FILE SYSTEMS

→ File : A coll^n of related information

→ File could be : data, program

numeric text α-numeric

→ In any case → seq. of bits

→ Can be free structure

→ smallest unit of secondary storage from user p.o.v.

→ File System : How to store files & how to retrieve them.

→ Many attributes are associated with any file :

↳ Permission
↳ Name
↳ Type
↳ Size
↳ Location
↳ Time
↳ Date
↳ User
↳ Identifier

→ In file system, ∃ a table (in directory) where name of file is given an identifier. After that, internally, file will be ~~refd~~ referred through that no. → will point to all attributes of that file

has ⇓ all info. about file

→ What can you do with a file ? (Abstract data type)

i) Create : have to find space in file system for this file, update the file directory

2) Read / Write : have to locate the file in file system, then check file ptr & update it (pehle kahaan tk tha, ab kaha jaayega)

* It will keep ptr per process for a file.

Teacher's Signature

→ In some systems, I've to first Open, then Read / Write

3) Delete : Free space allocated & remove entry from directory

4) Reposition in file :

5) Append : write near last ptr. → can add new info & update the ptr.

additional one's { 6) Copy :

7) Rename :

9/4/18

~~128 bit virtual~~

Ex. → 256 KB block

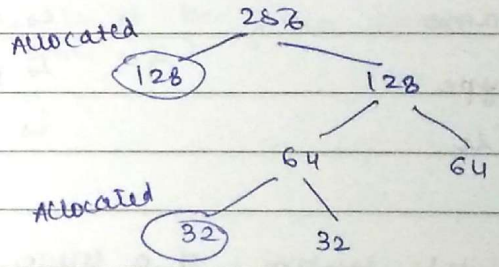
Request 120

Request 30

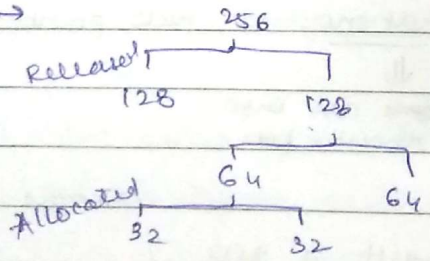
Release ~~Request~~ 120

" ~~30~~ 30

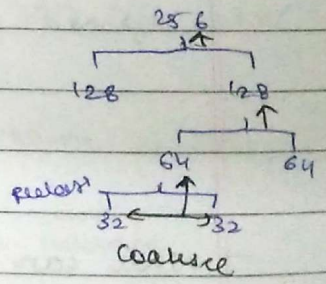
Buddy :

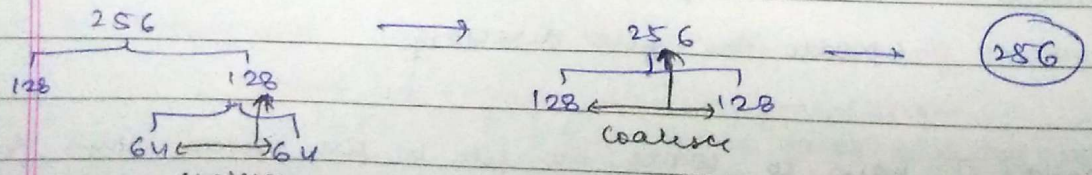


Release 120 : →



Release 30 →





Teacher's Signature

Different kind of files ? : can find type using extension (separated from name by '.')

↳ .exe : executable file

→ OS knows how to deal with executable files.

→ extension can give hint to appl^n which can open it.

→ each type of file should be supported by File system

→ 2 people want to ~~use~~ change file ? :

X clusive Lock : (Write)

Shared Lock : (Read Mode)

Structures :

1.) Text file : String of char, then a new line char, then n — etc..

2) Excel files : Records $\xrightarrow{\text{each naming}}$ fields

3) Src file : A collec^n of func^n's

4) Executable file : binary info.

→ Logical Unit of File (Smallest)

1) Text : 1 char ⇒ 1 Byte

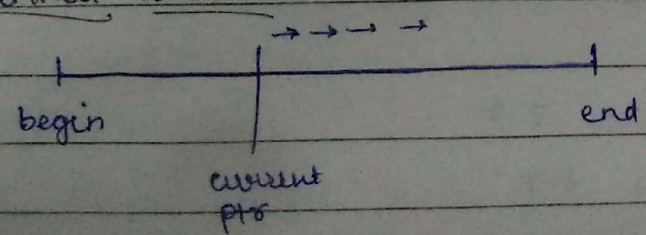
2) Db record : 1 record ⇒ set of fields

↓

Name, ROLL NO (let)

How to access a file ?

1.) Sequential access :



get next logical unit & to the next --- & so on.

Teacher's Signature

There may be possibility that you can jump 'n' units further, or backwards :
good for text files (compiler are working → reading src. file)

2.) Direct Access :
good for databases.

→ Storing Name, ROLL No. ⇒ Total size of each record : 16 B
Name: 10 char (10 B), ROLL No.: (6 B)

↳ It's not 1 record is kept at 1 place on storage disk.

⇒ Min. storage unit in disk : BLOCK.
(Usual size of sector : 512 B)
Let Block size : 512 B ⇒ everytime I access disk, I'll to pick whole 512 B & bring it to buffer

Block size : 1024 B ⇒ pick whole 1024 B & bring it to buffer

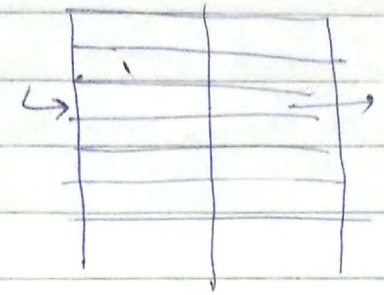
↳ Let Block size : 1024 B. → 2^{10}
No. of records in each block = $\frac{1024}{16}$ → 2^4 = 64

→ NO. of students = 2000
→ NO. of blocks needed = $\frac{2000}{64}$ = 3

→ need to keep info. this — block : this — record (direct access)

↳ If no. of students ↑↑
⇒ NO. of blocks ↑↑
⇒ Possibility that records aren't stored contiguously. Also, keeping track of which record in which block : tough

3.) Indexed : Have to be sorted (either on Name, or Roll No)



: know which record belongs to which block.

11/4/18

→ File System : is a S/w (part of OS)

↳ organizes how files are logically stored.

↳ File System will also be on some device. (may be, disk)

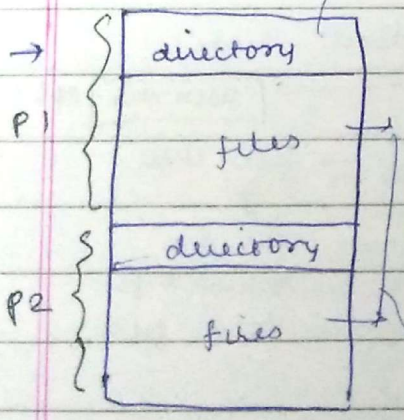
→ maybe 1 disk, or 2

↓ partitioned → install diff. FS on both

↓ install F.S. on both

Volume : device directory / Volume Table of Contents

directory for FS → info. about files



→ Partitioned into 2 parts

Disk I

If both have diff FS : maintain 2 directories

→ Directory : store info. about all files & some attributes of files.

→ Every process opens some file.

Teacher's Signature

Shell . : current dir

.. : parent dir

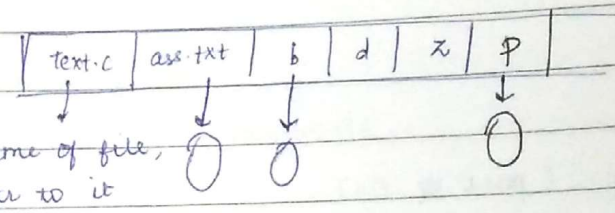
cd .. : go to parent dir

→ When a process given system call, it returns fd table

Structure of Directory

1) Single level Directory: → not an array, only that all info of all files are at same level

info. :



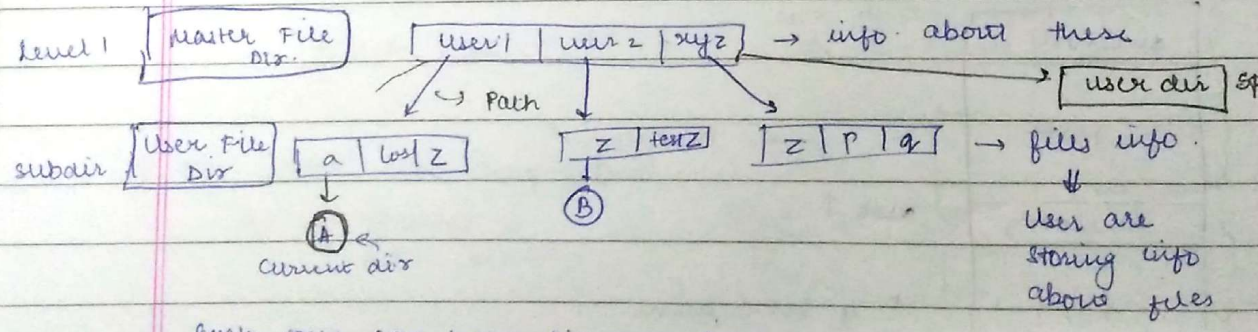
→ If new file is created : add its info

•) Problems : There can be no 2 files with same names.

(2 Users can't use same file with same name).
(Or replacement will occur)

Better option :

2) Two level directory :



Each user can have his own file

Adv → can have same name → diff. Users

↳ User wants more files : it can 'ne all at 2nd level

* These files are absolutely independent (z of 1, z of 2, z of 3)

↳ If User 1 wants to access test2 of 2 (permission granted)

→ has to 1st go to the 1st level than to test2

(current dir → current file contains

Teacher's Signature

A : Object prog for User 1 : needs Compiler
B : ~ ~ User 2 : ~ ~ } → Both need system lib & compiler

copy of ~~it~~ needs to be at both ~~directories~~

when sys. lib is needed : 1st check current lib → then check special user dir ← can create special user dir ← ↑ wastage of mem

3) ~~Tree directory~~ :

→ can have path name → absolute : /usr/bin/a.txt → starting from root
→ relative : a.txt ⇒ relative to (current dir)

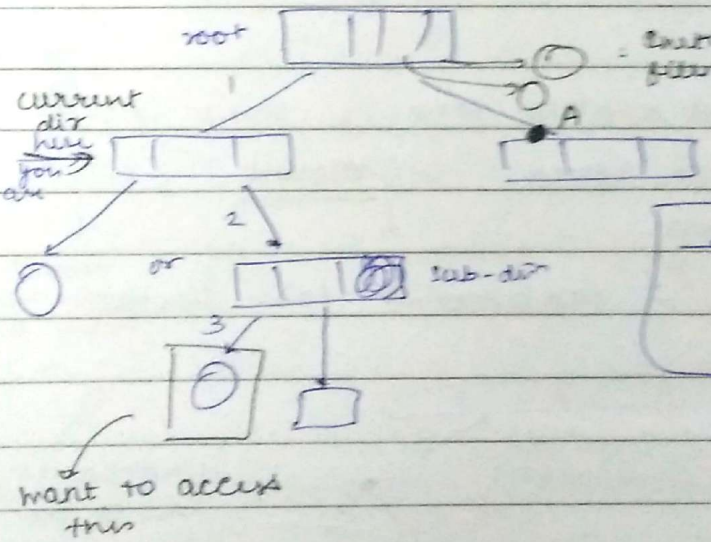
↳ each process has its own current directory

↳ OS ~~maintains~~ creates dir : User1, User2, ... , user dir

↳ We don't follow single level / two level.

3) Tree directory :

Eg 1



Logically, can have same file name, with same name as dir ↑

→ not allowing sharing of file b/c multiple users

absolute path : 1-2-3
relative : 2-3 (already at 2nd level)

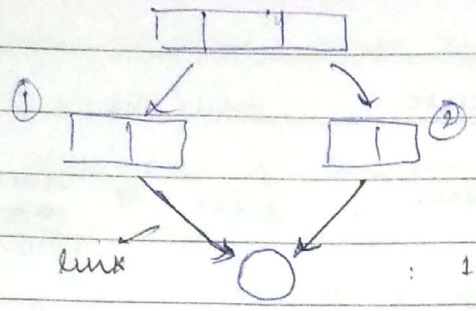
Roots : have ∅ subdirectories
leaf : have files

Adv. : More organized (1 type of file at one node)

→ every sub dir has a parent dir except root
↳ it is its own parent

4) Acyclic Graph directory :
Can share file

Teacher's Signature



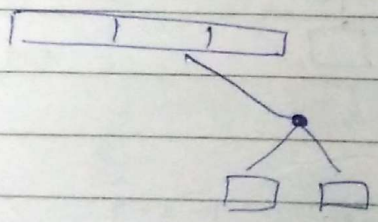
: 1 copy
whatever changes 2 make : visible to
1 : link

Disadv: If 2 deletes file, 1 will have dangling ptr. & if that file storage is given to some other file, 1 can access that.

↳ Eg: already exists : New file system : has to find place in existing system (Eg. 1)
a new device connected

OS will find a place for this new file system : mount

Maybe at A, I can mount this system



temporarily, you won't be able to see this directory (logically not visible)
↓
entries are not visible
↓
New file system will be visible

later when you unmount : get back original

→ Better way : have an empty dir. : every time new FS come : put there.
So, existing dir won't get invisible

Teacher's Signature

→ dir is also kind of file, only treated differently.

1 bit
- If file : bit = 0
- If dir : bit = 1

diff. dir & file

DATE : / /
PAGE NO. :

↳ Allowing permissions isn't easy.

1-5 : Read Access
5-7 : W
7-50 : R/W

} want to give these permission.

Either maintain access table for ~~each~~ all user for each file
↓
Not good way (many files, many user)

→ divide User :

User (For each file, name 3 categories)

Owner | Group | Others (Universal)

↳ creates file

Permission: O G U

9 bit no. : rwx rwx rwx : drwx rw- r--

1 1 1 | 1 1 0 | 1 0 0

↓ Owner will have all perm

↓ allow R/W, not x (let)

↓ allowing only read

↳ refers to sub directory (if named)
↓ optional

→ 7 6 4 : Permission (Hexadecimal)

↳ Write 5 Objective Questions from all syllabus.

13/4/18

Process A

① fd1 = open ("foo.c", R);

② fd2 = open ("test.txt", ARW);

④ → dup (3)

⑤ → open ("foo.c", W)

Process B

③ fd1 = open ("foo.c", W);

→ when process A says ①, file has to be opened → HD has to be accessed

Open → system call.

~~Os write~~ :

↳ There is a table in Kernel memory

Appl"

↓

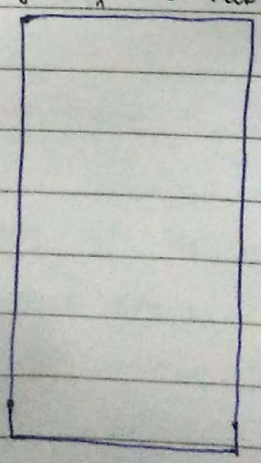
logical file system

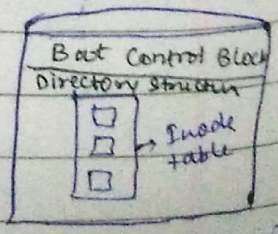
↓

Global File Table / inode System File Table

if file is open, info. about it might be available over here.

if not, it'll access disk, check dir.





1 volume (let)

Dir. has : File Name + Inode No.

FCB

Teacher's Signature

→ Just like each process has a PCB (all info about proces is there), similarly, all info of a file is stored in FCB (File control Block) → Inode ⇒ again a structure kind of thing

↓

stores attributes of a file

→ Name of file isn't stored in Inode. It has identifier → Inode no.

Directory has : File Name + INode no.

→ Have Inode table in HD : has INode no of diff files

→ HD : has Boot Control Block → 1st block

↓

contains

Boot info ⟹ how OS is kept, how it will be loaded from disk → all this info. (at time of boot, load OS from HD)

if OS not on this disk : Boot Block → empty

→ Volume Control Block : → HD has

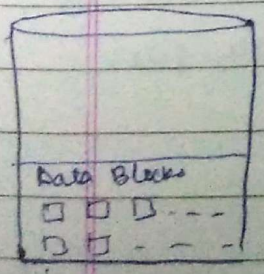
Info of Volume : size of block, no. of block, size of FCB, how many free blocks / Inode : all this info

↓ ptr also there

→ Inode : Metadata about files

↗ all remaining

→ also have Data Blocks : where actual data is stored



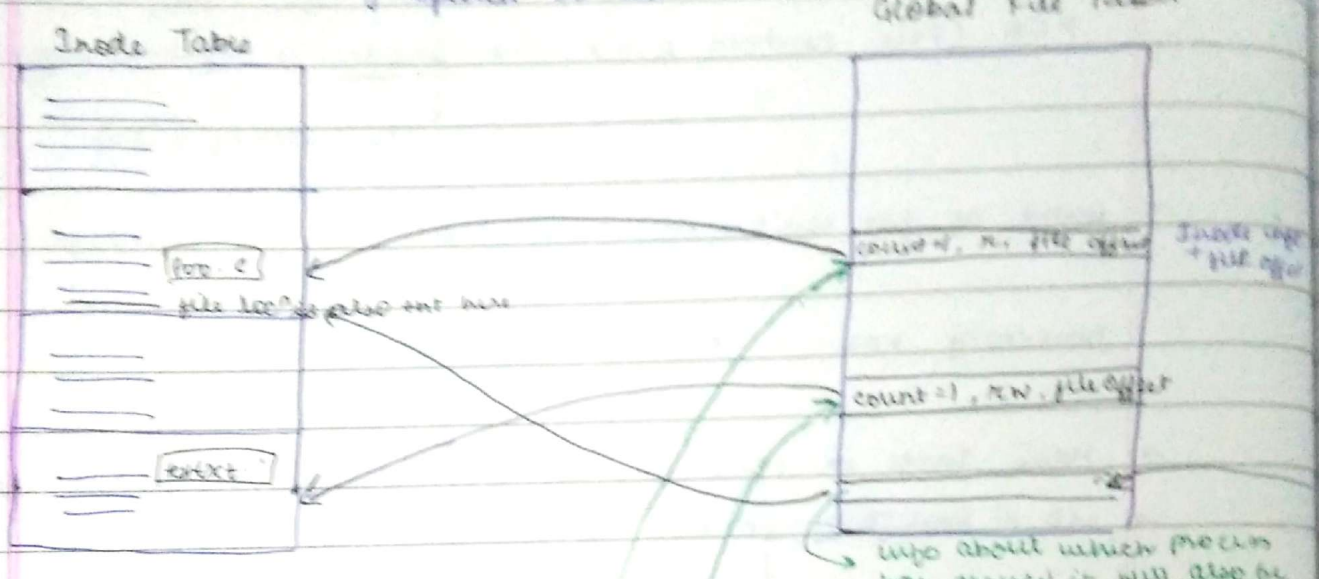
Pictorial

Actual

Teacher's Signature

Only 1 Global File Table.

① → check Global File Table & check whether foo.c is already opened or not

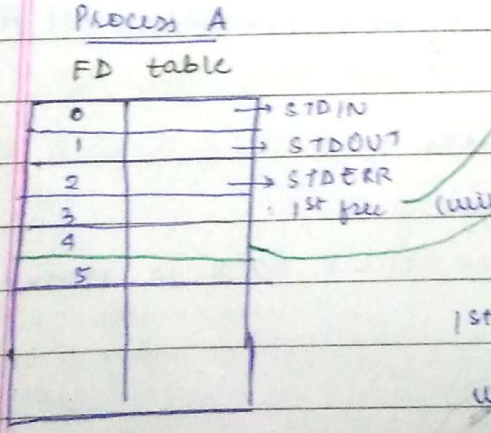


→ If not found : sometimes, it can bring dir also to mem / cache & get info about inode & make entry in GFT

→ file offset : where in file it is

→ Global File Table : store info about "foo.c".

→ when info is added to GFT : ⊕



1st descriptor is returned to process. It 'll use this fd no. to handle file.

② → Open ("test.txt")

Teacher's Signature

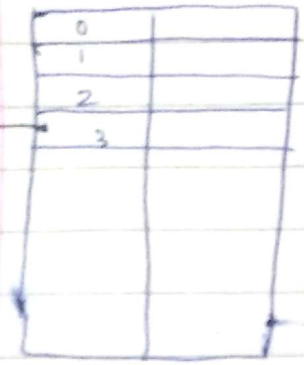
DATE : / /
PAGE NO. :

③ → can't use same entry because offset used by Process A will be diff. from Process B

⇓

entries will be diff → however, both are pointing to same physical file

Process B



However, if both want to access file, file synchronisation has to be maintained

→ Remove file : OS has to check all references to that file
; some OS can have same entry with increment in count
↓
mon + close file till count < 0 }

④ → dup (3) : also point to same entry in OFT
↓
sharing same offset → both will see changes made by other
count = 2

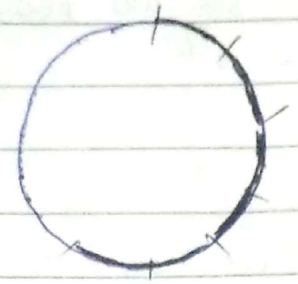
*↳ if has ② & ③ : we need have diff file offset for both → 2 diff entries will be there
↓
changes made by 1 won't be seen by other.

Ways of storing Data in Data Blocks:

1) Contiguous Allocation

One after the other.

If 2 sectors = 1 block = 1024 B



File : 4 KB

No. of Blocks needed : 4

have to see where do I've 4 blocks in series, only then I can allocate

Problem : I should know size of the file right at the beginning

starting add. of file will be stored in Inode

↳ whenever a file is created, an Inode will be created containing its info.

↳ can access file sequentially ✓

↳ can access file directly * ✓ (know Block size, Block no. add in start add. to allocate)

* What if user starts ↑ng size of file?

↳ find free space & copy whole file over there → very common.

what really do? → If need some more blocks, have free ones in last → extents : can store there & info about extent is stored in last block (prev. allocated)

store info of extent

OR

↳ rather than assigning blocks, I'll assign clusters.

Suppose 4 blocks = 1 cluster

→ Gave 1 cluster

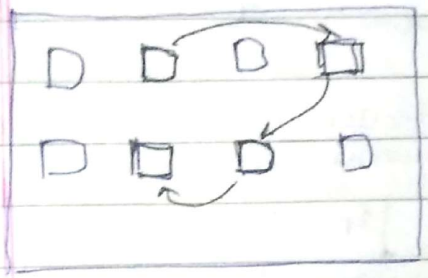
Problem : wastage ⇒ Internal fragmentation

Teacher's Signature

2) Linked Allocation:

don't know file size : can allocate blocks with prev one pointing to next one

File size : No problem



Sequential access ✓

Direct access : ~~✓~~ ✗

(info.

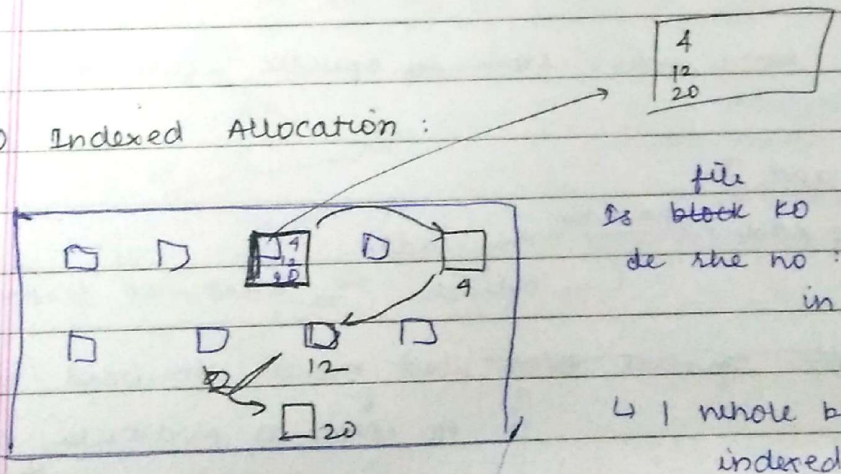
Problem : 1) wasting mem. in every block for ptr (4 bytes)

2) Time to access next block ↑

magnetic head will move ~~too~~ & fro.

3) If anyone block gets corrupted, ⇒ ptr. is lost.

3) Indexed Allocation :

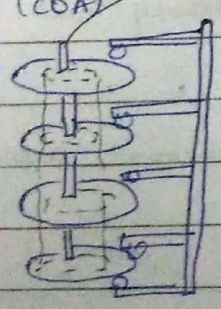


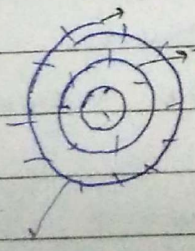
file

Is ~~block~~ ko all karne block de rhe ho : store in table in 1st block

↳ 1 whole block is wasted in indexed table

16/4/18





Formatting : sectors are formed

each secoor

header + data + ECC

error correcting code

HD

Teacher's Signature

whenever you store data in disk, a code is generated acc. to that data → ~~is~~ stored along with data

↳ There is also a controller → finding file, checking its size, ... is add
↓ → keep track of sectors
can have small cache

Depending on code in data → controller will know if has error (checking code & data)
↓ If error
Controller can correct it (if only data is corrupted.)
↳ Beyond repair (sector is declared bad sector & free sectors are available)
data can be retrieved from bad sector & put into some other spare sector. New mapping will be generated (mapping will be in controller)

→ ECC is checked regularly.

→ all tracks at same radius from ~~the~~ spindle : form a cylinder

How data is read ?

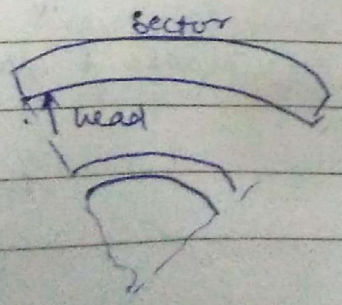
1) 1st, Logical Add. converted Phy. add.
(— cylinder, — track, — sector)

2) ~~1st we rotate spindle &~~ head moves forward / back
↓
to come at particular track
↓
seek time (Tm

3) Rotate spindle to get sector
↳ Rotational latency (Time)

↳ Time taken to do all this :

Random Access Time = Seek Time + Rotational latency



Teacher's Signature

head can have either const linear velocity

or const angular velocity : usually em

↓

bits in innermost track will be tightly packed as compared to outer track

→ Disk Transfer Rate (Disk Bandwidth) = No of bits transfered / Time from I/O request till last transfer taking place

Eg. 98, 183, 37, 122, 14, 124, 65, 67 : process requests OS to access disk. OS will communicate with device driver. device controller communicates with disk here

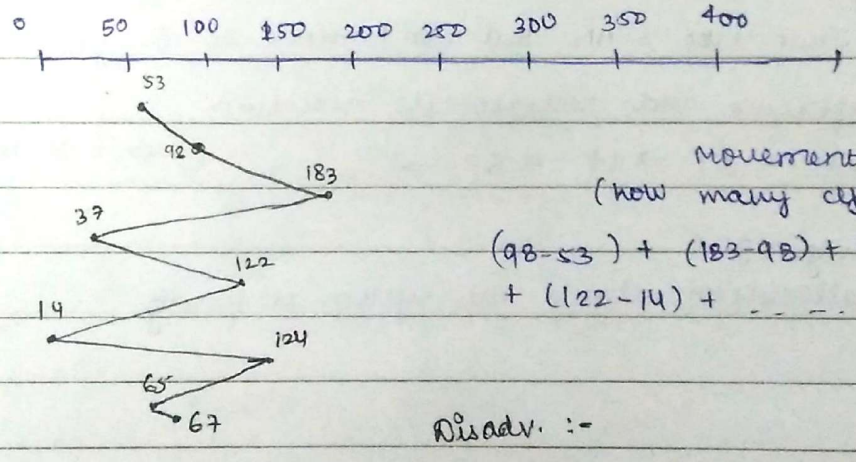
↓ cylinder no.

all in device Queue (request)

Head: current dt : 53

Disk Scheduling Algorithms :

1.) FCFS



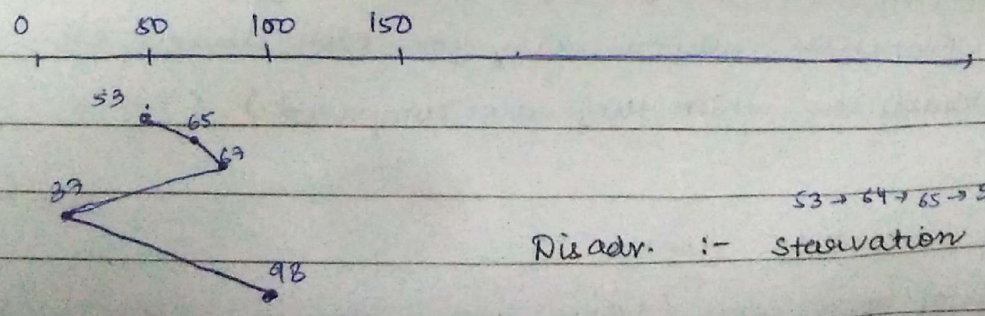
Movement of head : (how many cylinders it moved)

(98-53) + (183-98) + (183-37) + (122-37) + (122-14) + ...

Disadv. :-

2.) SSTF (Shortest seek time 1st)

↳ choose those which are closest to head



53 → 64 → 65 → 52 . . . Request from 165. . .

Disadv. :- starvation of resources (not dealt with)

if going ↓ : will go ↓

3) SCAN (Elevator algo)

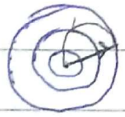
suppose at 53 : Know moving outwards ⇒ no. of cylinders Reach 0, then move inwards

53 → 37 → 14 → 0 → 65 → 67 → 98 → 122 → 124 → 183

Reach 0 requires dir^n

No of Head Movements : much less 53 + 183

4) C - SCAN (Circular SCAN)



moves outwards : reaches 0 & jump back to center & starts moving outwards.

disadv. :→ why to go to 0 again & again ?

just needs to go to extreme case & turn back

5) LOOK : Just like SCAN, but don't move to 0, just move to extreme reference and reverse its direction

Head : 53 → 37 → 14 → 65 → ... (won't go to 0)

→ Scheduling algo's

* depends on allocation algo's the system is using

18/11/18

I/O Devices

→ Modem

→ All I/O devices are of diff. kinds of

→ Diff. Properties which vary for each devices are: (attributes on which they are compared) :

↳ Speed

↳ I/p or O/p device

↳ Type of data sent to / taken from device (serial / block of data Byte by Byte)

I/p device which sends data byte by byte : keyboard, mouse character based

Monitor : ch. stream

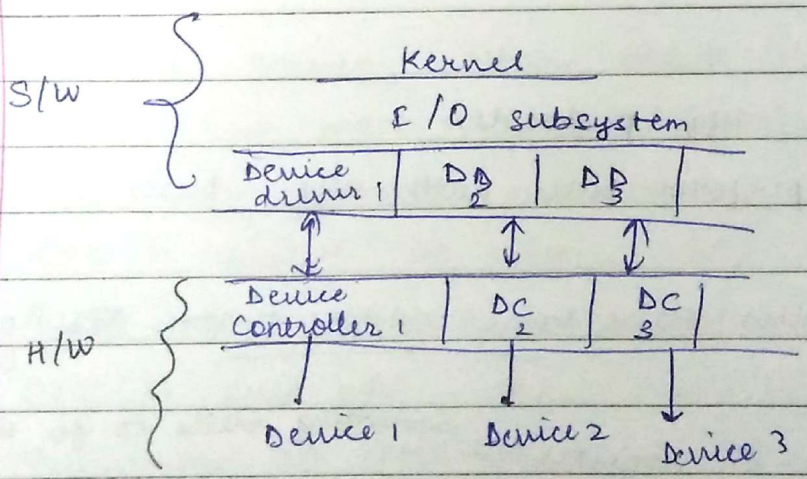
Teacher's Signature

↳ whether have sequential data or random access
↓ ↓
Magnetic tapes CD-Drives, HD

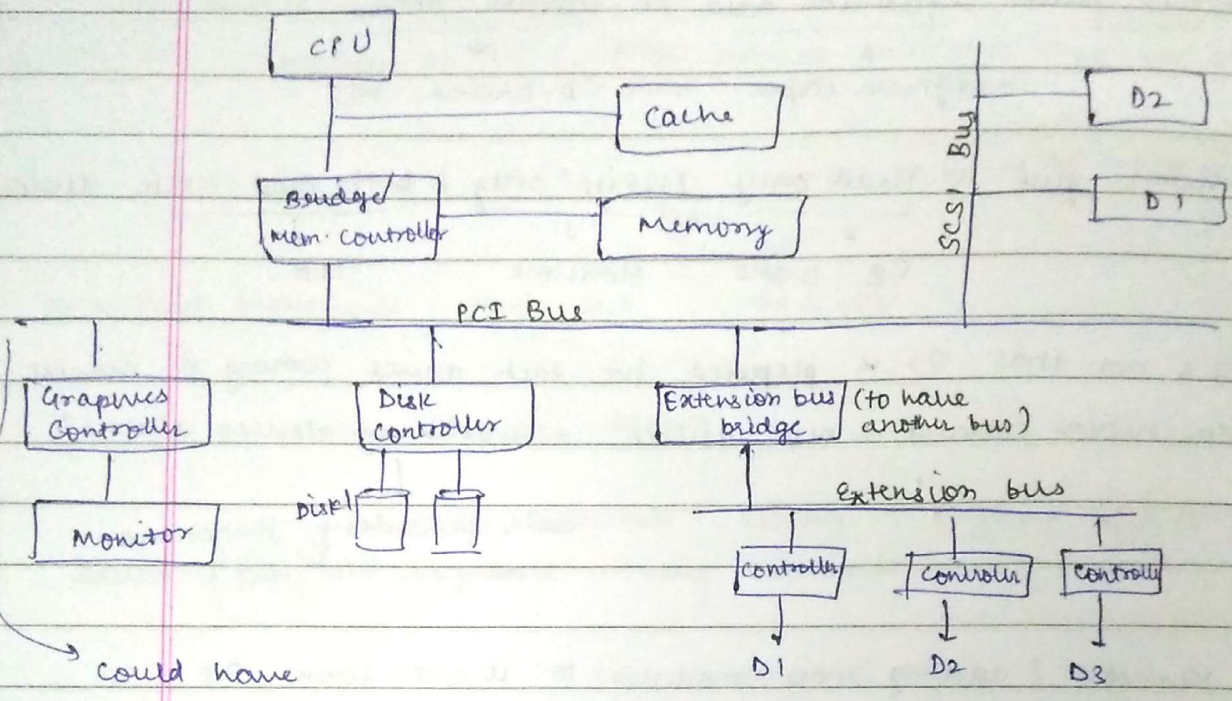
↳ whether just a read only / write only / both read-write device
↓ ↓ ↓
CD-DISKS Monitors HD

→ It's not that OS is prepared for each device coming in market
So, when connect a new device :- "installing device driver".
↓
codes providing a common interface b/w OS & device

→ Kernel : adding more modules to it to form OS
(Memory Management,
I/O " ...)



→ devices are connected through ports, wires (buses)
Commonly used buses are :
PCI : Peripheral connect Interface
SCSI : Small computer System Interface



could have
Interrupt controller,
then DMA controller

↳ All buses use diff. protocols.
Certain devices perform better with diff - buses.

↳ Each device controller have same registers + some Processor (may be)

CPU : host

Data - in register : Something needs to go in host
Data - out register : — — out —
status register : whether command is completed or not, whether got an error or not
control reg : change mode of comm^n

maybe a busy bit ← ↓ ⇒ device is busy

→ If wants to write on HD :
OS checks device Q : as soon as busy bit = 0 ⇒ device is ready to communicate
↳ checks data - out register → some value → right bit = 1, busy bit = 1

After execution : status reg. will have some value, make busy bit = 0

→ like a handshaking problem

↳ I/O Map : Each device will have an address comm

↳ Memory Map I/O : giving part of mem. to I/O device
can use simple LOAD & STORE inst^n
(Data transfer become fast)

~~Scan~~ Have buffer:

↳ either buffer in mem. region or with controller

⇒ Everytime CPU is checking whether device is busy or not ⇒ 1) Busy waiting

CPU time wasted

↳ 2) Interrupt driven : when device is free, it sends an interrupt
Many may want to send interrupt at same time.
Interrupt controller will choose 1 on basis of (priority, maybe).
CPU 'll complete its cycle : CPU will check into ISR table & find vector. Add. of ISR is kept in vector table. (↳ Interrupt Service Routine)
CPU 'll find add. ^{n}, run it & handle the comm^n.
Every time an interrupt comes, do same.

Interrupt may be

Maskable (MI)

↳ can stop them from interrupting the CPU (CPU is doing something imp. & don't want to get interrupted), low-priority can be masked

Non-Maskable (NMI)

↳ which are critical, need immediate attention of CPU.

Teacher's Signature

(maybe)

MS / NMI : decided upon priority : May create problem everytime an interrupt

3) DMA : DMA Controller

CPU is going to initialize DMA Controller. Info. CPU gives to DMA Controller : add. of device, no. of bytes that will be transferred, whether R/W op^n, Mem. address where it has to be kept.

DMA will keep on taking data from device & put it in mem.

↳ Controller has taken control of bus

If CPU wants to use it, it has to wait : Cycle Stealing