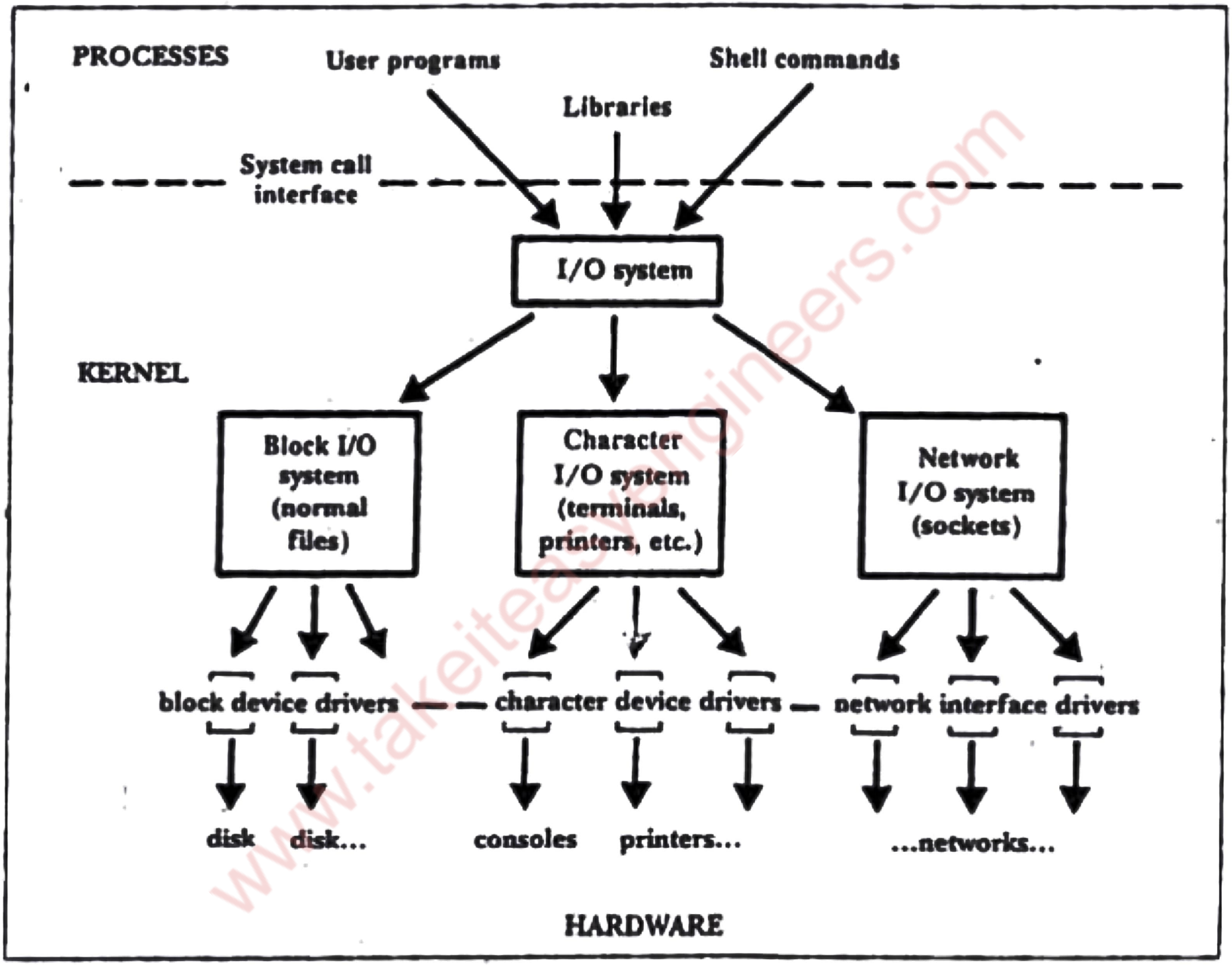


Figure 3.23 Kernel I/O structure.

The four tables are:

① A file descriptor table

② An open file table, write information about open files

③ A table of index nodes, with detailed information about a file.

④ A file allocation table, this is part of I node & contains the address of each cluster.

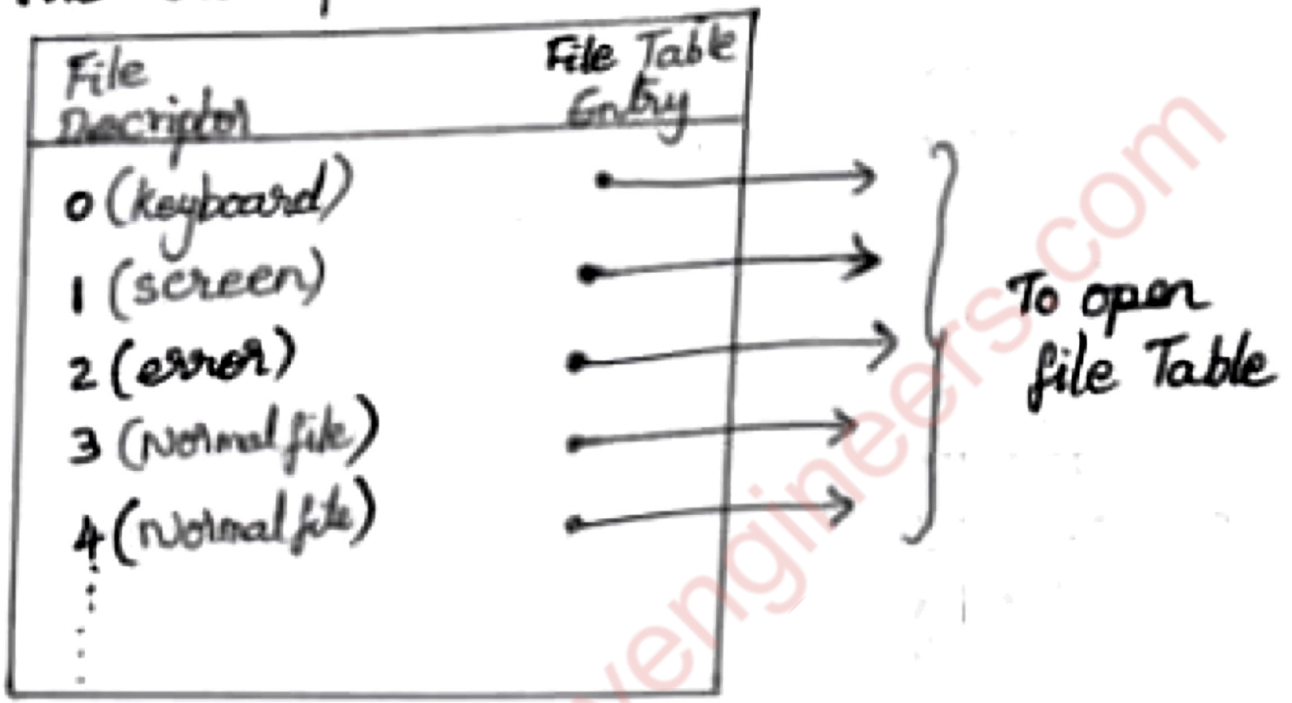
→ The file descriptor table is a simple table that associates each of the file descriptors used by a process with an entry in another table, the open file table.

→ The open file table contains entries for every open file: Every time a file is opened or created, a new entry is added to the open file table. Entries called file structures.

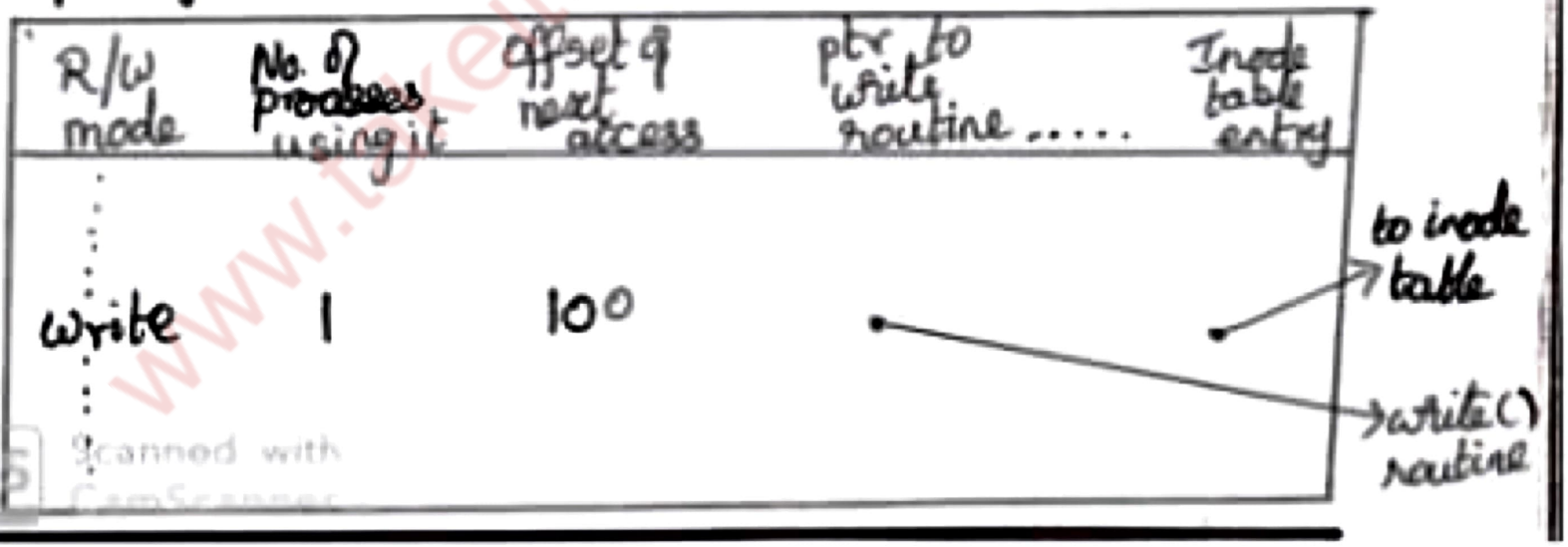
→ More information about the file is present in inode table. The files inode table will be kept on the disk with the file. It contains info like permission, owner's id, file size, no. of blocks used by the file, & [illegible]

→ File Allocation table (FAT) is within the inode table it associates the clusters of the file with pointer locating that address of the clusters.

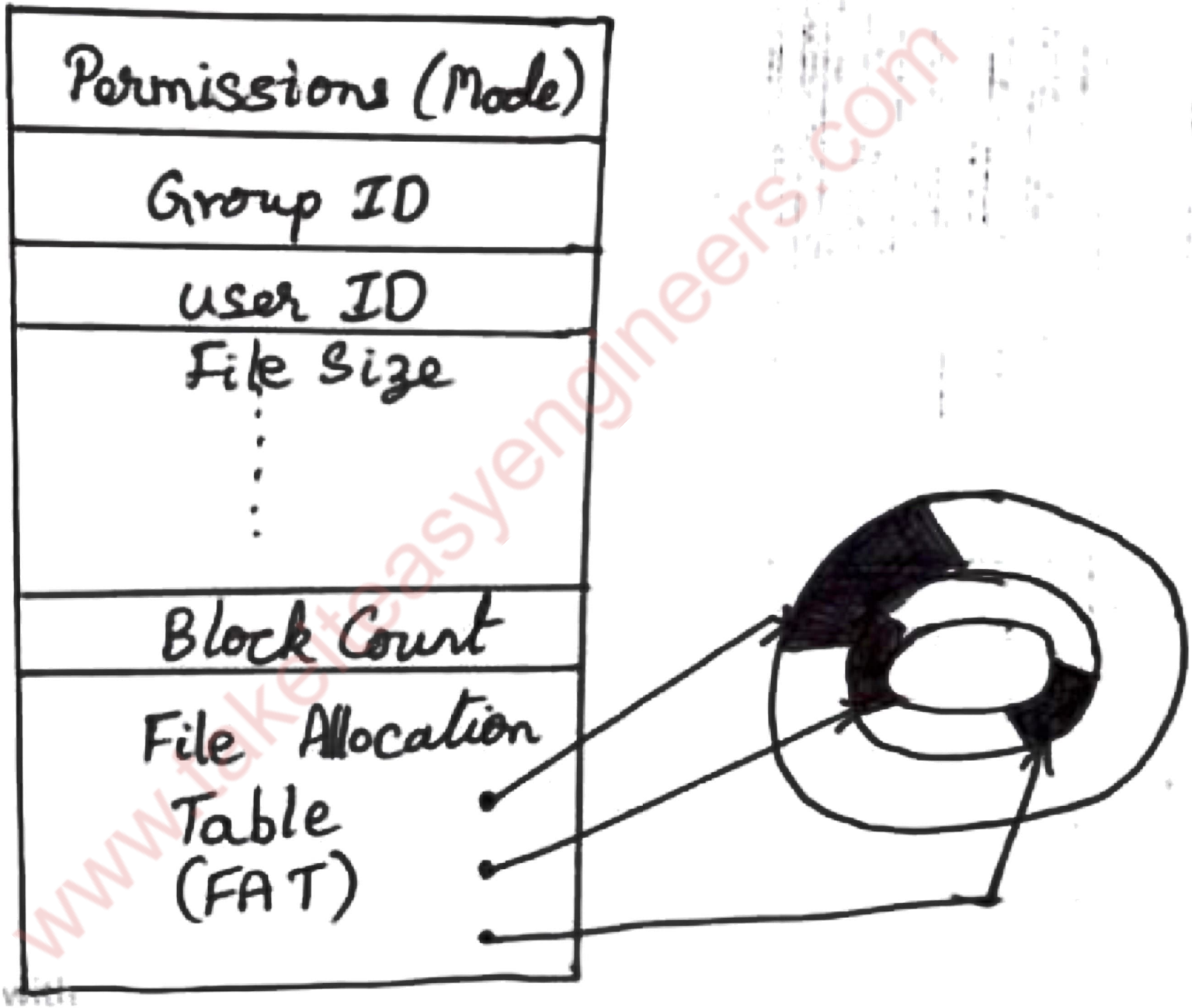
a) File Descripton Table



b) Open file table



An inode — to describe a file.



Linking Filenames to Files

→ All file path starts with a directory. A directory is just a small file that contains many files where each file name is associated with pointer to files inode on disk.

→ It is possible for a file to be saved in different names, in such case, all such filenames point to the same inode and there are many hard links to the same file.

→ A softlink or symbolic link, links a filename to another filename or path rather than pointing to inode.

→ Hence when the original file is deleted, the inode is also deleted & symbolic link becomes dangling.

Types of Files

Normal files - are normal text or program files.

Special files - are files that drive some device, such as line printer or graphics device.... (device drivers)

Socket - are abstractions that serve as end points for inter process communication.

Device Drivers

For every peripheral device, there is a separate set of routines called device drivers. It performs the I/O operations b/w I/o buffer and the devices.

Block I/O:

I/o between a computer & a block device.

Fundamental File Structures Concepts; Managing Files of Records.

Field and Record Organization.

A Stream File

A field is the smallest logically meaningful unit of information in a file.

Operator << is an overloaded function to write the fields to a file as stream of bytes.

Eg: fstream fp1;

fp1.open (filename, ios::out);

fp1<< name << usn ;

Field Structures

4 of the most common methods follows:

Method 1 : Fix the length of fields.

Each variable is stored as an array of characters of fixed length.

Suppose name [10] is declared.

The size of the array must be one larger than the longest string it can hold.

We can define a struct in C or a class in C++ to hold these fixed-length fields. The only diff b/w the C & C++ versions is the use of the keyword struct or class & the designation of the fields of class Person as public in C++.

Eg:

Ames	Mary	123 Maple	Stillwater	OK74075
Mason	Alan	90 Eastgate	Ada	OK74820

Using this kind of fixed-field length structures changes our output so it looks like the above example. Fields are organized by lemeting the max size of each field. This is called as fixed-length field.

Disadvantages:

→ Padding is required to bring the fields up to a fixed length, which makes the file larger.

→ Data might be lost, if it is not able to fit into the allocated space.

The method of structuring is very good solution if the fields are already fixed in length or there is very little variation in field lengths.

Method 2: Begin Each field with a length indicator.

Eg:

04Ames	04Mary	09123 Maple	10Stillwater	0200574075
05Mason	04Alan	1190 Eastgate	03Ada	0200574820

Another way to make it possible to count to end of a field is to store the field length just ahead of the field, as example above. If fields are not too long (< 256 bytes) it is possible to store the length in a single byte at the start of each field. These fields as length-based.

Method 3: Separate the fields with Delemeters.

In this method, the fields are separated by using a special character or sequence of characters.

The special character used to separate the fields is called the delimiter. The selected delimeter to separate the fields should not appear as a field value.
A delemeter can be '|', '#', space, newlew, etc.

Eg:
```
Ames|Mary|123 Maple|Stillwater|OK|74075|
Mason|Alan|90 Eastgate|Ada|OK|74820|
```

Method 4: Use a "keyword = Value" Expression to Identify Fields.

In this method, the keyword and its values are stored for each record. It is the first structures in which a field structures in which a field provides information about itself. It is easy to identify the missing fields

Disadvantage: 50% or more of the files space is occupied by the keywords.

Eg:
```
Last = Ames | first = Mary | address = 123, Maple | city = Stillwater |
state = OK | zip = 74075|
```

Record Structures:

A record is defined as a set of fields that belong together when the field is viewed in terms of a higher level of organization. A record in a file is represented as a structured data object.

The various method used to organize the records of a file:

1. Make records a predictable no of bytes.
2. Make records a predictable no. of fields.
3. Begin each record with a length indicator.

4. Use an index to keep track of addresses.
5. Place a delimiter at the end of each record.

Method 1: Make records a predictable no. of bytes.

A fixed length record file is one in which each record contains the same no. of bytes.

The record size can be determined by adding the max space occupied by each field. This method is called as counting bytes structures.

Eg: Fixed length records with the fixed length fields

AMAR ###### IVACS057 #### 17 ### 54 ## 13

Fixed length records with variable length fields

AMAR|IVACS057|17|54|13| ← unused space →

Method 2: Make records a predictable no. of fields.

In this method the no. of fields in each record is fixed. This method is also called as fixed count structure.

Eg: AMAR|IVACS057|17|54|13|MARY.S|IVAD9IS054|15|54|17....

Method 3: Begin each record with a length indicator.

In this method, every record would begin with an integer, that indicates how many bytes are there in the rest of the record. This is commonly used method for handling variable length records.

Eg: 03 AMAR|IVACS057|17|54|13|27 MARY.S.|IVAD9IS054|15|...

Method 4: Use an index to keep track of addresses:→

An index is used to keep a byte offset for each record. The byte offset allows us to find the beginning of each

Successive record and compute the length of each record. The position of any record can be taken from the index file then seek to the record in the data file.

Eg: Data AMAR|1VAC5057|17|54|13|MARY.S|1VA09CS054|15

Index 00 23

Method 5: Place a delimeter at the end of each record.

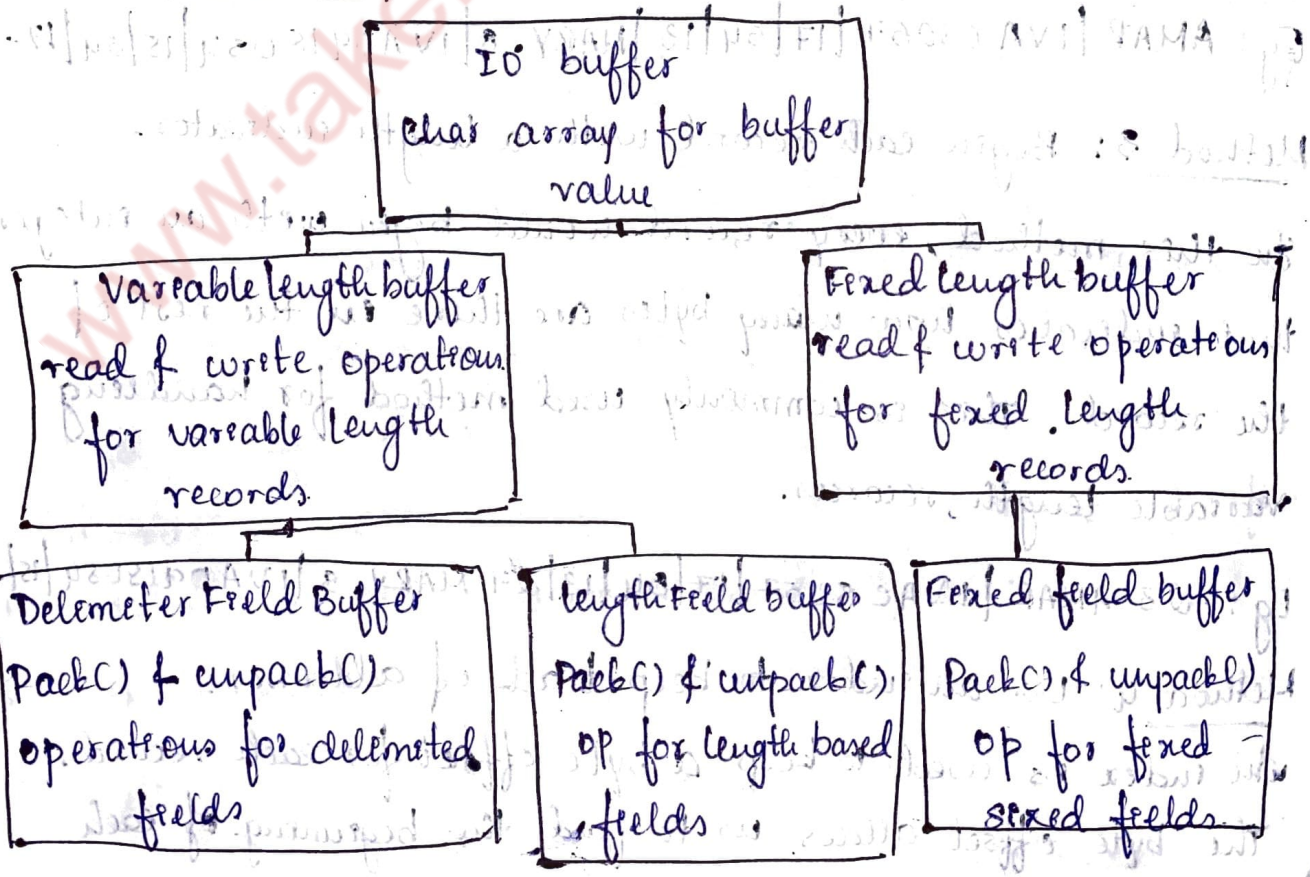
In this method, a special character other than the field delimeter is placed at the end of each record.

Eg: AMAR|1VAC5057|17|54|13|#.....

Using Inheritance for record buffer classes.

Inheritance allows multiple classes to share members and methods. One or more base classes define members & methods, which are then used by subclasses.

Class Hierarchy.



Record Access:

To view the contents of a specific record with a key, the key must be in canonical form. A standard form of representing a key is called the canonical form.

The unique key, which identifies a single record is the primary key. Eg: USN.

secondary key is a key that is common to a group of records. Eg: Semester.

Sequential Search:

It is the technique of searching, where the records are accessed one-by-one and checked till the searching key is found.

The work required to search sequentially for a record in a file with n records is proportional to n.

Record blocking - A collection of records stored as a physically contiguous unit on secondary storage. In this chapter, we use records blocking to improve I/O performance during sequential searching.

Repetitive lists : -Searching for patterns in ASCII files.
-Searching records with a certain secondary key value.

Small search set : Processing files with few records.

Devices/media most hospitable to sequential access : tape, binary file on disk.

UNIX Tools for Sequential Processing:

Some of the UNIX commands which perform sequential access are :

Cat: displays the content of the file sequentially on the console.

% cat filename

Eg: % cat myfile

Ames Mary 123 Maple Stillwater Ok74075
Mason Alan 90 Eastgate Ada Ok74820

wc: counts the no. of words lines and characters in the file

% wc filename

Eg: % wc myfile

2 14 76

grep: (generalized regular expression) used for patter matching

% grep string filename

Eg: % grep Ada myfile

Mason Alan 90 Eastgate Ada Ok74820

Direct Access:

A file accessing mode that involves jumping to the exact location of a record. Direct access to a fixed length record is usually accomplished by using its relative record number (RRN) computing its byte offset and then seeking to the first byte of the record.

Byte offset = record size (r) * required RRN (n)

the byte offset of record of RRN 2 of fixed length record of size 128 bytes is

Byte offset = 128 * 2 = 256

from 256th byte the record of RRN 2 starts.

Header Records:

A header records is often placed at the beginning of the file to hold some general information about a file.

Two methods:

- int readheader(); reads the header and returns
  1 - if header is correct &
  0 - if header is wrong.
- int writeheader(); adds header to the file & returns the no. of bytes in the header.

File Accen and File Organization:

→ File organization is static
  - Fixed length Records
  - Variable length records.

→ File accen is dynamic
  - Sequential Accen
  - Direct Accen.