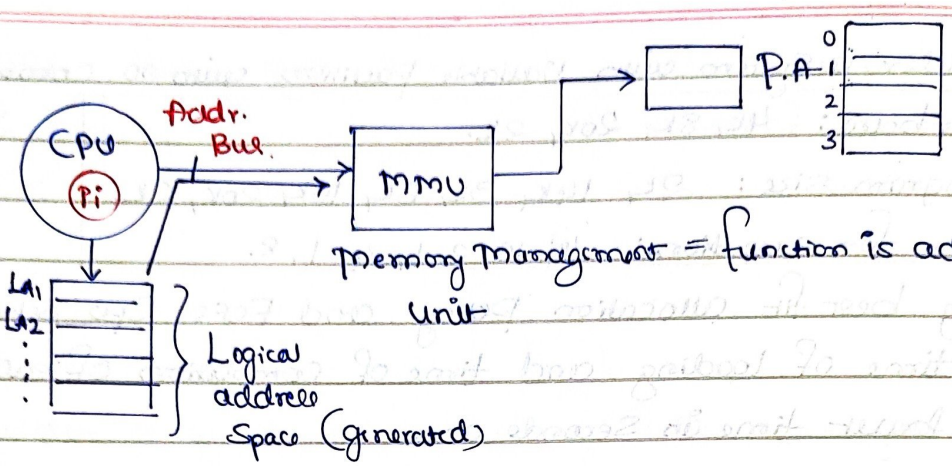


Memory Management = function is address translation.
Unit

Eg: ① L.A.S = 64 MB ; PAS = 4 MB
LA = 26 bits PA = 22 bits

② LA = 33 bits
LAS in bytes, if word size is 64 bits = 8B.
$LAS(B) = 2^{33} \times 8B = 8G \times 8B$
$= 64\,GB$

PA = 23 bits
PAS, W.S = 64 bits = 8B
$8M \times 8B = 64MB$

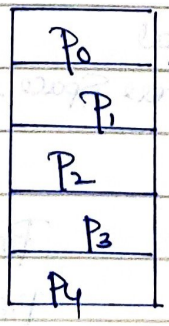
* Logical address: address generated by CPU to access instruction/data unit of program in execution.

* Physical Address (Real/Absolute): address needed to access needed to access the program instructions/data unit in physical memory ie address in MAR.

1. Simple Paging:

* Organising of LAS/VAS:

LAS: 5KB. → LAS is divided into equal size units known as pages.



→ Page size is generally in the power of 2.

Formulas

→ No. of page (N) = $\frac{LAS}{PS}$

→ Page No (P) = $\log_2 N$

$N = 2^P$

→ Page offset / displacement (d) = $\log_2 PS$ (bits)

→ P.S = 2^d B

L.A / V.A format

LA = 13 bits

P	d
3	10

Q1.

(i)

L.A.S = 32 MB

P.S = 4KB = 2^{12}

N = 8K [2^{13}]

P = 13 bits

d = 12 bits

L.A: | P | d |

13 12

LAS = 2^{25} = 32 MB

(ii)

L.A = 25 bits

d = 13 bits

* What are the no. of pages in the system?

LA: P d (25 bit)

12 13

N = 2^{12} = 4K

(iii)

System has 2K pages and LA of 32 bits. What is P.S?

N = 2K

P = 11 bits

32 bits

P	d
11	21

d = 21, P.S = $2^d = 2^{21}$

= 2 MB

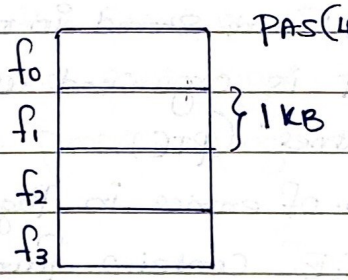
* **Organization of P.A.S (4KB):**

→ P.A.S is divided into equal size units known as frames [page frames].

→ Frame Size = page size

→ Each frame holds one page

→ Any page can be stored in any frame (NCG)



Formulas

* No. of frames (m) = $\frac{P.A.S}{FS}$

* frame no. (f) bits = $\log_2 m$.

$m = 2^f$

* frame offset = page offset = d

* P.A format:

12 bits

f	d
2	10

Q1. L.A = 31 bits, P.A = 20 bits, P.S = 4 KB

$N = LAS / PS = \frac{2^{31}}{2^{12}} = 2^{19} = 512K$

$m = \frac{2^{20}}{2^{12}} = 2^8 = 256$, $P = 19b$. $f = 8$ bits, $d = 12$ bits

LA: | P | d | 19+12 = 31

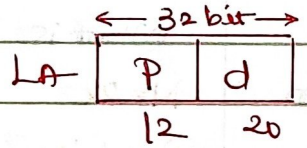
PA: | f (8) | d (12) | = 20 bits.

Q2 System has 4K page and 1K frames, Calculate the size of P.A.S, if LA = 32 bits.

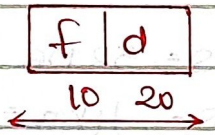
P = 12

N = 4K ; m = 1K = f = 10

LA = 32 bit



PA:



$PAS = 2^{30} = 1GB$

* Organization of M.M.U [page table / address table]

→ Each process has its own P.T

→ P.T's are stored in memory

→ P.T is organised as set of entities, known as "Page table entries" (PTE).

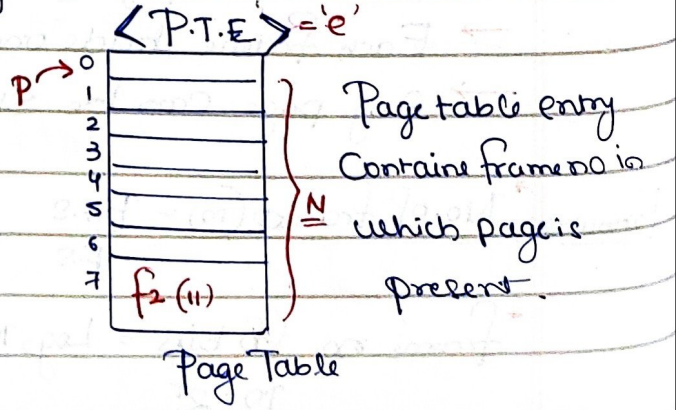
→ No. of entries in Page table = No. of pages.

→ P.T.E Contains frame number in which page is present.

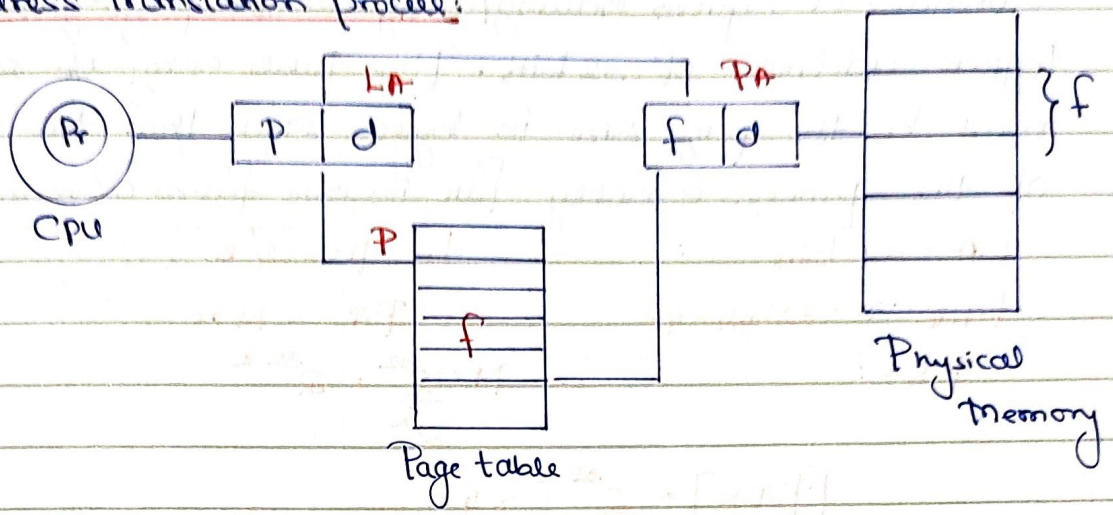
→ P.T.E is denoted as 'e' (Bytes.)

→ P.T Size = N * e (Bytes)

e ≥ 1B



Address translation Process:



Q1. A computer System using paging technique implements an 8KB page with page table of size 24MB. The page table entry is 24 bits. What is the length of virtual address in the system?

Ans

P.S = 8KB	P.T.S = 24MB
d = 13 bits	e = 24 bit = 3B.

P.T.S = N x e.

$N = \frac{P.T.S}{e} = \frac{24}{3B}\ MB = 8M$

$\therefore$ 23 bits

Q2 Consider a computer System with 40 bit virtual addressing and page size of 16KB. If the computer System has a one level page table per process and each page table entry requires 48 bits, then the size of per process Page table is megabytes.

Soln

V.A = 'l' bits

N = '2'

n = 'H'

$P = \log_2 2$

$d = (l - \log_2 2)$ bits

PA: $(\log_2^H + l_0 - \log_2 2)$

$(l + \log_2 H - \log_2 2)$

$P.T.S = 2$

$= 2^{1 + \log(H/2)}$

$= 2^1 \times 2^{\log(\frac{H}{2})}$

$= 2^1 \times (\frac{H}{2})$

Q3. Consider a System having/using Simple paging technique with logical address of 32 bits. Page table entry of 32 bits. What must be the page size in bytes, such that the page table of the process exactly fits in one frame of memory (PAS)?

Sol^n L.A = 32 bits
P.T.E = e = 32 bits = 4B
P.S = ?

Let P.S = 2^x B
P.T.S = N * e
$N = \frac{2^{32}}{2^x} = 2^{32-x}$

$[P.T.S] = 2^{32-x} \times 4B$
$= 2^{34-x}$

$2^{34-x} = 2^x$
$34 = 2x \quad \therefore x = 17$

Q4. Consider a System using Paging technique with VA = 32 bits and P.S = 4KB, what must be appropriate P.T.S in bytes, given P.A.S = 64MB.

Ans PAS = 64 MB
VA = 32 bits
PS = 4 KB

P.T.S = N * e
$N = \frac{2^{26}}{2^{12}} = 2^{14}$, $N = \frac{2^{32}}{2^{12}} = 2^{20} = 1M$

P.T.S = 1M × 2B
= 2MB

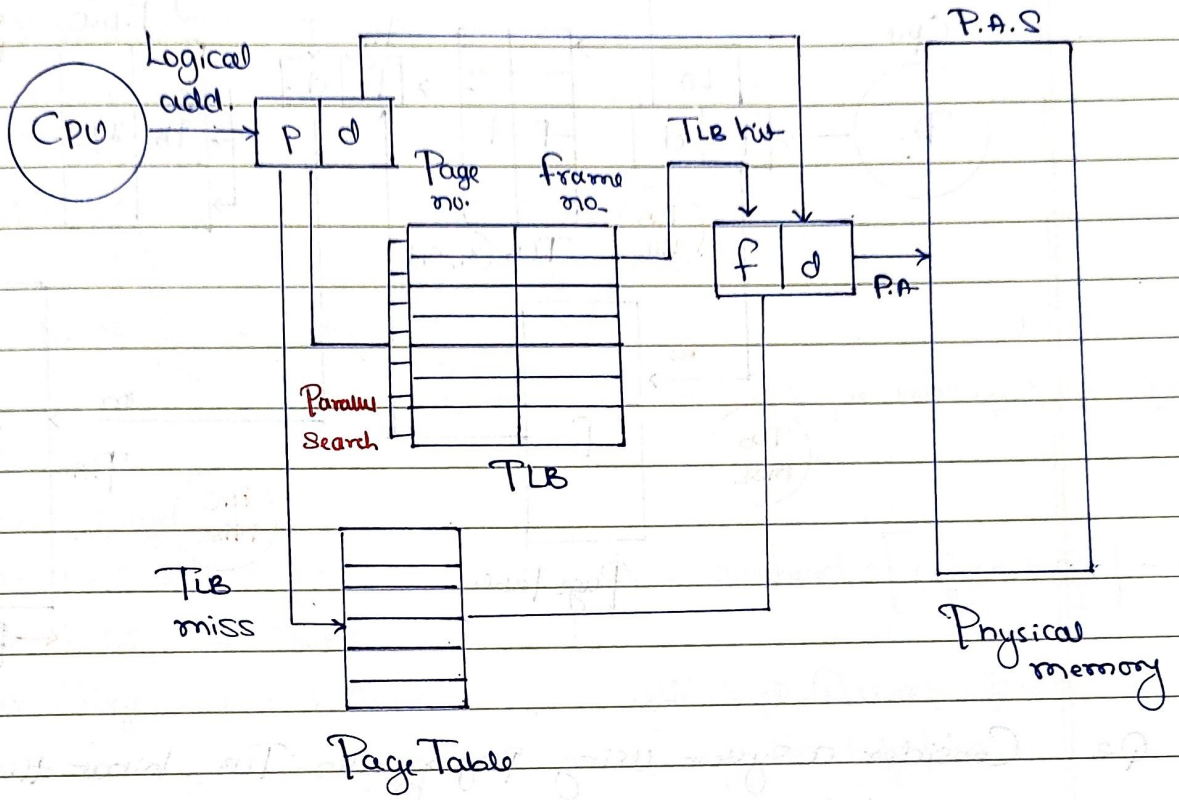
Performance of paging:

1. Timing issue:
   * Main memory access time (MMAT) = 'm'
   * Effective memory access time (EMAT) = 2m.
   * Objective: Reduce EMAT from 2m to m.

Cache memory:
* faster memory access time
* costly
* Smaller in size
* Supports parallel search.

↓
TLB
(Translation Lookaside Buffer)

Paging Hardware with T.L.B.



Formulas

* EMAT = 'm'
* TLBAT = C
* EMAT = 2m.
* TLB hit = 'x' ratio
* TLB miss = '1-x' ratio
* EMAT = x(C+m) + (1-x)(C+2m)

Q1. m = 100 ns

C = 20 ns

(a) EMAT with paging : 2m = 2x100 = 200 ns

(b) If x = 90·1. what is ELB TLB : x = 0.9, (1-x) = 0.1

= x(C+m) + (1-x)(C+2m)

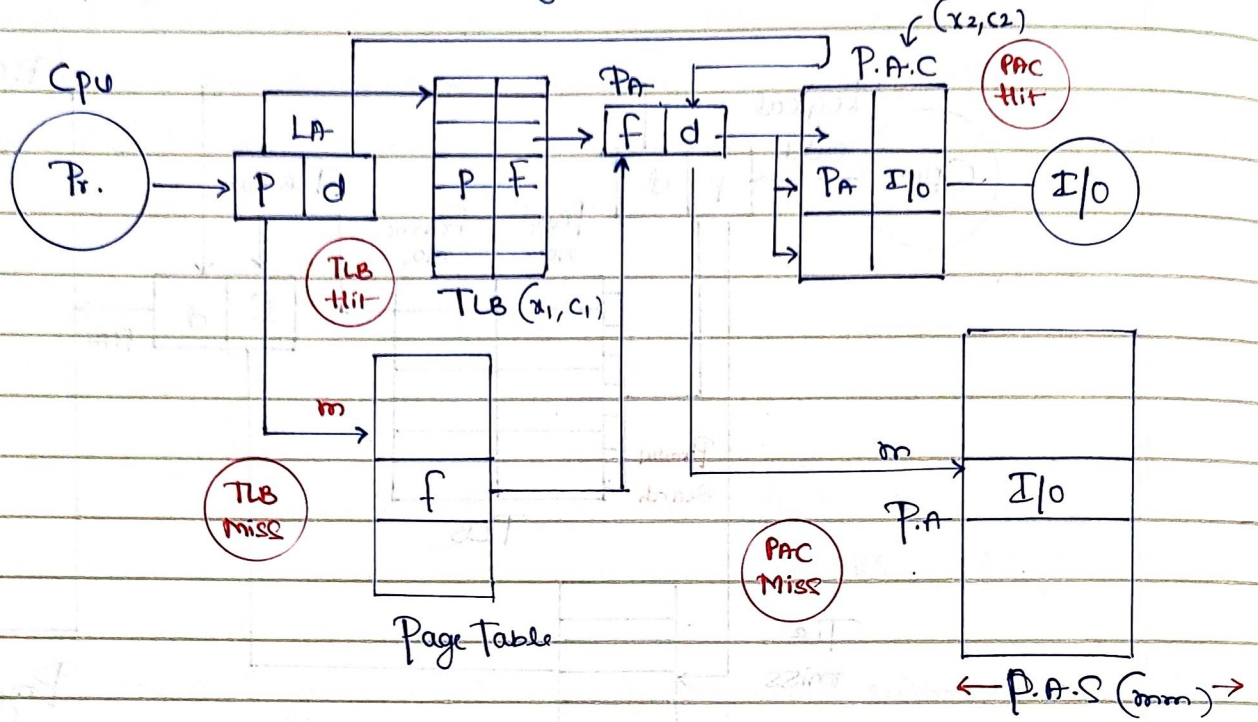
= 0.9 (120) + 0.1(220) = 108+22 = 130 ns [200 - 120]

(c) If x = 10·1 what is EMAT TLB?

= 0.1 (120) + 0.9 (220)

= 210 ns [200 - 210 ns]

Paging with TLB and Physical Address Cache (PAC)



Q2 Consider a system using paging with TLB. What hit ratio is required to reduce the EMAT from 'D' to '2' ns using TLB. Assume that TLB access time is 'k' ns.

$2m = D$

$m = D/2$

$EMAT = x(C+m) + (1-x)(C+2m)$

$'Z' = x\left[K + \frac{D}{2}\right] + (1-x)[K+m]$

(2) Space Consumption in Paging

$PTS_{(By)} = N * e\ (By)$

$PTS \propto N$

eg: LA = 32 bits, PS = 4KB

Appropriate PT size ?

$N = 2^{32}/2^{12} = 2^{20}$

Objective: Reduce Page table size of processes.

if e = 4B

PTS = 1m × 4B = 4MB

If system has 100 processes, amount of memory required = 100 × 4 = 400MB.

Reduce P.T Size | Associate Smaller PTs with processes.

* P.T.S = N * e

P.T.S $\propto$ N ——(i)

* P.T.S $\propto$ N $\propto \frac{1}{PS}$

* $N = \frac{LAS}{PS}$

$N \propto \frac{1}{PS}$ ——(ii)

$$P.T.S \propto \frac{1}{PS}$$

(I.) Reduce P.T.S by increasing Page Size.

Optimal Page Size?

Let V.A.S = 'S' Bytes
Let P.T.E = 'e' Bytes
Let Page Size = 'P' Byte.

(I). P.T.S = $\left[\left(\frac{S}{P}\right) * e\right]$ Bytes

(II). If = $\frac{P}{2}$ Bytes.

Total overhead = $\left[\left(\frac{S}{P}\right)e + \frac{P}{2}\right]$ — (1)

diff eqn (1) w.r.t. P — 0

$$\left[\frac{-1}{P^2} Se + \frac{1}{2}\right] = 0$$

$$\frac{Se}{P^2} = \frac{1}{2} \Rightarrow P^2 = 2Se$$

$$P = \sqrt{2Se}$$

Q1. A machine has a 32 bit address space and an 8KB page. The page table is entirely in hardware, with one 32-bit word per entry. When a process starts, the page table is copied to the hardware from memory, at one word every 100 nsec. If each process runs for 100 msec (including the time to load the page table), what fraction of CPU time is devoted to loading the page tables?

$$N \text{ pages} = \frac{2^{32}}{2^{13}} = 2^{19}$$

Time to load P.T = $2^{19} \times 100$ ns.

% Fraction of CPU time = $\frac{2^{19} \times 100 ns}{100 ms} = \frac{2^{19} \times 100 \times 10^{-9}}{100 \times 10^{-3} s} \Rightarrow 50\%$

Q2. Consider a system using paging technique with an address space of 65,536 Bytes. The page size in this system is 4096 Bytes. The program consists of Text, Data and Stack sections as per the space:

Text: 32,768 B
Data: 16,386 B
Stack: 15,870 B

A page of the program contains portion of only one section i.e either Text or Data or Stack

(a) Does program fit in given address space? No

(b) What is the maximum page size in Bytes such that the program fits in given address space.

VAS = 65,536 = 64KB

$N = \frac{64KB}{4KB} = \frac{2^{16}}{2^{12}} = 2^4 = 16$

Text = 32768/4096 = 8
Data = 16386/4096 = 5
Stack = 15,870/4096 = 4
Total = 17 = So you cannot.

(II) Hashed paging / Paging with Hashing

* Element(x) → h(f(n)) → 'i' → index
* We will design a Page table using hashing

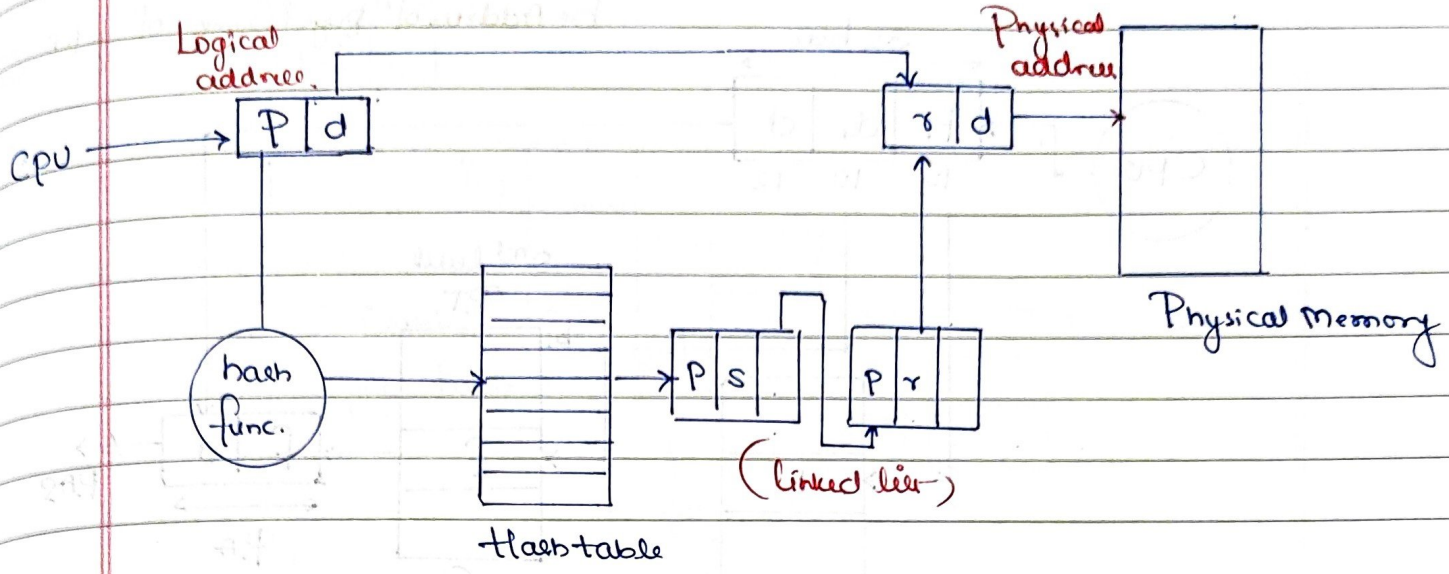
[x % 10] → 0 : 9

Collision: Let I_1 and I_2 be two distinct elements

$f(I_1) = f(I_2) = i$

* To resolve collision → Chaining is used.
* Space optimized, but time inefficient

Hashed page table



(III) Multi Level Paging / Hierarchical Paging (Recursive paging)

Objective: To reduce page table size overhead by associating smaller page table's with process.

eg:: VA = 32 bit
P.S = 4 KB
e = 4B
PTS = 1MB × 4KB
= 4MB

When do we say P.T is small?
:- If the page table fits in one frame of memory

Paging as a Concept involves 3-Steps:

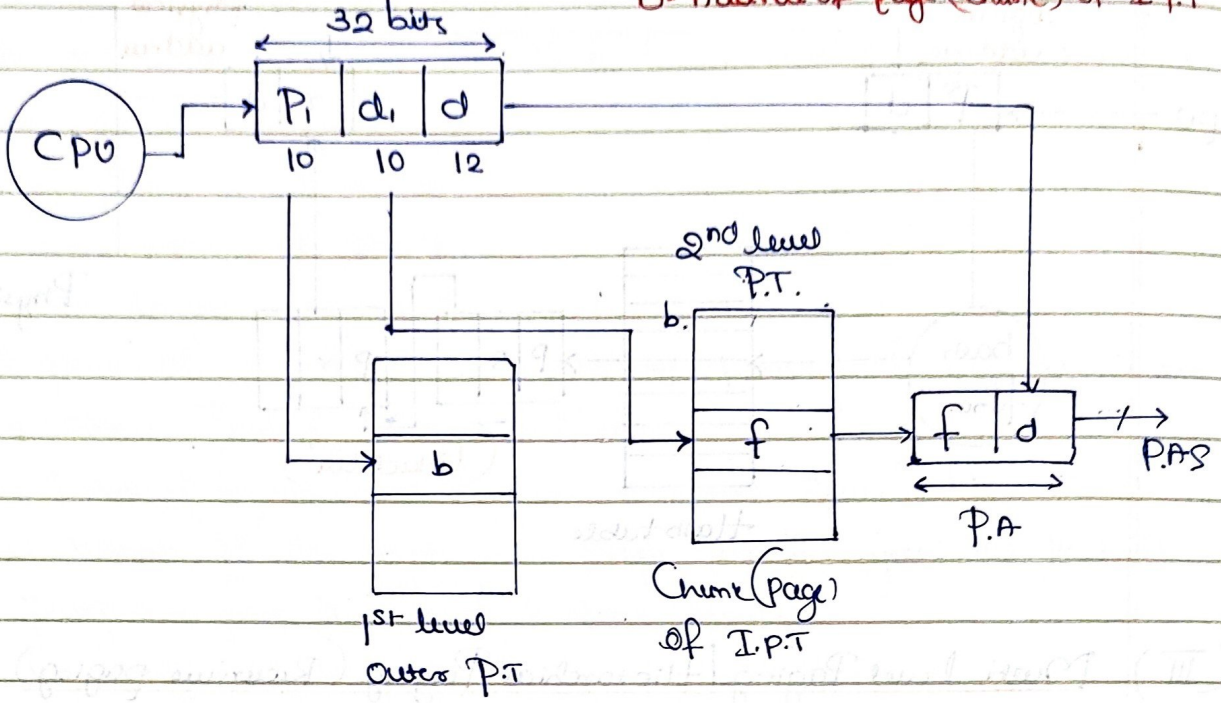
1. Divide the address space into page. (chunks).
2. Storing the page in physical address space
3. Access the page (chunks) of address space in physical address space through page table

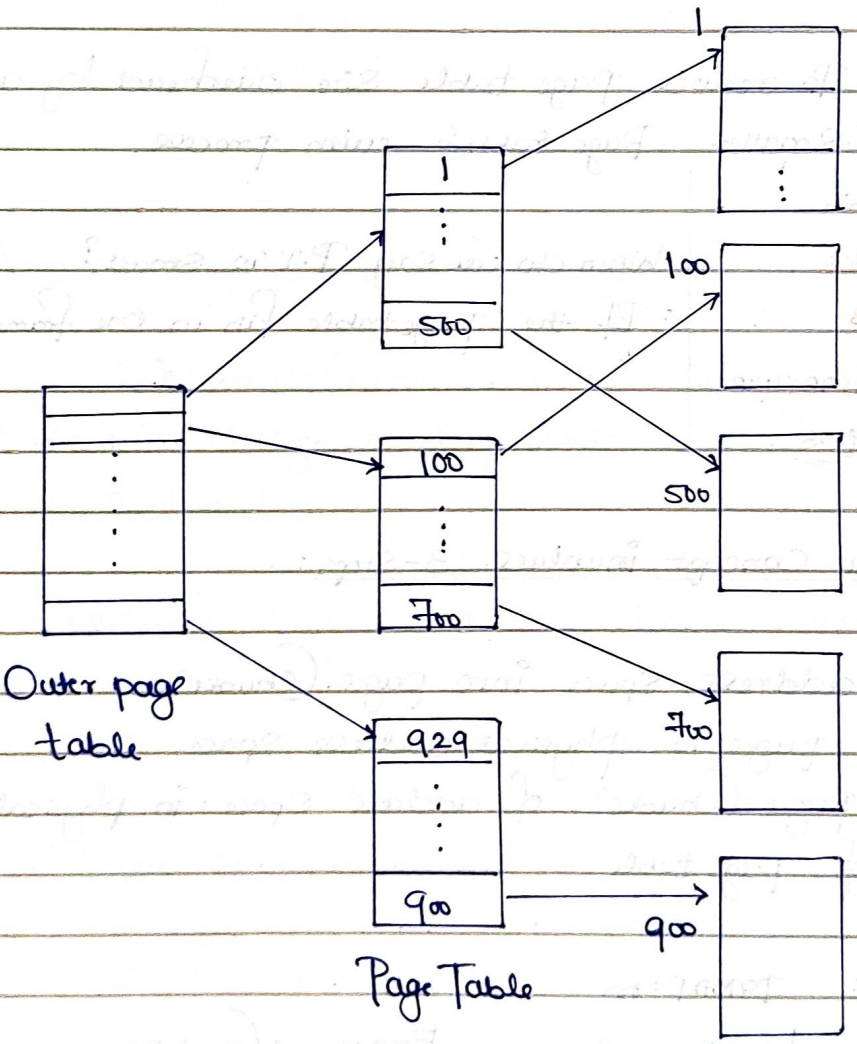
Performance: $EMAT = m$

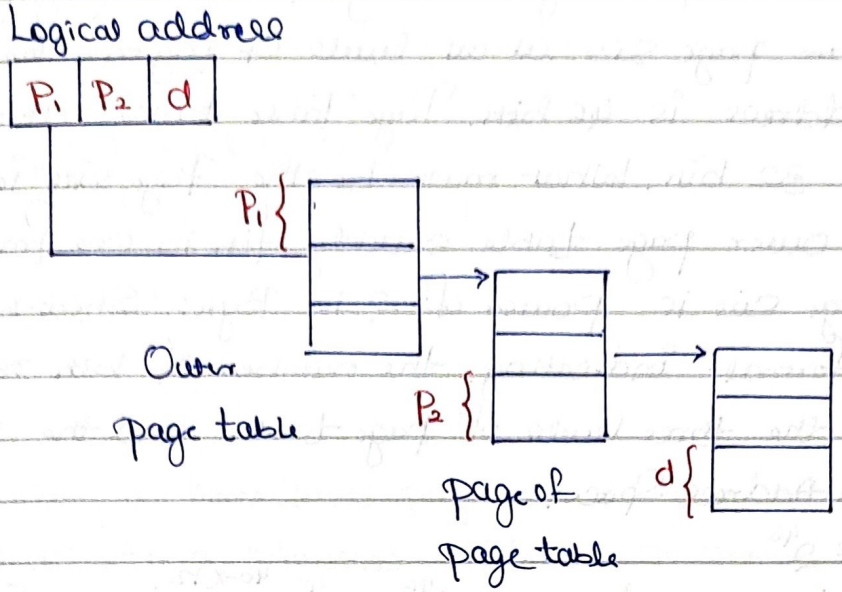
$EMAT_{2LP} = 3m$ $\quad EMAT_{nLP} = (n+1).m$

eg:: LA : 32 bits, P.S = 4 KB.

b = Address of page (chunk) of I.P.T







Q.1 In the context of operating systems, which of the following are correct w.r.t. paging?

* Paging helps solve the issue of external fragmentation
* Paging incurs memory overheads.

Q2. Consider a system using 2 level paging Architecture. The top level 9 bits of the virtual address are used to index into the outer Page Table. The next 7 bits of the addresses are used to index into next level page table. If the size of virtual address is 28 bits. Then

(i) How large are the pages and how many are there in virtual address space?

P → No of page in VAS , $N = 2^{16} = 2^6 \times 2^{10} = 64KB$

P.S of VAS $= 2^{12} = 4KB$

(ii) If P.T.E at both levels is 32 bits in size then what is the space overhead needed to translate virtual address to Physical address of an instruction or Data Unit?

P1	P2	d	
9	7	12.	2KB + 512B
			2.5KB

Q3. Consider a Computer System using three level paging Architecture with uniform page size at all levels of paging. The size of virtual address is 46 bits. Page Table Entries at all levels of paging is 32 bits. What must be the Page size in Bytes such that the outer page table exactly fits in one frame of memory. Assume page size is power of 2 in Byte. Show the virtual address format indicating the number of bits required to process all the three levels of page tables and the Page offset of virtual Address Space.

$VAS = 2^{46}$

Let $P.S = 2^x B \rightarrow \left(\frac{2^{46}}{2^x}\right) = [2^{46-x+2}]_B \rightarrow \left[\frac{2^{46}}{2^x}\right]^2_B$

$\Rightarrow [2^{46-3x+2+2+2}]_B$

$x = 13$

$\therefore 2^{13} = \underline{8KB}$

Derivation:

Let $VAS = 2^S$ Byte $VA = S$ bits

$P.S = 2^x$ Byte, $P.T.E = 2^c$ Byte

Levels of paging = 'l'

$\rightarrow$ Size of outer level $P.T = [2^{S-l.x+l.c}]$ Byte.

$\rightarrow \frac{2^S}{2^x} \cdot 2^c = [2^{S-x+c}]$

Q4 Consider a computer System with 57 bit virtual Addressing, using multi level tree Structured page tables with L levels for virtual to Physical address translation. Page size is 4KB and Page Table Entry is of 8 Byte at all level.

The value of L is $\underline{5}$

VA = 57 bit P.S = 4KB OPT = 1 Page / Frame

No. of level = 'L' P.T.E = 8B OPT = 4KB.

$$OPT = \left[2^{57-l \cdot 12+L3} \right] \text{ Bytes.}$$

$$= 2^{12} B$$

$$57 - 9l = 12$$

$$l = 5$$

HW

Qs. Consider a three level page table to translate a 39-bit virtual address to a physical address as shown below.

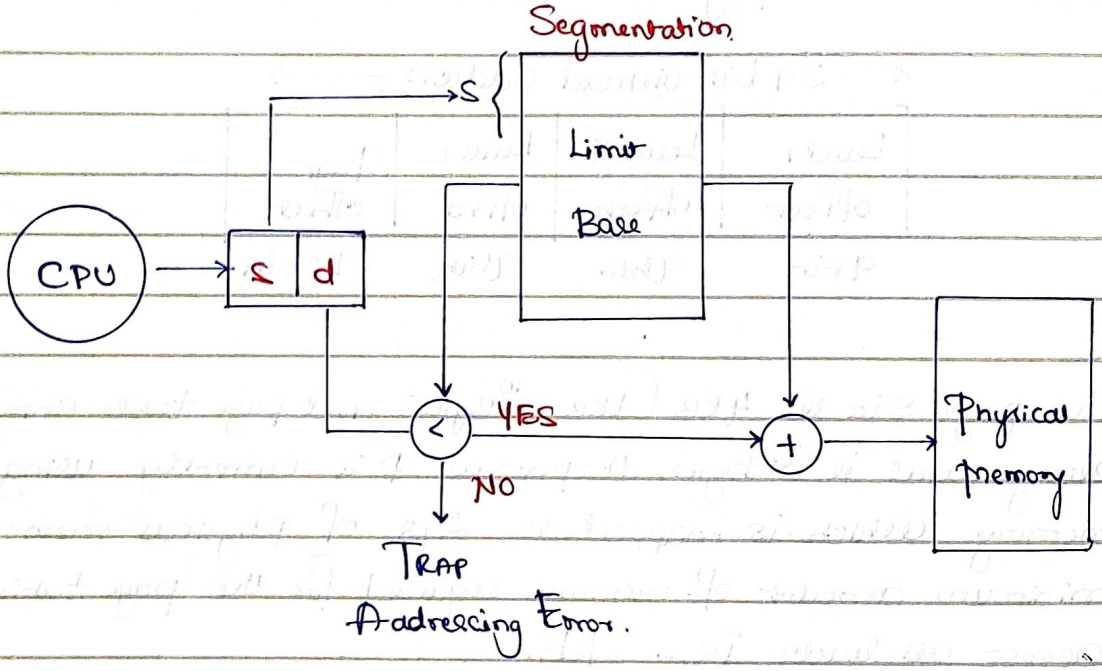
← 39 bit virtual address →

Level 1 offset	Level 2 offset	Level 1 offset	Page offset
9 bits	9 bits	9 bits.	12 bits

The page size is 4KB (1KB = 2^{10} Bytes) and page table entry size at every level is 8 bytes. A process P is currently using 2GB virtual memory which is mapped to 2GB of physical memory, the minimum amount of memory required for the page table of P across all levels is ___ KB.

2. Segmentation.

* Paging does not pressure user's view of memory allocation to programs
* As per user's view of memory allocation, Program is divided into logical units, known as segments (function, block, procedure, D.S, class).
* These segments are assumed to be stored in their activity entirely at non contigous locations.



HW

Q1. Consider the following segment table:

Segment	Base	Length
0	1219	600
1	3300	14
2	90	100
3	2327	580
4	1952	96

What are the physical address for the following logical addresses?

Performance of Segmentation

1. Time : MMAT = 'm' TLBAT = C
   EMAT = '2m' TLB Hit ratio = α
   TLB Miss ratio = $1-\alpha$
   $EMAT = \alpha(C+m) + (1-\alpha)(C+2m)$

2. Space :
   When segment table become large, apply paging on segment table

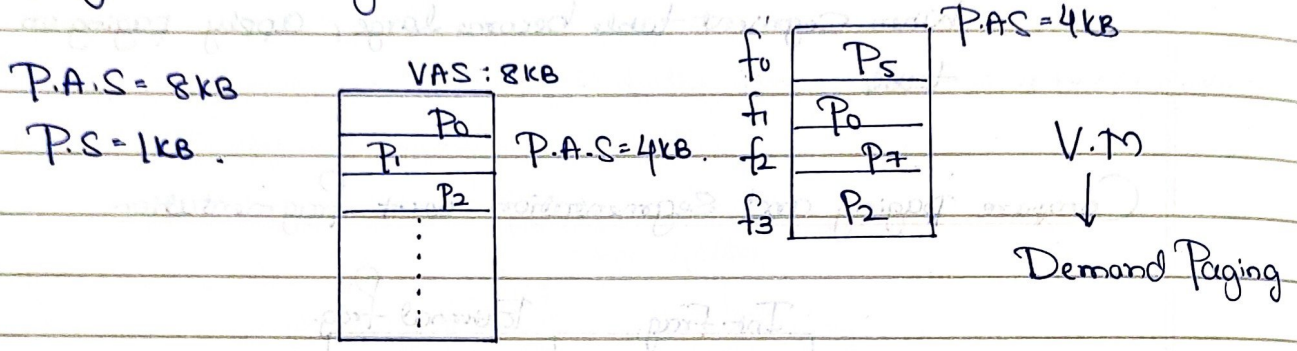
Compare Paging and Segmentation w.r.t fragmentation

	Int. Frag.	External frag.
Paging	✓ <last page>	X
Segment	X	✓ in P.A.S.

Segmentation
↓
External fragmentation
↙ ↘
1. Compaction
De-fragmentation.
2. Paging
↓
(Segmented-paging)

Virtual Memory (VM)

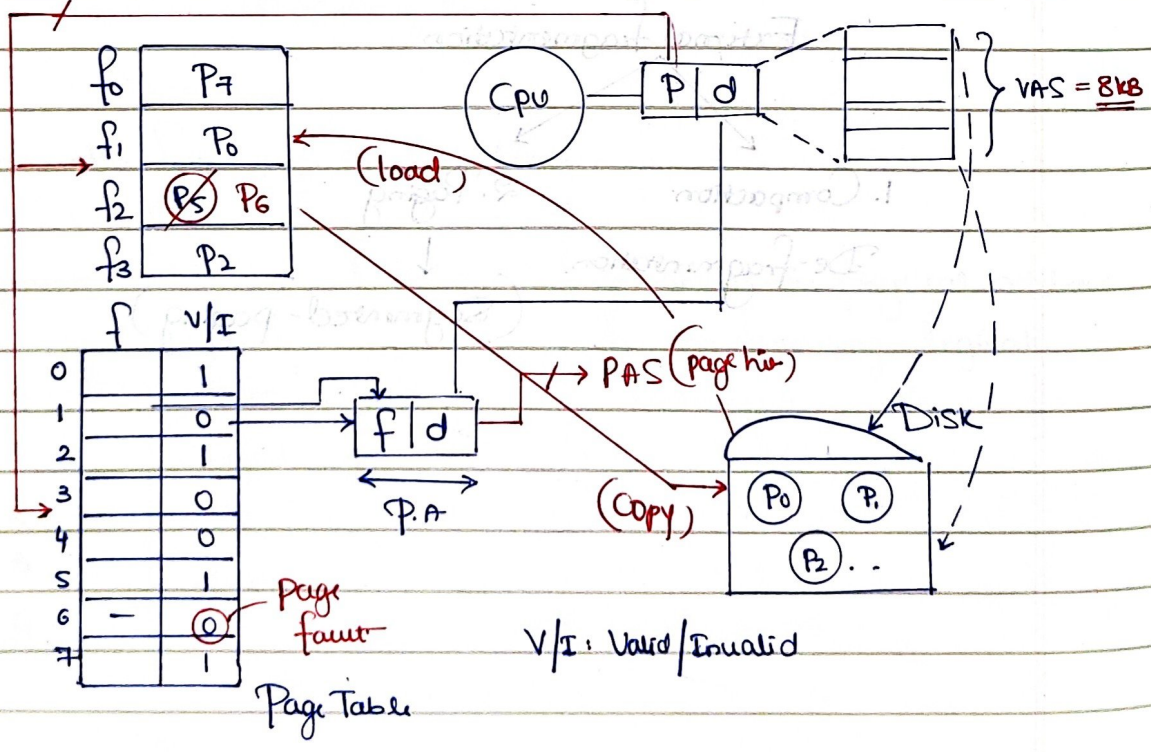
Virtual memory gives an illusion to the programmer that a huge / large amount of memory is available for executing programs, that are larger than the available / given physical memory



Demand paging : * Loading the page from disk to memory on demand at runtime.

* Virtual memory is implemented through "Demand Paging".

* Program (8KB / 8 Pages) : stored initially on disk, required pages are loaded in P.A.S.



Demand paging

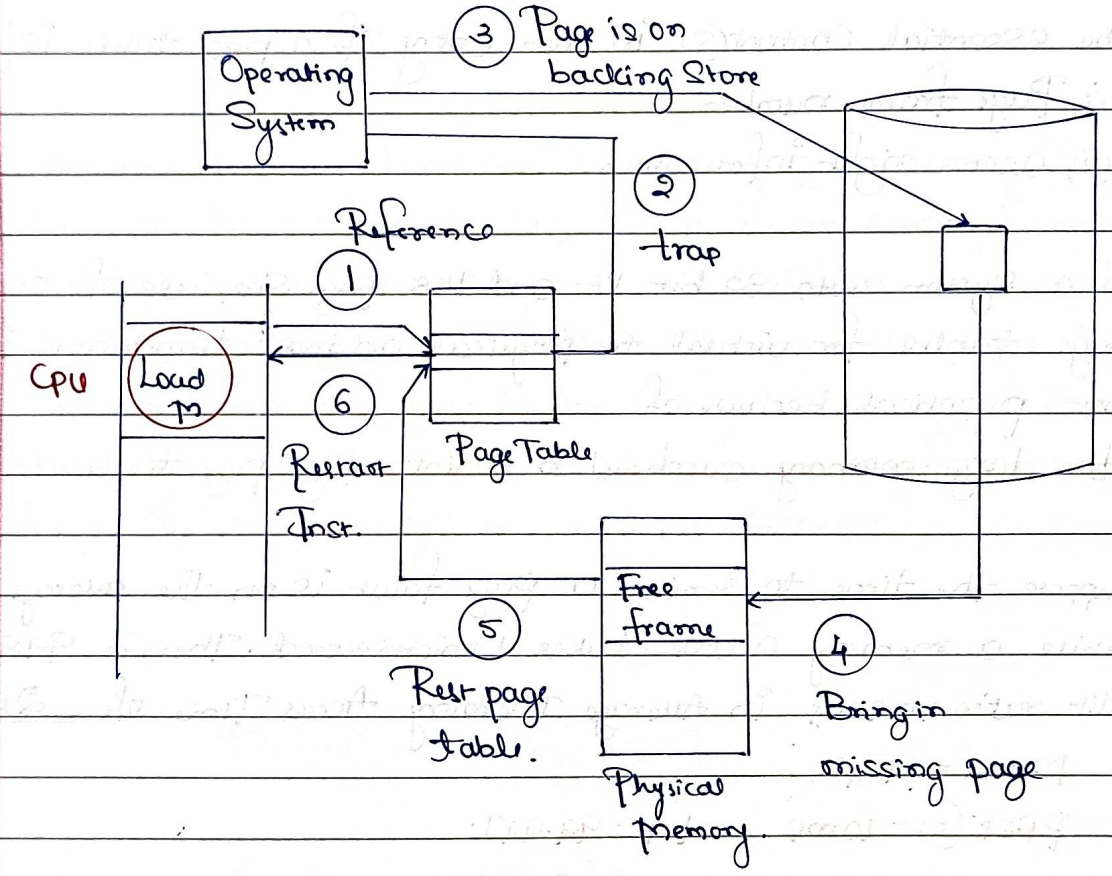
Pure DP — Pre Fetched D.P

< Execution of the process will start with empty frame >

Note: * Size of virtual memory is limited by size of disk.

* P.A.S ≤ VAS ≤ Disk AS.

Steps taken to service page fault:



* Amount of time taken by O.S (VMM) to service a page fault is known as "Page fault service time." (PFST)

Q1. In a virtual memory system the address space specified by the address lines of the CPU must be Larger than the physical memory size and Smaller Secondary Storage size.

Performance of Virtual Memory:

1. Time issue (Temporal)

M.M.A.T = 'm' [ns | µs]

PFST = 'S' [S >> m]

Page fault rate = 'p'

Page hit rate = '1-p'

EMAT = (1-p)*m + p*S

D.P

Q2. The total size of address space in virtual memory system is limited by:

Ans. Size of secondary storage

Q3. The essential content(s) in each entry of a page table is/are

Ans (i) Page frame number

(ii) access right information

Q4. In a system with 32 bit VA and 1KB page size, use of one level page tables for virtual to physical address translation is not practical because of

Ans the large memory overhead in maintaining page tables

Q5. Suppose the time to service a page fault is on the average 10 ms, while a memory access takes 1 microsecond. Then a 99.99% hit ratio results in Average Memory Access Time of 2µs

Ans MMAT = 1µs

PFST (S) = 10ms 1-p = 99.99% = (0.9999)

p = 0.01% = 0.0001.

EMAT = (1-p) m + p*S

= 0.9999 x 1x10^-6 s + 0.0001 x 10x10^-3 s

= 0.9999 x 1x10^-6 + 1x10^-6

= 1.9999 ≈ 2 µs ~ 2µs

Q6. If an instruction takes 'i' microseconds and a Page fault takes an additional 'j' microseconds, the Effective Instruction Time if on the average a page fault occurs every 'k' instructions is _____

Ans Instruction time without Page fault : 'i' μs

Instruction time with Page fault : 'i+j' μs

$P = 1/k$

$(1-P) = \left(1 - \frac{1}{k}\right)$

Effective Inst. Time : $\frac{1}{k}(i+j) + \left(1-\frac{1}{k}\right)i$

$= \frac{i}{k} + \frac{j}{k} + i - \frac{i}{k}$

$= i + j/k$

HW

Q7. Assume that we have a Demand-paged memory. It takes 8 milli-seconds to service a page fault if an empty frame is available or if the rejected page is not modified, and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nano sec. Assume that the page to be replaced is modified 70 percent of the time. What is the acceptable page fault rate for an effective access time of no more than 2000 nanoseconds?

Ans

30%

S ⟶ 8(ms) : Ef + Clean page [70%]

⟶ 20 (ms) : Dirty page [30%]

70%

m = 100 ns

P = ?

EMAT = 2000 ns

$2000\,ns = (1-P)*100\,ns + P(0.7*20\,ms + 0.3*8\,ms)$

P =

Q8. Consider a process executing on an Operating System that uses demand paging. The average time for a memory access in the system is M units if the corresponding memory page is available in memory, and D units if the memory access causes a page fault. It has been experimentally measured that the average time taken for a memory access in the process is X units. Which one of the following is correct expression for the page fault rate experienced by the process?

Ans $m = M$ $\quad X = (1-P)m + P*D$

$s = D$ $\quad = P = \frac{(X-m)}{(D-m)}$

Q9. Consider a paging system that uses 1-level page table residing in main memory and a TLB for address translation. Each main memory access ~~time~~ takes 100 ns and TLB lookup takes 20ns. Each page transfer to/from the disk takes 5000 ns. Assume that the TLB hit ratio is 95%, page fault rate is 10%. Assume that for the 20% of the total page faults, a dirty page has to be written back to disk before the required page is in read in from disk. TLB update time is negligible. The average memory access time in ns (round off to 1 decimal place) is 154.5

Ans m — 100ns $\quad$ Disk R/W — 5000ns

c — 20ns $\quad$ TLB hit rate = 95%

Page fault rate = 10%

80% — Clean page

20% — Dirty page

EMAT = 0.95 [c (TLB hit) + 20ns + 100ns] + 0.05 [20ns + (TLB miss)

0.9 (100ns + 100ns) + 0.1 (P.F) page hit

(100ns + 0.2 x (5000 + 5000ns) + (D.P.)

0.8 (5000 ns)] (Clean page)

EMAT = 154.54 ≈ 155

Q10. The minimum number of page frames that must be allocated to a running process in a virtual memory environment is determined by

Ans Instruction Structure Architecture.

Page Replacement : one important concept of Virtual Memory implement.

(a) Reference String (Page-Ref string) : Set of successively unique pages (Page Nos) referred in given list of V.A's.

eg:: (Pi) < 702, 704, 123, 654, 483, 012, 122, 124, 180, 785, 934, ... >

< 7, 1, 6, 4, 0, 1, 7, 9, 6, 7, . >

Let P.S = 100 B

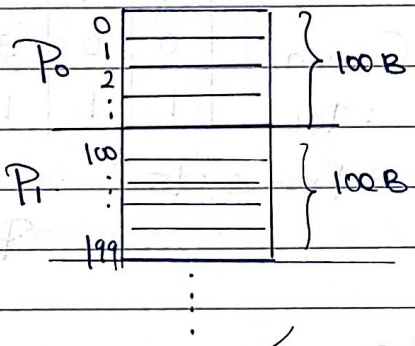
$P = \frac{V.A}{PS}$ $d = V.A \% PS$

Length = 10

Unique pages = 6

P7 (702)

(P) 7 (d) 2



<0...99> : P0

<100...199> : P1

<200...299> : P2

(b) Frame allocation Policies

* n = no. of processes (P1...Pn)
* Si = Demand (frames) of process Pi
* D = Total Demand $= \sum_{i=1}^{n} Si$
* m = Available frames
* ai = Allocated frames

n = 5 <P1...P5> ; m = 40

Pi	Demand Si	Eq. alloc. ai = m/n	Pro.p alloc ai = (Si/D)m	ai = Si/2
P1	10	8	5	
P2	5	8	3	
P3	35	8	17	
P4	18	8	9	
P5	12	8	6	

D = 80

* Min. no of frames to be allocated to a process:
→ < Process cannot execute without this minimum number of frames >
→ Process should be able to execute minimum one instruction.

Page Replacement Techniques:

1. Ref string : < 7; 0; 1; 2; 0; 3; 0; 4; 2; 3; 0; 3; 2; 1; 2; 0; 1; 7; 0; 1 >
length = 20, n = 6

(1) FIFO

	7	2	2	2	4	4	4	0	0	0	7	7	7
Criteria : TOL	0	0	3	3	3	2	2	2	1	1	1	0	0
	1	1	1	0	0	0	3	3	3	2	2	2	1.

3 Frames : 15 — frames —

$P = \frac{18^3}{20_4} = 75\%$

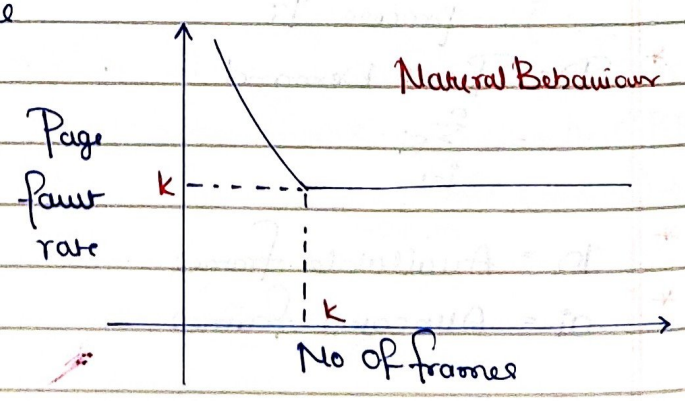
eg:: 2 Ref string: < 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 >
FIFO : pure Demand paging

hints 3 frames:

~~1~~	4	4	~~4~~	5	5	5
2	~~2~~	1	1	~~1~~	3	3
3	3	~~3~~	2	2	~~2~~	4

3 Frames: 9
4 Frame : 10
(~~HW~~)

Frames



Belady's Anomaly :

As the number of frames allocated to the process increases, page fault also sometimes increases.

Only FIFO and FIFO based suffers from Belady Anomaly.

(2) Optimal Replacement : replace that page which will not be used for the longest duration of time in future references.

eg:: Ref string <7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1>

3 Frames : 9
4 Frames : ?
(HW)

7	2	2	2	2	2	7
0	0	0	4	0	0	0
1	1	3	3	3	1	1

Frames.

* We use this algorithm as a benchmark to check performance of other replacement strategies. Optimal replacement is not implemented practically.

(3) Least Recently used (LRU) : replace that page which has not been used for the longest duration of time in the past.

Criteria : Time of Refrence.

eg:: <7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 7, 7, 0, 1>

3 Frames : 12.
4 Frames : ?
(HW)

7	2	2	4	4	4	0	1	1	1
0	0	0	0	0	3	3	3	0	0
1	1	3	3	2	2	2	2	2	7

(4) Most Recently used (MRU) : Smyttrical opposite of least recently used, replace that page which has been used for longest duration of time.

HW eg:: <7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1>

3 Frames:

4 Frames:

(5) Counting Algorithms:

a> L.F.U [least frequently used]

b> M.F.U [most frequently used]

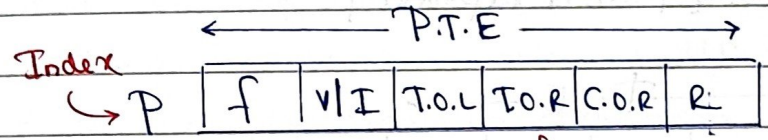
*Criteria: Count of reference.

eg:: <7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1>

(HW: Solve with 3 frames (both)!)

Note: * Among all the algorithms, Optimal is having least page fault rate.
* LRU ~ Optimal
* Many OS implement either LRU / LRU approximately
* The best method to implement LRU is by "stack method."

LRU Approximations: <These algorithms are not really LRU, but they approximate to the behaviour of LRU>.
They are based on a concept called <Reference bit (R)> : Each page in the page table will be associated with a reference bit 'R'.



a.> Reference bit (R):

Criteria: 'R' <
0 : Page is not referred so far during present Epoch.
1 : Page has been referred atleast once during present Epoch.

eg::

	P	f	V/I	T.O.L	R
	(0)	a	1	2	1
	1	e	1	3	0
	2	d	1	0	1
P.F	(3)	–	0	–	–
	4	c	1	4	10
	(5)	b	1	1	0

Page Table

Reference bit fails when all R values are '1'.

b>. Additional Reference bits: Each page is associated with more than one reference bits.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8 ↓ Current Epoch
P_i	1	0	0	1	0	1	1	1
P_j	0	0	1	1	1	0	1	1
P_k	0	0	1	1	0	1	1	1

(P_i) X : Cause Page fault

When current Epoch gets over

↓

[Shift left operations]

c). Second Chance [clock algorithm]

Criteria : [T.O.L + Reference]

eg::

← P.T.E →

P	V/I	T.O.L	R	f
0	1	2	1	e
1	1	3	0	a
2	1	0	X	b
3	0	–	–	–
4	1	1	0 ✓	c
5	1	4	1	d

Page Table

When all pages 'R' value is '1' then FIFO page gets selected

* Second ~ FIFO Chance

↓

Belady's Anamoly

d). Enhanced Second Chance / NRU (Not recently used).

Criteria: Reference + modified

RM

I. 00 → Not referenced and not modified

II. 01 → Not referenced but modified

III. 10 → Referenced but clean

IV. 11 → Referenced and modified

eg::

← P.T.E. Structure →

P	f	V/I	T.o.L	R	m
0	c	1	4	1	1
1	a	1	3	0	0
2	-	0	-	-	-
3	b	1	0	1	1
4	d	1	2	1	0
5	e	1	1	0	1

(i) FIFO: P_3
(ii) R: P_1
(iii) Second Chance: P_5
(iv) Enhanced Second Chance: P_1

Q1. Consider a system with V.A.S = P.A.S = 2^{16} Bytes. Page size is 512 Bytes. The size of the page table entry is 32 bits. If the page table entry contains besides information 1 V/I bit, 1 Reference, 1 modified bit, 3 bits for page protection. How many bits can be assigned for storing other attributes of the page. Also compute page table size in Bytes?

Ans:

← P.T.S = 32 bits →

P	f	V/I	R	m	Pr.	Other.
	7	1	1	1	3	2

$\#frames = \frac{2^{16}}{2^9} = 2^7$

P.T.S = N x e
$= 2^7 \times 2^2 B = 2^9 B \rightarrow 512B$

Q2. Consider a virtual memory system with FIFO page replacement policy, for an arbitary page access pattern. Increasing the number of page frames in main memory will.

Ans Sometimes increases the no. of page faults (Belady's Anamoly)

Q3. A memory page containing a heavily used variable that was initialized very early and is in constant use is removed when

Ans. FIFO page replacement algorithm is used.

Q4. Recall that Belady's anamoly is that the page fault rate may increase as the number of allocated frames increases. Now consider the following Statements:

S1: Random page replacement algorithm (where a page chosen at random is replaced) suffers from Belady anamoly

S2: LRU page replacement algorithm suffers from Belady anamoly.

Ans: S1 is true and S2 is false.

Q5 Consider a process having reference string of length 'l' in which n - unique pages occur. 'z' frames are allocated to the process. Calculate the lower bound and upper bound of the no. of page faults.

a) maximum = 'l' [z=1]
(Upper bound)

b) minimum (1) = 'n' [z≥n] - if we use pure demand paging
(Lower bound)

= 0 - if we use pre-fetched demand paging

Thrashing: Excessive / high paging activity (high page fault rate)
↓
(The act of causing a page-fault leading to loading and saving the pages on disk.)

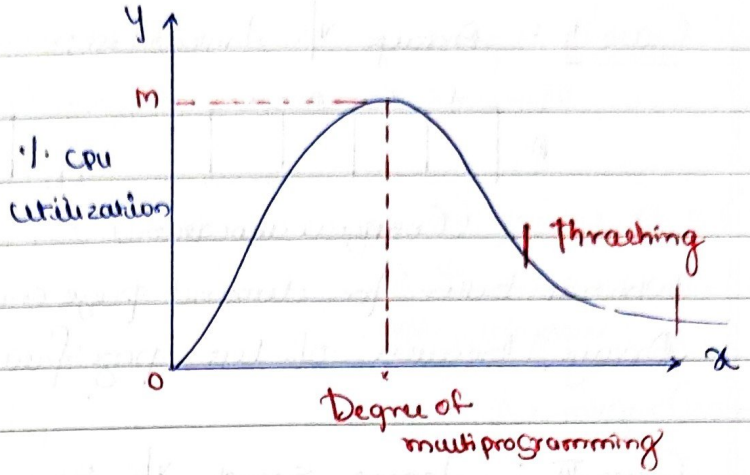
* Thrashing like deadlock is also an undesirable feature of O.S.

Reasons for Thrashing: (main)

1. High degree of multiprogramming
2. Frame allocation decline <lack of memory>

Other Reasons:

1. Page replacement policy
2. Page size:
   large: (less pages) → Less page faults
3. Programming techniques and data structures



Thrashing Control Strategies

1) Prevention

* Thrashing never occur.
* By controlling degree of multi-programming, thrashing can be controlled.
  ↓
  (Long term Scheduler)

2) Detection and Recovery

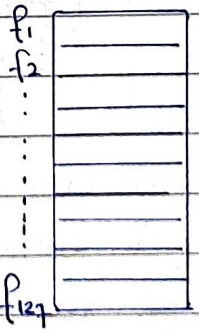
* Thrashing occurs
* Thrashing can be detected when there is low CPU utilization.
* Max no. of processes getting blocked
* High degree of multi-programming
* High disk utilization
* Recovery: Process suspension
  ↓
  (Medium Term Scheduler)

Case I: integer A[1...128][1...128]; P.S = 128 w; Row major order

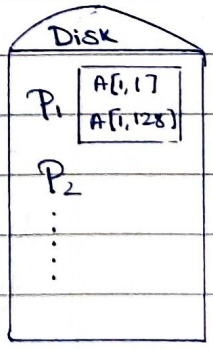
```
1. for i ← 1 to 128
     for j ← 1 to 128
       A[j, i] = 1;

2. for i ← 1 to 128
     for j ← 1 to 128
       A[i, j] = 1;
```

No of page faults:
a) Pure DP: 128^2
b) FIFO: 128

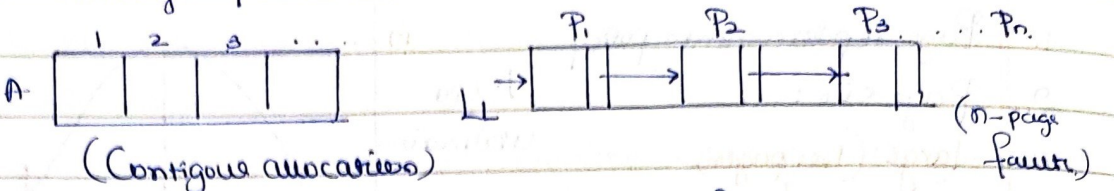
(Locality of Reference)



$128 \times 128 = 2^{14}$
$= 16k$



Case II : Array v/s Linked List



(Contigous allocation)

Q1. Which is better for demand page environment?

Ans Array, because of less page fault.

Case III : Linear Search v/s Binary Search

O(n) O(Logn)

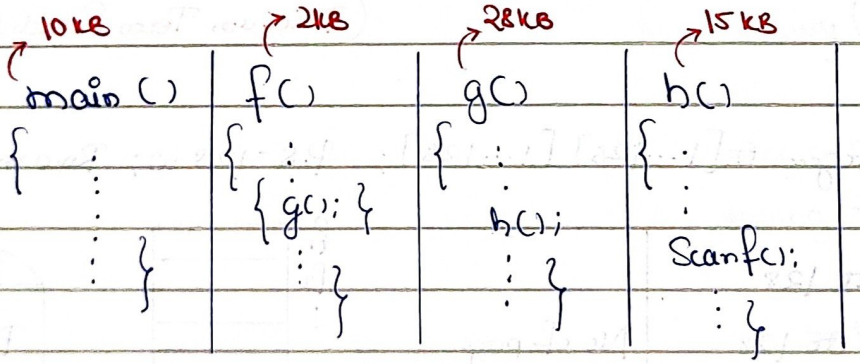
Q1. Which is better for demand page environment?

Ans Linear Search may generate less page fault, (locality of reference).

Conclusion : A programming technique or a data structure is said to be good in demand page environment, if it satisfies locality model.

Working Set Strategy (model) : To minimize page fault rate and also utilize memory effectively.

< Principle of locality of reference >



The basic idea of working set model is to estimate the size of locality in which the program is executing and only ask for those many frames.

W.S.S follows dynamic type memory allocation (Allocation of frames)

Working Set window (WSW) : Set of unique pages referred in the reference string during the past 'Δ' references.

eg:: Pri < 5, 4, 8, 2, 9, 12, 54, 58, 56, 53, 51, 52, 58, 56, 57, 54,

Δ=10 (Sliding window) t

WSW_i^t (Δ=10) = {51, 52, 53, 54, 56, 57, 58}

Δ = integer (value)

WSW_i^t (Δ=10) = 7

→ Let, No. of processes = 'n' <P1 ... Pn>
→ Total available frames = m
→ Demand of each = $S_i = WSWS_i^t$ (Δ=10)
→ Proces for frames
→ Total demand @'t' = $\sum_{i=1}^{n} S_i = D$

I. D ≤ m [no-thrashing]
II. D < m [no thrashing → Inc. degree of multi-programming]
III. D > m [System is trashing]

Δ

Small
* Leads to more page faults

large
* Inefficient utilization of memory

Q1. Let the page reference and the Delta (Δ) be "ccdbceceead" and 4 respectively. The initial working set a time t=0 contains the pages {a, d, e, where 'a' was referenced at time = t=0, 'd' was referenced at time t=-1, and 'e' was referenced at time t=-2. Determine the total number of page faults and the average number of page frames used by computing the working set at each reference.

t	-2	-1	0	1	2	3	4	5	6	7	8	9	10
	e	d	a	c	c	d	b	c	e	c	e	a	d

$\Delta = 4$

$t_0 = \langle e\ d\ a \rangle$

$t_1 = \langle e\ d\ a\ c \rangle : 4$

$t_2 = \langle d\ a\ c \rangle : 3$

$t_3 = \langle a\ c\ d \rangle : 3$

$t_4 = \langle c\ d\ b \rangle : 3$

$t_5 = \langle d\ b\ c \rangle : 3$

$t_6 = \langle d\ b\ c\ e \rangle : 4$

$t_7 = \langle b\ e\ c \rangle : 3$

$t_8 = \langle c, e \rangle : 2$

$t_9 = \langle c\ e\ a \rangle : 3$

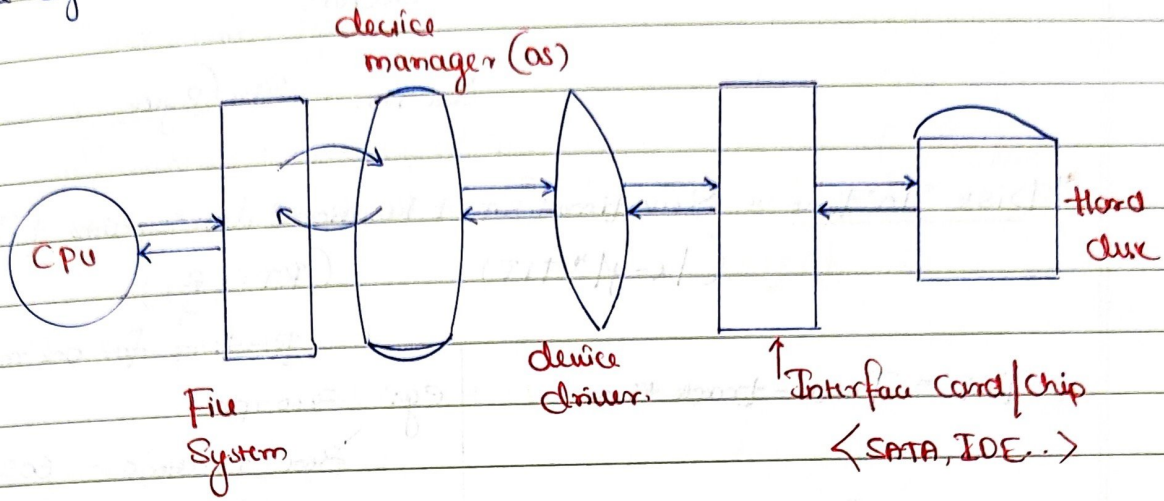
$t_{10} = \langle c\ e\ a\ d \rangle : 4$

No. of page faults = 5

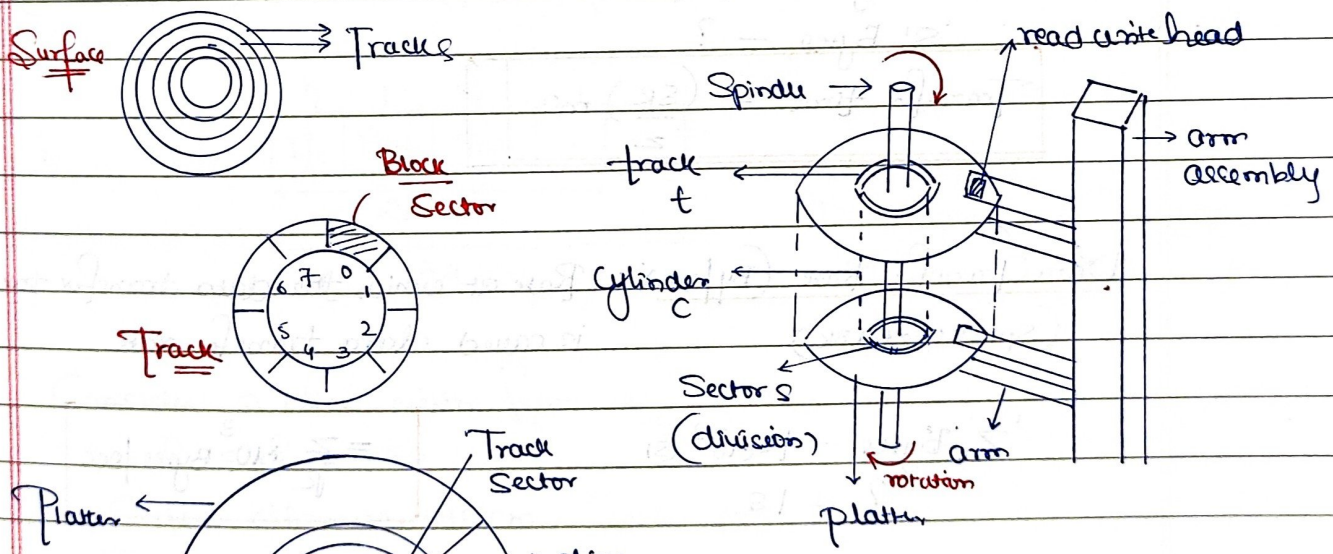
Avg. WSS$_0$ = $\frac{7}{10}$

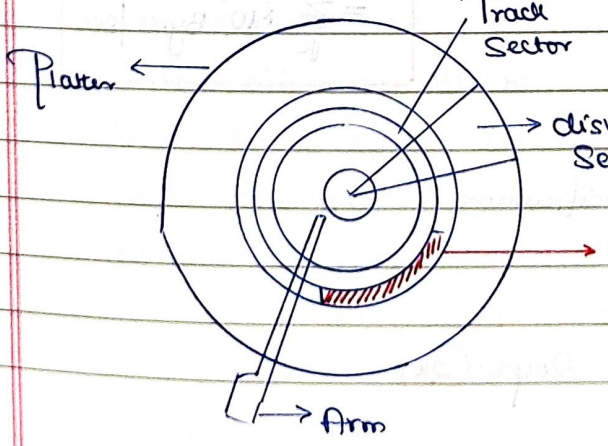
File System and Device Management

* File System is the only visible part of the operating System.



* Before storing the data on disk, it must be formatted in some file System (NTFS, ExFAT32, Ufs)
* Disk is a collection of circular/elliptical platters.
* Every platter has 2 ~~tracks~~ Surface, and consists of circular tracks.
* Each track consist of Sectors, Sector is measured in bytes





→ Cluster: a group of one or more adjacent Sectors

Disk → Platter → 2 surfaces → Tracks

Tracks → Sectors

Sectors → Sec-No. , Size (Bytes)

* **Disk IO Time** = Seek time (ST) + Rotational latency time + Transfer time

($|x-y| * TTT$) ($RT = (\frac{R}{2})$)

R = time for one rotation.

TTT = Track-track time

eg. 3600 rpm

3600 - rotations — 60s

1 rotation — ?

$R = \frac{60}{3600} = 16.67$ ms

→ Track Size = 'Z' bytes

→ Sector Size = 'S' bytes

→ Rotation time = 'R'

→ To one rotation, we can read one complete Track.

'Z' Bytes — R (ms)

'S' Bytes — ?

$$\text{Transfer time} = \frac{(SR)}{Z} \text{ ms}$$

Data Transfer Rate (By/sec) (speed of disk): Rate at which the data transfer takes place is called data transfer rate.

'Z' Bytes — $R * 10^{-3}$ (s)

? — 1s.

$$= \frac{Z}{R} * 10^{3} \text{ Bytes/sec}$$

Q1 Consider the following disk specifications

Number of platters = 16

No. of tracks/surface = 512

No. of Sectors/track = 2048 (2k)

Sector offset = 12 bit $= 2^{12} = 4KB$

Average seek time = 30 ms

Disk rpm = 3600

Calculate:

(i) Unformatted Capacity of the disk:

$= 32 \times 512 \times 2K \times 4KB = 2^5 \times 2^{11} \times 2^{12} = 2^{37}$ ($+2^5$ missed to write.)

$= 128\,GB$

(ii) IO time / Sector:

$ST + LT + TT = 30ms + 8.3ms + 0 = 39ms$

Track size = $2K \times 4KB$

$= 8MB$

$\left(\frac{4KB}{8MB} \times 16.6\,ms\right) \leftarrow TT.$

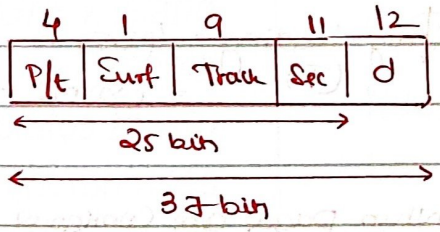
(iii) Data transfer rate:

$\frac{Ts}{8MB} - \frac{R}{16.6 \times 10^{-3} s}$

$? - 1s$

$\Rightarrow \frac{8MB}{16.6} \times 10^3 s. = 1/2\ GB/s$

(iv) Sector address = 25 bit required to select one sector.



Disk capacity = $2^{37} = 128\,GB$ (another approach)

HW

Q2. Consider a disk with following specs:

No of surfaces = 64

Outer diameter = 16 cm, Inner dia = 4 cm

Inner track space = 0.1 mm

Max density = 8000 bit/cm

Calculate the unformatted Capacity of disk.

No. of tracks = $\frac{6cm}{0.1mm} = 600$

Inner Track size = $2\pi r = \pi \times d$

$= 3.14 \times 4 = 12.56$ cm

8000 bits — 1cm

? — 12.56 cm.

Track size = $\frac{8000 \times 12.56}{8}$

$= 1000 \times 12.56$ B.

Disk capacity $= 64 \times 600 \times 12.56$ KB

$= 64 \times 6 \times 1256$ KB

$= 64 \times 6 \times 1.256$ MB

$= 482.3$ MB $= 0.48$ GB

$\therefore$ Disk Cap $= 0.48 \simeq 0.5$ GB

Q3. How long does it take to load a 64KB program from a disk whose avg. seek time is 30ms, Rotation time is 20ms, Track size is 32 Kbytes, Page size is 4Kbytes, Assume that pages of program are distributed randomly around the disk. What will be the % saving in time if 50% of the pages of program are contigous?

S.T = 30ms

R = 20ms

T.S = 32KB

P.S = 4KB

$N = \frac{64KB}{4KB} = N = 16$

Time to load program = T. to load a page * N

Time to load page = S.T + L.T + T.T

= 30ms + 10ms + 2.5ms

= 42.5ms

Prog-loading (T) = 42.5 x 16

= 680 ms = 0.68s

32KB — 20ms

4KB — ?

$\frac{4KB}{32KB} \times 20 = 2.5$ ms

When pages are contigous then only one seek and one latency:

= (30ms + 10ms + 2.5 x 8) + 8 x 42.5 ms

(CG) (NCG)

= 60ms + 34 = 400ms = 0.4s

(i) 0.68s

(ii) 0.4s = 0.28s (diff)

% Savings = $\left(\frac{0.28}{0.68}\right)$ = 41% percentage

Savings = 41

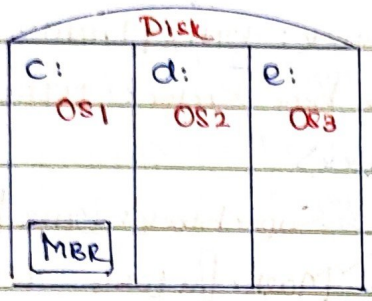
File System and Device Management (Continued)

Logical Structure of disk: [Formatting Process]

↳ Placing infrastructure on disk

↓

[for effective Storage and faster retrival.]



"Multi boot Computer".

→ Disk divided into partitions (Volumes / Mini disks)

(OS) Primary (Bootable) : Data + S/W

Extended / Logical drives (Non-bootable) : Data + S/W

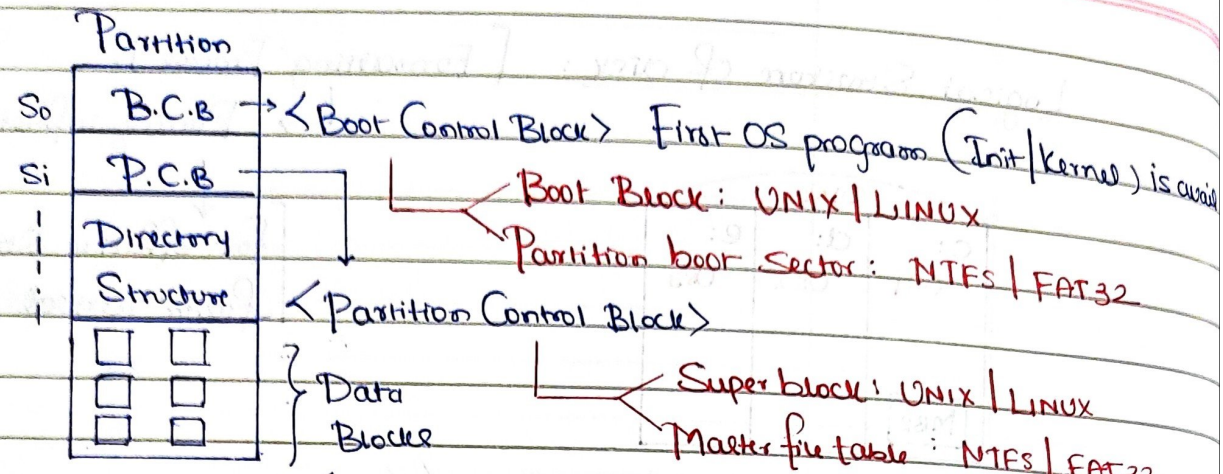
MBR <Master Boot record>

- Partition Table
- Boot Loader

Steps in Booting Process:

1. Power On or Switch on.
2. POST <Power on Self test>
3. BIOS <Basic IO System>
4. BootStrap loads MBR into RAM and hands over the control to boot loader
5. Boot loader reads partition table and display options if available
6. Boot loader will load the kernel program from disk into memory.

Partition Structure [What data structure (Infrastructure) are embossed on the disk/partition after formatting]



Disk is divided into 2 part: 1. Control data
2. User data

Some of the memory will be taken as memory overheads (B.C.B + P.C.B + D. Structure) that is the reason in Pendrives the size mentioned will be 128 GB but memory actual avail is 114GB.

File v/s Directory:

File: File is a collection of logically related records of entity. It is an abstract data type <Defination, Representation, Operations, Attributes>

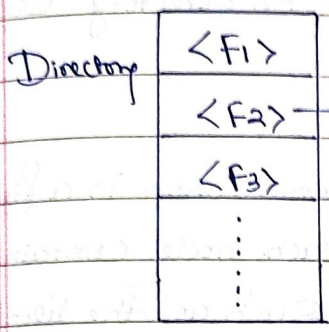
- Representation → Flat (Series of Bytes); Record (Series of records) — Hierarchical Tree (B-trees, B+..)

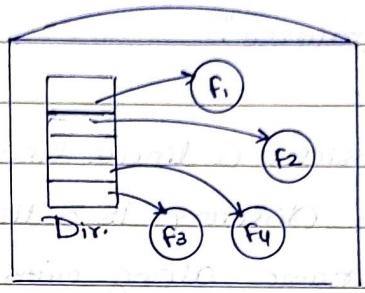
Operations: Create, Open, read, write, truncate, seek, close, copy, move, delete,..

Attributes: Name, identifier, type, protection, time, date, location... (different OS have diff attributes)

* File's attributes are stored in File Control Block (FCB)
  - UNIX (I-node)
  - Windows (Directory entry)

Directory: It is a collection of files. It is also a special file (data about other files → "Meta data")
Directory contains meta data of files.





(similar like pointers in C)

Directory Structure.

* "Single level directory"

directory	car	bo	a	test	data	mail	Cont	hex	records

(Single directory for all users)

* no sub directory
* Simple to implement
* Search time is more
* Name conflict.

* Two level directory
  * Seperate directory for each user
  * Path name
  * Can have the same file name for diff. user
  * efficient searching
  * no grouping capability

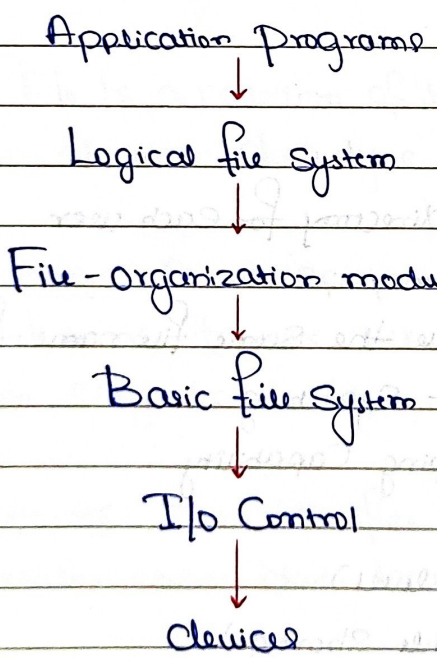
* Tree structured directories (Multi level)
* Acyclic graph directories (File sharing)
  - also called directed acyclic graph directory for file sharing purposes
  - Link count: Counts the no. of people sharing a file.
* General graph directory (Cycles)

* To search a file, we may have to traverse the directory, for that we may use Graph traversale like DFS, BFS.
* Directory as a file also supports operations like open, read, write but there is a special operation on directory called "traverse".

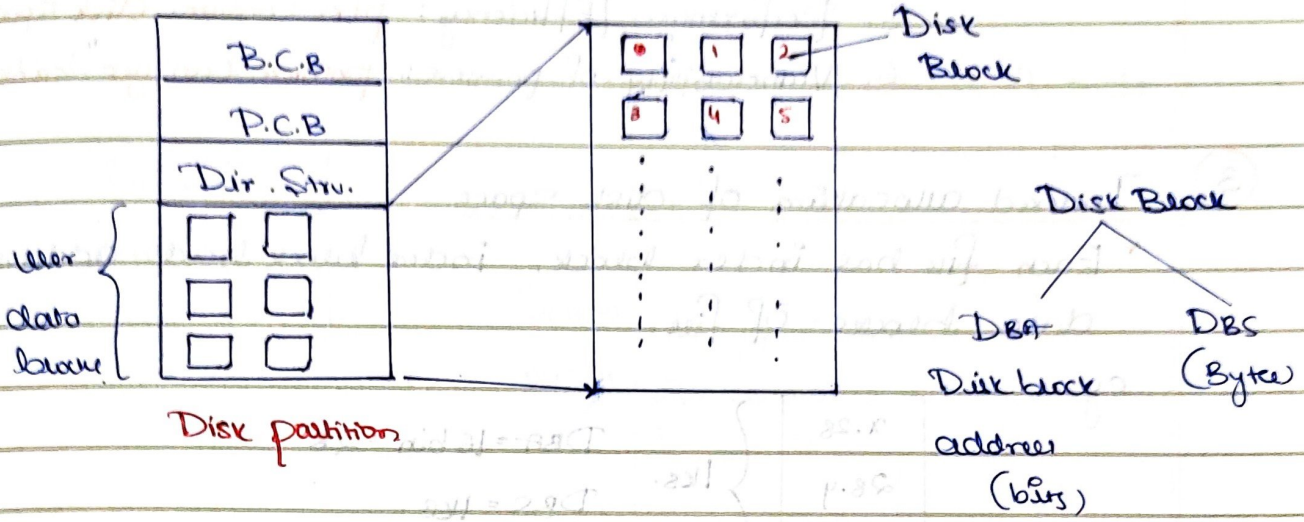
Q1. Consider a linear list based directory implementation in a file system. Each directory is a list of nodes, where each node contains the file name along with the file metadata, such as the list of pointers to data blocks. Consider a given directory foo. Which of the following operations will necessarily require a full scan of foo for successful completion?

Ans Creation of a new file in foo.
Renaming of an existing file in foo.

Layered File System.



Allocation Methods



* DBA = 16 bits
DBS = 1 KB * Max possible file size on the disk = Max possible (given) disk size.

eg:: no. of blocks = 2^{16} = 64K
∴ Disk Size = 64K × 1KB = 64MB

★ Maximum possible disk Size = $(2^{DBA}) \times DBS$

(1) Contigous Allocation of disk space.

Performance: 1. internal fragmentation : Yes (last block)
2. external fragmentation : Yes
3. Increasing file size : Inflexible
4. type of access : Sequential / Random/direct access (faster)

(2) Linked allocation of disk space.

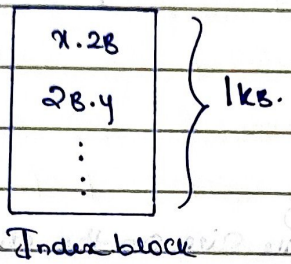
Performance: 1. Internal fragmentation : Yes (last block)
2. External fragmentation : No
3. Increasing file size : Flexible.

4. Type of access : Only sequential (slower)
5. Performance / Efficiency : ptrs consume disk space
6. Vulnerability of pointers : pointers can get broken.

(3) Indexed allocation of disk space

* Each file has index block, index block holds address of data blocks of file.

eg::



DBA = 16 bits = 2B
DBS = 1KB.
No. of addresses = $\frac{1KB}{2B} = \underline{512}$

eg:: DBA = 32 bits
DBS = 4KB.
Max possible file size with index block : ?
No. of addresses = $\frac{4KB}{4B} = \underline{1K}$

File size = 1K x 4KB = 4MB

Q1. A file system supports a DBS = 4KB, each block can hold 512 addresses (ptrs). What is the size of DBA in bits?

$$DBA = \frac{DBS}{N.\ addresses.}$$

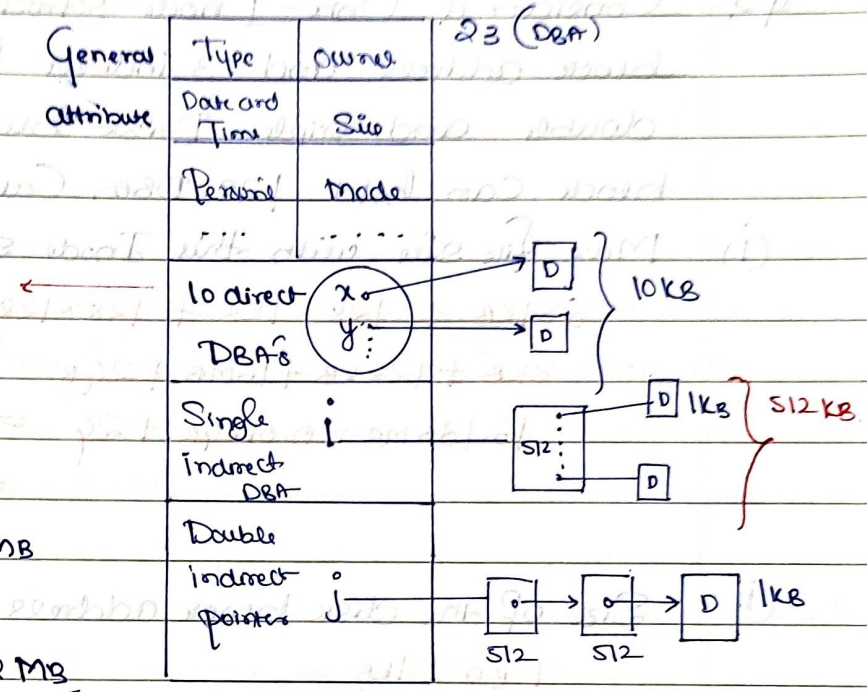
$$DBA = \frac{4KB}{512} = \frac{2^{12}}{2^9} = 2^3 = 8B = \underline{64\ bits}$$

Case Studies of O.S

1. UNIX / LINUX : each file is associated with a I-node

File N	I-node
KKC	28

I-node Structure :



→ With double indirect

$512 \times 512 \times 1KB = 2^{28} = 512MB$

→ Total file size = $256 * 522 MB$

2. DOS / WINDOWS

Block info.

File Name	(Other attributes)	First DBA	Last DBA
KKC	- - - - - -	5	10

Directory

Master File Table

0	
1	
2	<x>
5	<b>
10	- - -
a	
b	<3>
x	
y	<10>

Linked allocation

* No. of entries in FAT (MFT) = no. of blocks
* Fat entry contains addresses (DBA) of next data block

FAT-(32) FAT entry size DBA

Q1. In a file allocation system, which of the following allocation scheme(s) can be used if no external fragmentation is allowed?

Ans (i) Linked allocation
(ii) Indexed allocation

Q2. Consider a Unix-Inode structure that has 8 direct disk block address and 3 indirect block address, namely single, double and triple. Disk block size is 1KB and each block can hold 128 DBA. Calculate:

(i) Max file size with this Inode structure: 16.136 MB.

$= 8 \times 1KB + 128 \times 1KB + 128 \times 128 KB + 128 \times 128 \times 128 KB.$

$= 8KB + 128KB + 16MB + 2GB$

$16.136 MB = 0.016GB + 2G = 2.016 GB$

$= \underline{2GB}.$

(ii) Size of the disk block address:

$DBA = \frac{1KB}{128} = \frac{2^{10}}{2^7} = 2^3 = \underline{64 Bit}$

(iii) Is this file size possible over given disk?

Q3. The index node of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointers. The disk block size is 4KB, and the disk block address is 32 bit long. The max possible file size is 4GB GB.

Ans $N. addressed = \frac{4KB}{4B} = 1K$

File Size $= 12 \times 4KB + 1K \times 4KB + 1K \times 1K \times 4KB$

$= 48KB + 4MB + 4GB$

$= 4.004GB \sim 4GB$

Q4 A file system with 300GB disk uses a file descriptor with 8 DBA, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 B and size of each DBA is 8 Bytes. Max possible file size in the file system is

Ans DBS = 128 B

DBA = 8B No. addressed = $\frac{128}{8} = 16$

File Size = 8×128B + 16×128B + 16×16×128 B

= 1KB + 2KB + 32KB

= 35KB

Q5. The data blocks of a very large file in the Unix file system are alloced using:

Ans An extension of indexed allocation

Q6 Using a larger block size in a fixed block size file system leads to

Ans Better disk throughput and poor disk space utilization

Q7 A FAT based file system is being used and total overhead of each entry in the FAT is 4 bytes in size. Given a 100×10^6 byte disk on which the file system is stored and data block size is 10^3 Bytes, the maximum size of a file that can be stored on this disk in units of 10^6 Bytes is 99.6 $N = \frac{100 \times 10^6}{10^3} = 100 \times 10^3$

Max. file size = Given disk size − FAT size

$= 100 \times 10^6 - 0.4 \times 10^6$

$= 99.6 \times 10^6$

FAT Size $= 100 \times 10^3 \times 4B$

$= 400 \times 10^3 B$

$= 4 \times 10^5$

$= 0.4 \times 10^6 B$.

Q8 A file system with one level directory structure is implemented on a disk with disk block size of 4KB. The disk is used as follows:

Disk block 0 : Boot control block

Disk block 1 : File allocation system/table, consisting of 10-bit entry per data block, representing the data block address of next data block in file.

Disk block 2, 3 : Directory with 32-bit entry per file.

Disk block 4 : Data block 1

Disk block 5 : Data block 2, 3, etc.

(i) What is the max possible number of files?

(ii) What is the max possible file size in bytes?

Ans Number of files (max) = no. of entries

$= \frac{8KB}{4B} = 2K = \underline{2048}$

DBA = 10 bit

DBS = 4 KB

No. of block = 2^{10}

$= 1024 - 4 = \underline{1020}$

Max file size = Disk size − (control overheads)

$= 1024 - 4 = 1020 \times 4\,KB$

$= 4080\,KB$

$= \underline{4.808\,MB}$

Q9. Consider a file system that stores 128 DBA in the index table of the directory. Disk block size is 4 KB. If the file size is less than 128 Blocks, then these addresses act as direct data block addresses.

However, if the file size is more than 128 Blocks, then the 128 addresses in the index table point to next level index blocks, each of which contains 256 data block's address. What is the max file size in the system?

Ans. Min file size = $128 \times 4\,KB$

$= \underline{512\,KB}$

$128 \times 256 \times 4\,KB$

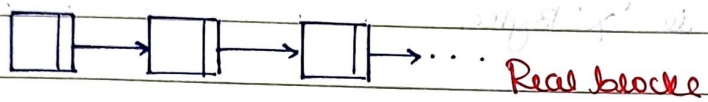
$= 2^7 \times 2^8 \times 2^{12} = 2^{27} = \underline{128\,MB}$

Disk free space managment algorithms :

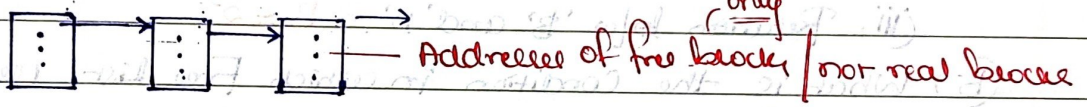
Asume: disk - 20 MB;
DBS : 1KB
DBA : 16 bits
N. blocks = $\frac{20MB}{1KB} = 20K$

Given disk size	≤	Max. possible D.S
		$2^{DBA} \times DBS$

1. **Free linked list** : linked list of free blocks.



2. **Free list** : Keeps track of free blocks only, faster compared to bit map.



Q. How many blocks are (free list) are needed to store 20K free DBS?
Ans: 1 Block — 512 free DBA
? — 20K free " $= \frac{20K}{512} = 40$

3. **Bit-Map / Vector**

: Associate a Binary bit with each block
: Bit map keeps track of all blocks.
: Slower Compared to bit-map.

0 (In-use) 1 (free)

Q. How many blocks of disk are needed to store our bit map?
Ans: DBS = 1KB
DBA = 16 bits

1 Block → 8K bits
? → 20K bits

$= \frac{20K}{8K} = 2.5 \simeq 3$ Blocks

4. Counter method: when we have many contigous free blocks.

	Start free DBA.	No. of CG free blocks
1.	5	45
2.	85	200
3.	325	125
4.	600	25

eg: New file F=20 blocks
Consumption can be done by first fit, best fit, worst fit.

Q1. Consider a disk with 'B' blocks, 'F' are free, DBA is 'D' bits. Disk block size is 'X' Bytes.

(A) Calculate:

(i) Given disk size : "B * X" Bytes.

(ii) Max. possible disk size = $2^D \times X$ Bytes

(iii) Relation b/w 'B' and 'D' : $B \leq 2^D$.

(B) What is the condition in which Free list uses less space than Bit-Map?

: $F * D < B$

Q2 The beginning of a free space bit map looks like after the disk partition is first formatted : 1000 0000 0000 0000. The system always searches for free blocks starting at the lowest numbered block, so after writing file A, which uses 6 blocks, the bit map looks like this : 1111 1110 0000 0000. Show the bit map after each of the following additional actions at the end.

Ans
* File B is written, using 5 Blocks
* File deleted.

Q3. A file system uses an in-memory cache to cache disk blocks. The miss rate of the cache is shown in the figure. The latency to read a block from cache is 1 ms and to read a block from disk is 10 ms. Assume that the cost of checking

whether a block exists in the cache is negligible. Available cache sizes are in multiples of 10 MB.

The smallest cache size required to ensure an average read latency of less than 6ms is 30 MB.

Ans

$6ms = x(1ms) + (1-x)(10ms)$

$= x + 10 - 10x$

$-4 = -9x$

$x = 4/9 = 0.45.$

Miss ratio = 0.55

55% ⇒ 25 ms

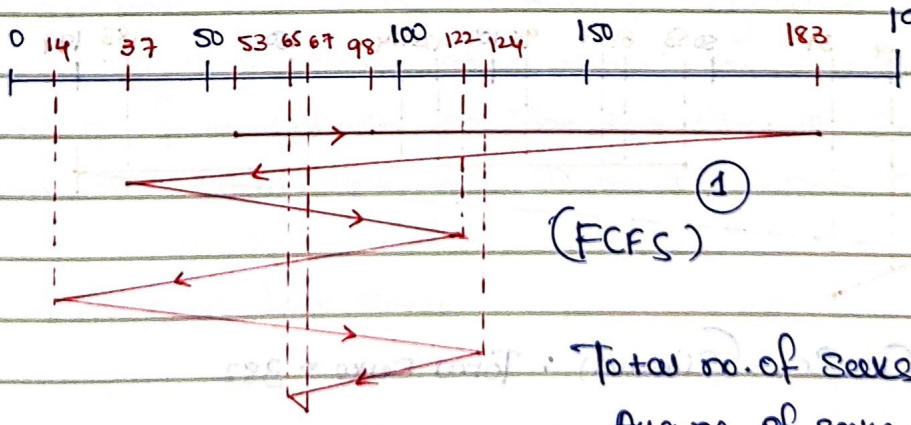
= 30 ms (for this miss ratio reduces, EMAT ↓)

Q4. The amount of disk space that must be available for page storage is related to maximum number of processes 'N', the number of bytes in virtual address space 'B' and the number of bytes in RAM 'R'. Give an expression for worst case disk space required.

Ans. N * S Bytes

Device Scheduling:

Process requests : 98, 183, 37, 122, 14, 124, 65, 67. (Track no./cylinder no.)



(1)

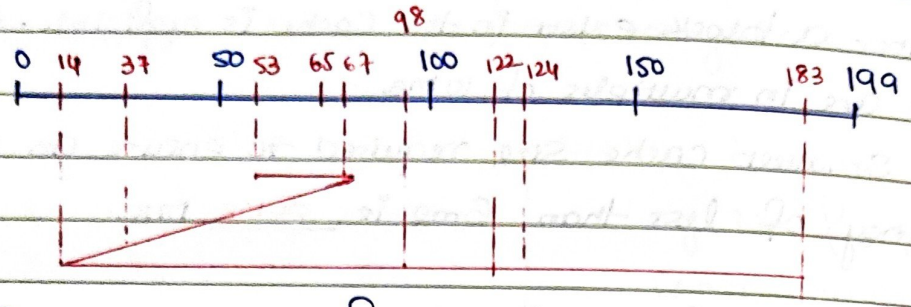
(FCFS)

Total no. of seeks = 640

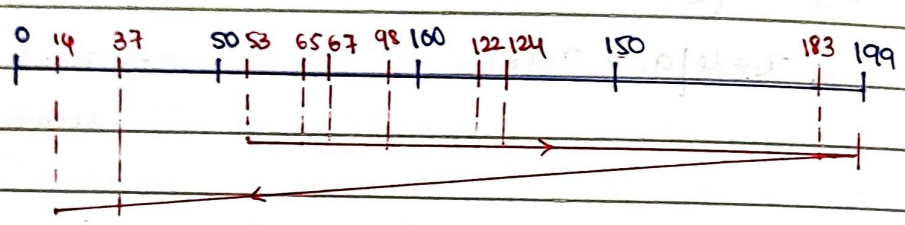
Avg no. of seeks = 640/80

= 80

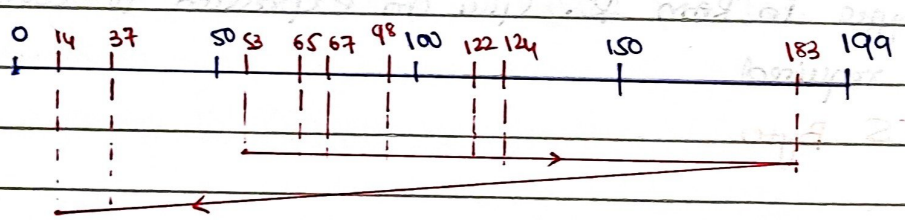
* Objective of disk scheduler is to reduce average seek time.



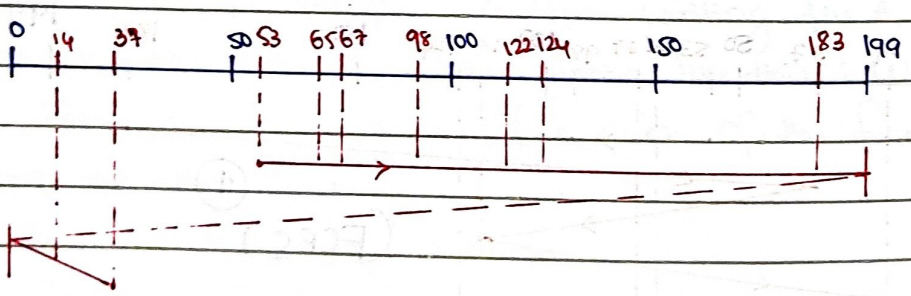
(2.) Shortest Seek time first : Total Seeks = 236



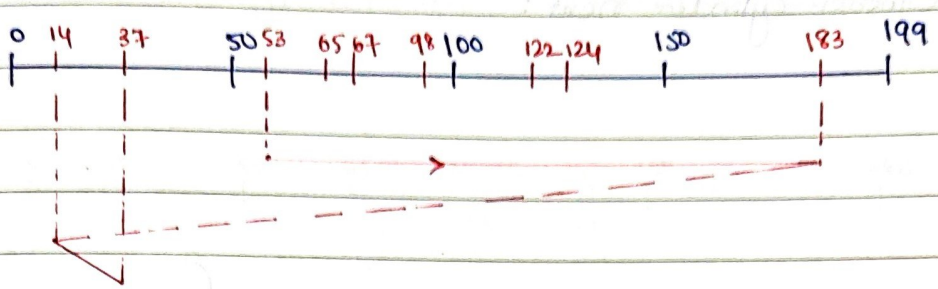
(3) Scan / Elevator : Total Seeks : 331.



(4) Look: Total Seeks = 299

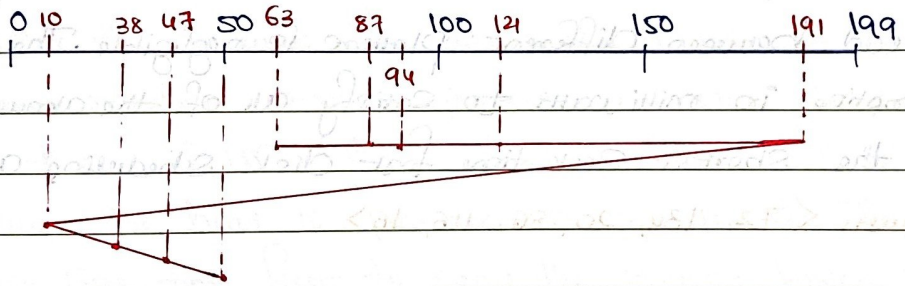


(5) C-Scan (Circular) : Total Seeks = 382



(6) C-Look : Total Seeks : 322

Q1. Consider a disk queue with requests for I/O to blocks on cylinder 47, 38, 121, 191, 87, 11, 92, 10. The C-look scheduling algorithm is used. The head is initially at cylinder no. 63, moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement incurred while servicing these requests is: 346



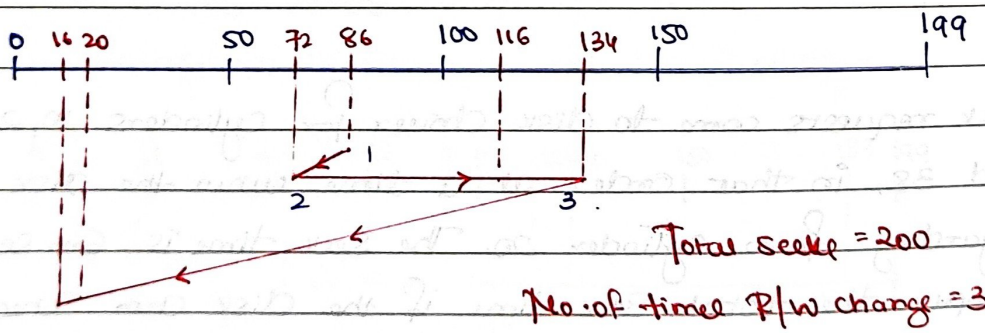
Q2. Disk requests come to disk driver for cylinders 10, 22, 20, 2, 40, 06 and 38, in that order at a time when the disk drive is regarding from cylinder 20. The seek time is 6ms sec per cylinder Compute the total seek time if the disk arm scheduling algorithm is :

(a) First come first serve.

(b) Closest cylinder next:

Q4. Consider a storage disk with 4 platters (0-3) 200 cylinders (0-199) and 256 sectors per track (numbered as 0, 1, 255). The following 6 disk requests of the form [Sect. no., Cyl. no., platter no] are recieved by disk controller at same time [120, 72, 2], [184, 134, 1], [60, 20, 0], [212, 86, 3], [56, 116, 2], [118, 6, 1].
Currently head is at no. 100 of cylinder 80 and moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatt and for reversing the direction of the head movement once is 15 milliwatt. Power dissipation associated with rotational latency and switching of head between different platters is negligible. The total power consumption in milliwatts to satisfy all of the above disk request using the shortest seek time first disk scheduling algorithm is 85

Ans. Request <72, 134, 20, 86, 116, 16>



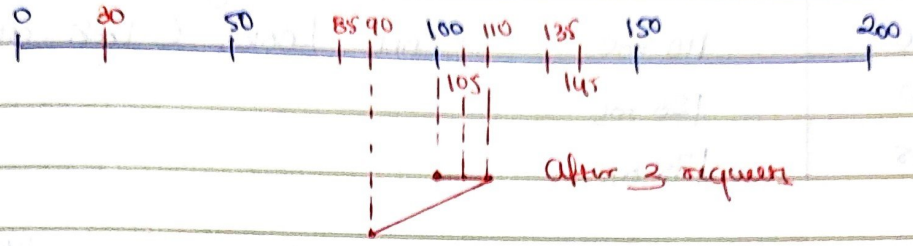
Total seeks = 200 − 20×2 = 40
No. of times R/W change = 3 − 3×15 = 45
its direction.
85

∴ 85 milli watts

Q5. Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time the disk arm is at cylinder 100, there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135, 145. If SSTF is being used for scheduling the disk access

the request for cylinder 90 is serviced after servicing 3 no of request

Ans.



Q6 Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used in FCFS. If FCFS is replaced by SSTF, claimed by vendor to give 50% better benchmark results, what is the expected improvement in the I/O performance of user programs?

Ans 0% (no improvement)

Q7. Consider the following five disk access requests of the form (req. id, cylinder number) that are present in the disk scheduler queue at a given time (P, 155), (Q, 85), (R, 110), (S, 30), (T, 115)

Assume the head is positioned at cylinder 100. The scheduler follows Shortest Seek time first to serve the requests. Which of statements is false?

Q8. Consider a disk with following details:

Track-track time = 1ms.

Current head position = 65

Direction: moving towards higher tracks with higher track = 199

Current clock time = 160ms.

Consider the following data set:

HW

Track no.	T. of arrival
12	65 ms
85	80 ms
40	110 ms
100	120 ms
75	175 ms.

Calculate time of decision, Pending requests, Head position, Selected request, Seek time using FCFS, C-Scan, SSTF, SCAN, Look, C-Look algorithms.