setjmp
longjmp } jmp in any where in scheduler f^n.

27 Feb:

Files:

A seq. of bytes can be interpreted by (an appl^) not kernel.

1) Text = All bytes are human readable.
2) Binary - elf file
• Most typically describing the organizat^ of data inside the file.
↳ each type serving the needs of a particular appl^ (Not kernel) — (file ke type se kernel ko jarak kahi padta).

• Vlc (i.e appl^) will do open()
- Kernel will simply provide open(), read(), write() to access file data.

•

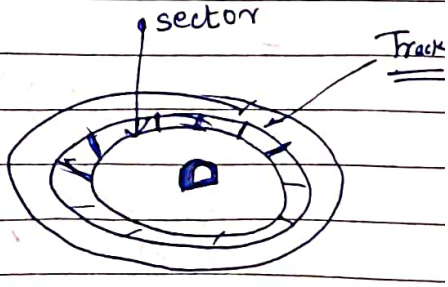
* Access Method:

OS provide 2 types of access-
1) sequential: — linux provide (using open(), write())
2) Direct: directly read that byte/block.
↳ popen(), pwrite() etc.

(sequential has no. block.)

* Disk:

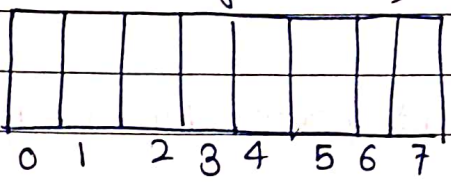
7200 rpm = 120 rps



Device Driver:

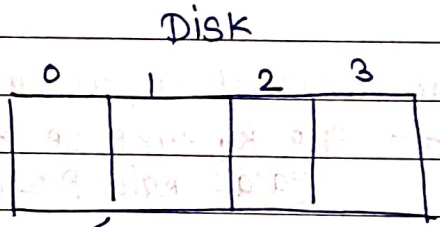
ATAPI devicecontroller } listen on I/O port.

Disk (seq. of sectors).



512 bytes (sector) (physical block)

either read sector interms of sector or ~~bytes~~ KB



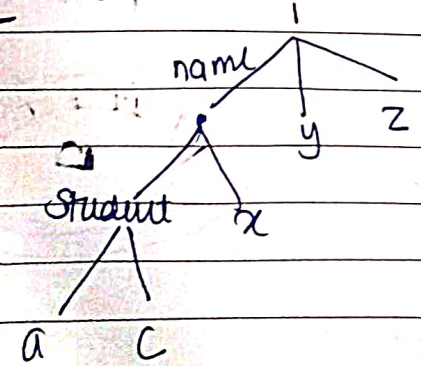
(logical block)

* File system implementat^n layering.

talking in block

1) Appl^n program:
2) logical file system (system calls) sys read
3) file organizat^n module (simple file reading) (file read)
4) Basic file system - read data into OS buffer
5) IO control — idrew → bread
   - wait until get data
     - we use sleep
     - we wait for buffer..
       - sched()
         - switch()
           - jump into ~~another~~ schedule
             - New process

* OS job:



disk gets data of buffer.
then disk will interrupt — call wakeup() → wakeup(1)
then return, i.e return return
Date Page
Resume Runnable process.

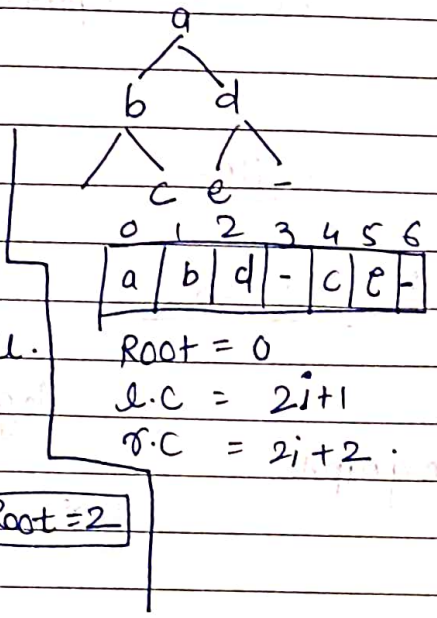
- **formatting:**

Binary tree:
- impl. using array
-

- explicit indexing:
  — any element can go anywhere.

0	1	2	3	4		
b		a	c		d	e
-1		0	-1		c	-1
3		3	1		-1	-1

Root = 2



Root = 0
l.c = 2i+1
r.c = 2i+2.

- formatting is like initialising data structure.

- Diff. types of layout:
  1) 1 level directory
  2) 2 level directory
  3) Tree structure directories: (folder ke andar folder).
  4) Acyclic Graph directories: } — edge always down ↓
     ↳ multiple parents.
  5) General cyclic : edge upward ↑

ln ; ln -s
↳ link system call.
- alias — another name for same file.
ln cp /etc/passwd
ls -l ./passwd — rw...1
ln ./passwd ./aaa rw...2 ./aaa alias
rw...2 /passwd.

Hard link: do not create on directory
create on file.

ln & ln -s

ie. delete s1 file) = s1 = NULL.

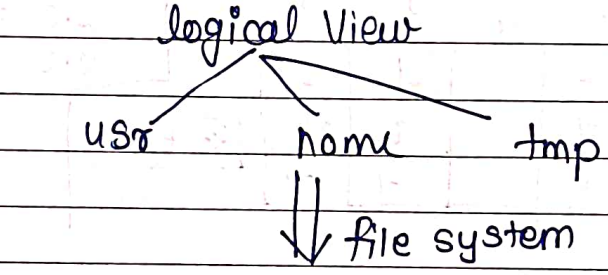
if we delete passwd then delete whole file, (soft link (size is linked to ~~the another file~~)

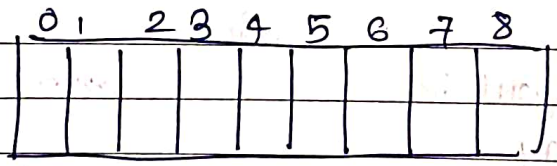
ln -s ./passwd ./s1 — ./s1 is link to password.

sysfile.c

* 28 feb:

cyclic graph





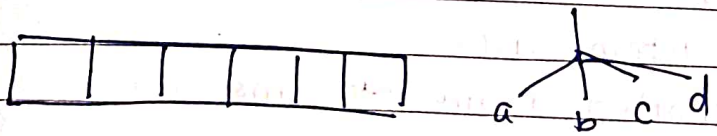
Disk.

logical block = 512 bytes

* Mounting of file:

/dev/sda 8

Partition

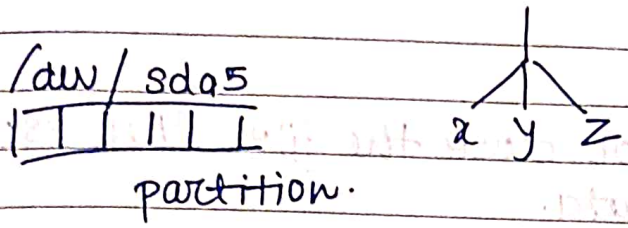


ls -l /dev/nvme0*

nvp1 nvp2 ~ partition files.

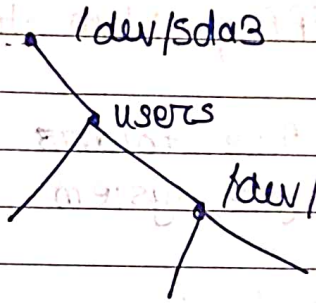
fstab – filesystem table.

cat /proc/partitions.



sudo mount /dev/sda5 /users — on /dev/sda3

attach that tree onto user.



sudo fdisk —— create partition

sudo mkfs -t ext4 /dev/sda2 —— create filesystem

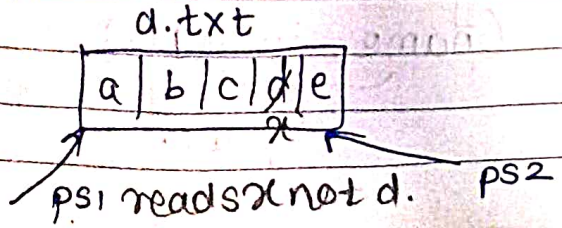
make filesystem on partition of disk/pendrive.

Now we can mount that filesystem. (it is mount then when we unmount, then it will take time, becoz we have to unmount from OS buff. to hard disk or write on OS buffer)

* File sharing semantics:

Unix file system (UFS):

-1)



If 1 process change the data then next process must read new data (even P2 does not close the file)

AFS:
If PS2 does not close the file then PS1 read the old data.

- Bootloader must know the locat^ of ext4.

* filesystem : Diff. problem to be solved

1) Main block, free, folders etc. i.e summary.
2) mounting a file system

1) How to store file:

1) we have to store data, attributes, name, link count.
↳ permission etc.

- Hardlink:

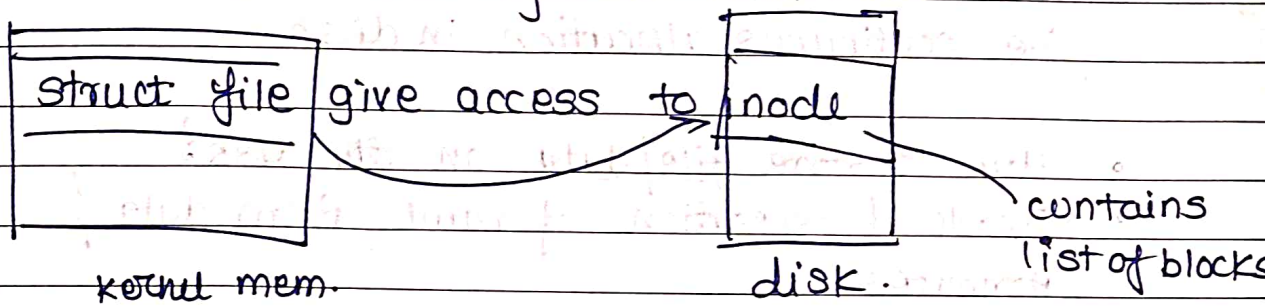
- need to seperate name from data
  i.e data is seperated from name.
  - Both should refer to the same data.

- Attributes:

- typical file control block:

- Name is stored seperately
  i.e in directory (name, inode no.)
  in pair.

• In memory data structure.
Kernel also has in memory data structures:

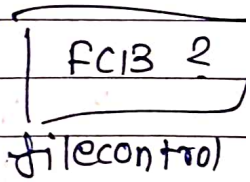


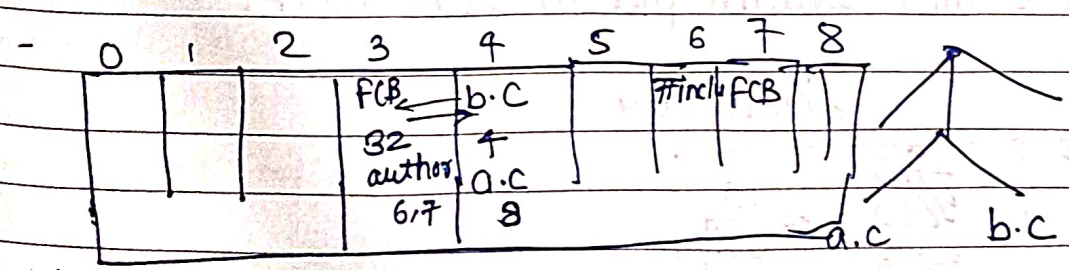
ln ← if we create 3 files ~~then~~ then
inode = 3
and if we copy one of them then
inode value = 1.

2)* Directory implementat^n:

– lookup
/home/student/b.c
if i am looking on home then i have to lookup on student.

treat
• directory as file ← files & folders
data of that file is



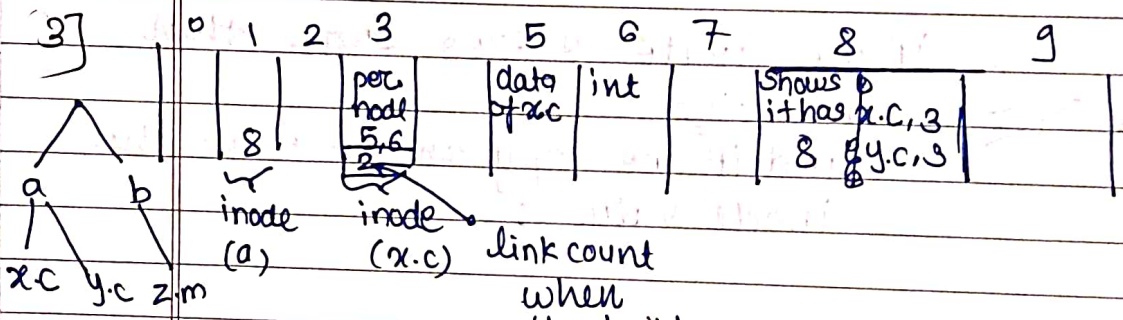


initially,
linear search

block no. read from inode
- inode in INode table
- inode no. present in file ke data block mein.

How to grow the file
i.e increase blocks.
i.e continuous allocation in disk.

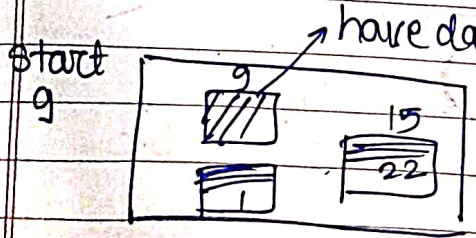
- How to store file/data on the disk?
- Incode] seperation of name from data / attributes.



when create hardlink; i.e → increment link count.
~~another~~ same file with another name.

→ Cont. allocat^n read file fully.
→ Problem is fragmentation.

* linked allocat^n of block of file.
block will contain ptr to next block.

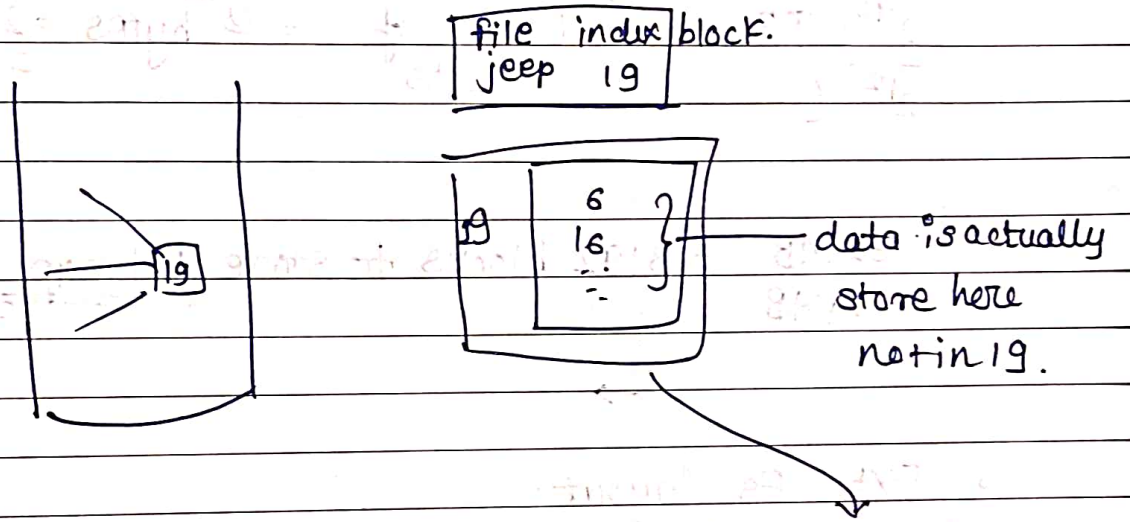


defragmentat^ on window.

- FAT - file allocat^ table.

- in the beg. we have array of indx.
- In the beg. list was madi.

- Indexed allocat^:

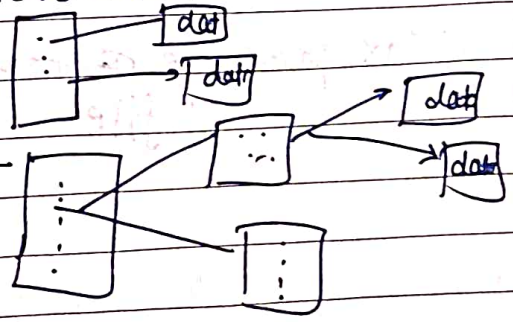


The block which we have read we check into this table and go to that block.

size of file limited to 256 blocks.

- Multilevel Indexing:

Today we use UFS - combined scheme for block allocat^.
if we want more block to store data we use
- single indirect
- double -||-
- triple -||-



4]. Free space Management:

- bit map indicating which block is free & empty.

$\frac{2^{40}}{2^{12}}$ (TB = 2^{28} blocks of bits = 2^{25} bytes = 2^5 MB

$\frac{32\,MB}{4\,MB}$ = 8192 blocks to store bitmap.

- EXt2 Fs layout:

In linux kernel code ext2_inode.

__le32 i_block [EXT2_N_BLOCKS] ⟶ 15 blocks

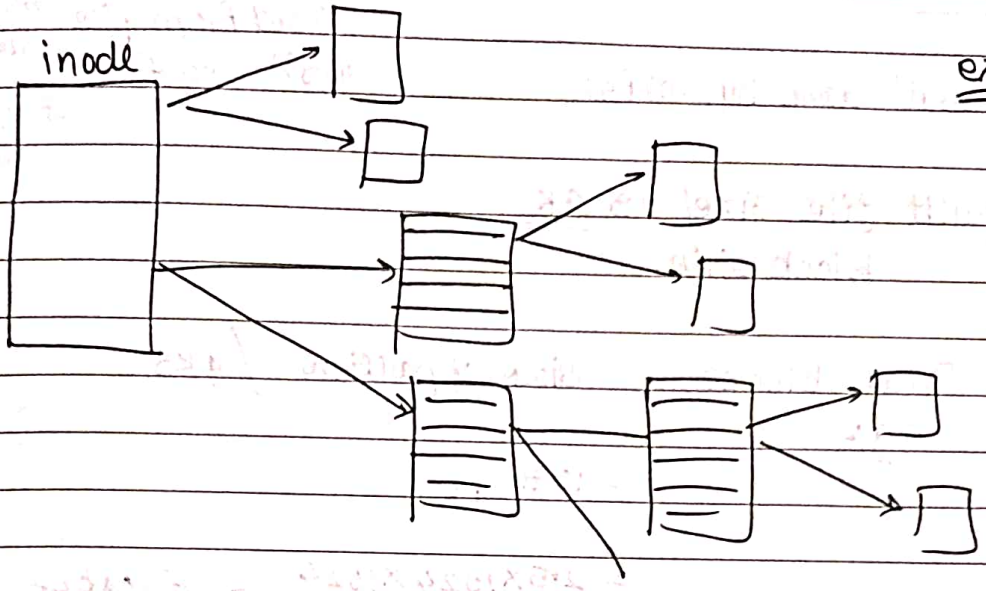
Block size = 4K
1 block = 4K/4 = 1024 entries

12 - direct
single - indirect
i.e 1024 entries — block no. we store
double for 2-indirect - $(1024)^2$
triple = $(1024)^3$

Max possible size of file = $4K\left(12 + 1024 + 1024^2 + 1024^3\right)$

$\approx$ 4TB

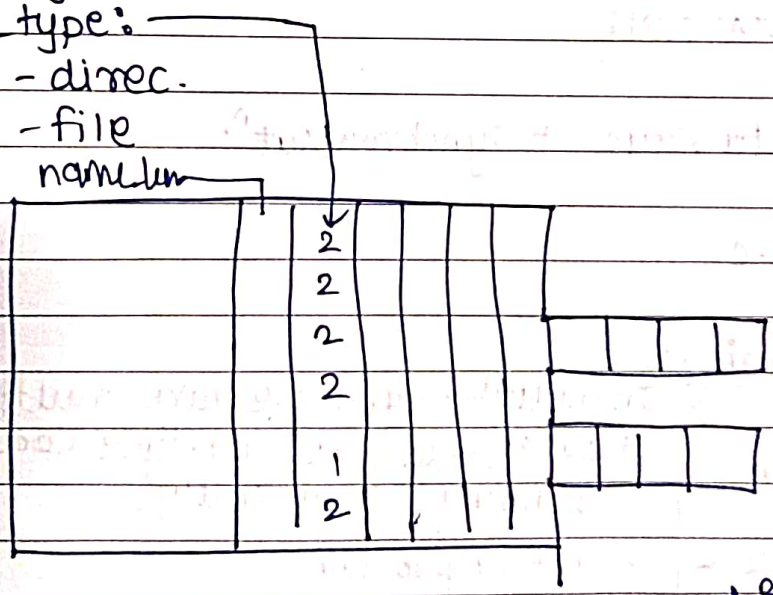
Magic no: ef53



ext2

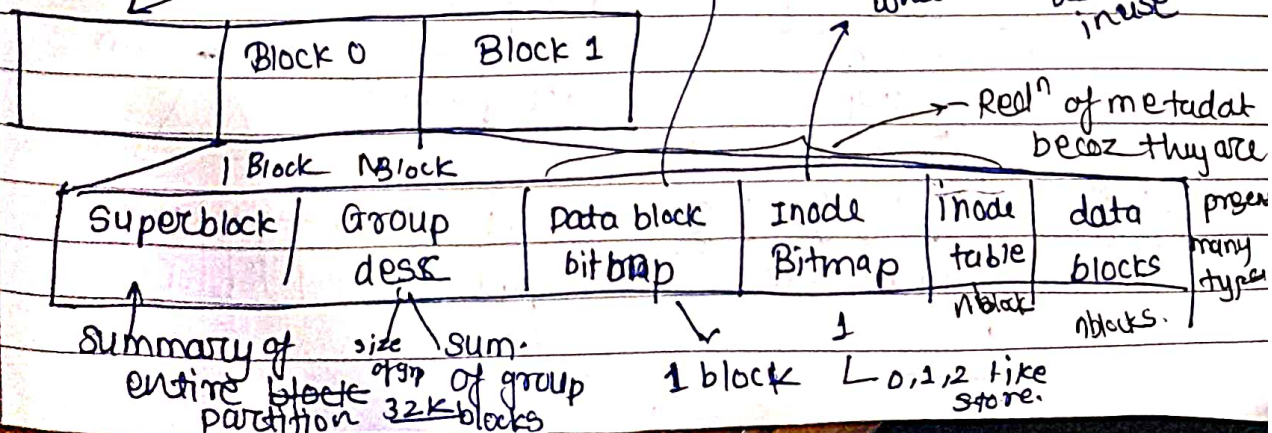
file name always stored with multiple of 4 with ending Null.

file_type:
2 - direc.
1 - file



skip → delete

- 0x400 - 1K



bitmap of each block in the grp.

which inode is in use & not in use

Real^n of metadat becoz they are

many types

ls -lhi → give inode no. du filename → gives file size.

we can browse file/data structures

sudo debugfs /dev/sdb1

sudo dd if=/dev/sdb1 skip=0 bs=4k count=1 of=/tmp/4k

read from file write to file.

* Calculat^n done by mkfs:

Default file size is 4K.
Block size.

Total blocks = size of partion / 4KB

eg:

= 10GB / 4K

= 2.5 × 1024 × 1024 = 2621440

6 March:

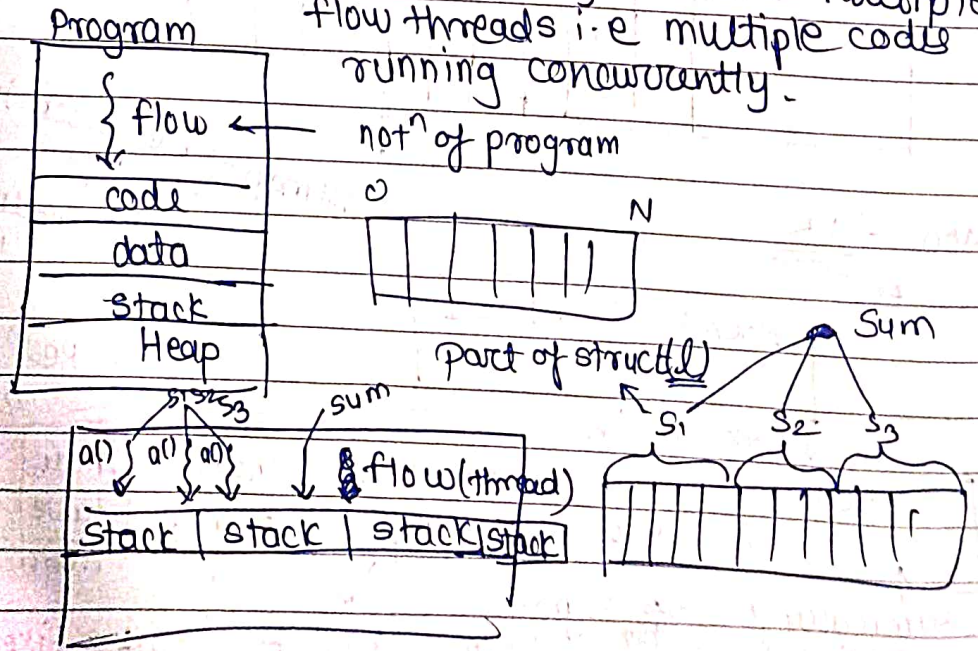
* SYNCHRONIZATION:

OS = data struc. + synchronizat^.

vi pthread.c.

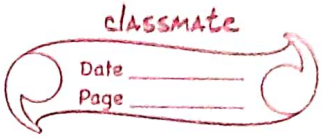
* Multithreading:

In multithreading we have multiple flow threads i.e multiple code running concurrently.



There are 4 stacks.

- Concurrent is not always parallel
- parallel is always concurrent.




```
int main()
{
```

```
struct limit{
    l, h, sum
}
```

// till here single thread

```
for(        ){
.
} for() {
    sum =
}
```

⟹ create pthread. main wait for child.

```
void * add(     ){
}
```

// sum (in the structure).

- vi race.c

c is not atomic ⟶ it is not guaranteed. either 0 or 1.

thread 1	thread 2
mov	mov
add	add
mov	mov

c = 5 6 shared globally.

thread 1 — thread 2.

r1 = 5 r1 = 5

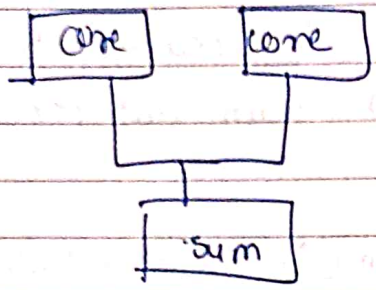
r1 = 6 r1 = 6

it is not 7 it is 6

th1 mov c, r1
th2 mov c, r1
th1 add r1, 1
th1 mov r1, c
th2 = add r1, 1
th2 = mov r1, c.

* **Races: Reasons:**

Introduces concurrency.

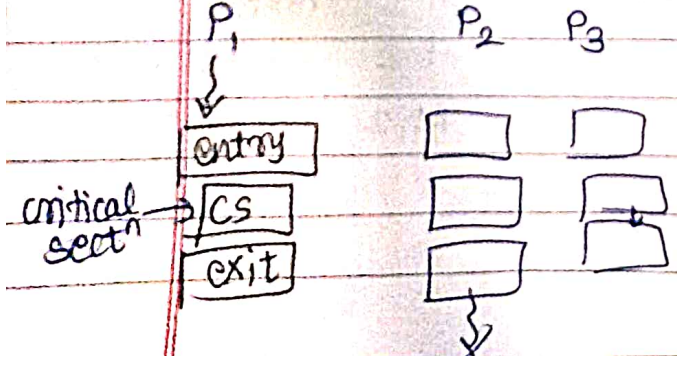


7 March

* If one process in critical sec. then other will not in critical section.

```
do {
    entry section
        critical section
    exit sect^n
} while (true).
```

2. **Progress:**



P_2 & P_3 is not in critical sect^n so P_1 will go to critical sect^n for limited. P_2 & P_3 ke wajah se nahi time rok sakte.

3. Bounded waiting:
if P_2 & P_3 have loop, then it will assures to the P_1 it will go after sometime in critical sect^n.

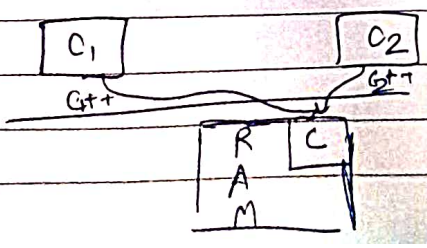
* Peterson's Solutions: (software sol^n).

- 2 process sol^n
- LOAD, STORE instr^n.
- 2 procc. share 2 var.
  int turn ← whose turn is to enter the CS
  Boolean flag ← shows process is ready TO enter the CS.

flag1 = ~~T~~ F flag2 = T
turn = 1

P_1	P_2	
1) flag 1 = ~~T~~ F	1) flag 2 = T	i = 1
2) turn = 2	2) Turn = 1	j = 2
3) while(flag2 && turn = 2)	3) while(if flag 1 && turn == 1)	as long as other process turn to run then wait.
~~flag~~ ??	→ when flag 1 = F then it will go to the critical section	

Test&set: return the value of word variable & set it to true.



both access C, hardware assure then can access one by one.