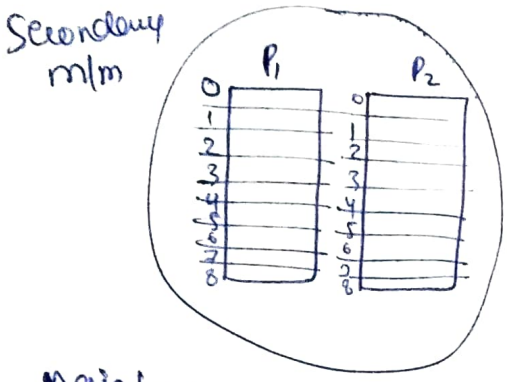
→ Internal fragmentation can be there due to overhead of (P/2) in last page. But it is not considerable.

Consider a snapshot of system at any particular time,



0	OS
1	OS
2	OS
3	$P_1 : p_1$
4	$P_1 : p_2$
5	$P_2 : p_3$
6	$P_2 : p_1$
7	$P_2 : 6$
8	$P_1 : 3$
9	$P_2 : p_7$
10	$P_1 : P_4$
11	Page table of P_1
12	
13	Page table of P_2
14	
15	

OS

Page table of P_1

	frame	P/A	D	R	P
11+0		0	0	0	0
11+1	3	1	1	1	0
11+2	4	1	0	1	1
11+3	8	1	0	0	0
11+4	10	1	1	0	0
11+5		0	0	0	0
11+6		0	0	0	0
11+7		0	0	0	0

→ Relocatable Address (11 + no.)

P/A are valid or invalid bit If its one (1) means page is present in main m/m if it is zero (0) then page is not present in m/m means page fault.

→ In case of page fault, OS reload the page from secondary m/m.

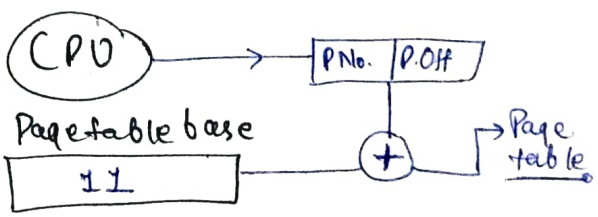
→ D (Dirty bit) or modified bit means page has been modified in main m/m or not If it is modified then the changes will reflected in lower level secondary m/m too called 'write-back' policy at the time of replacement.

→ R (Reference bit) changes after each life cycle. So if page is not refered from long then we can replace it. for this we have various page replacement Algo.

→ P (Protection bit) It contains Read / write / execute.

CS → R/E SS → R/W
DS → R/W
(Code, Data and stack Sections)

If you have no permission than you can't perform action.



x

TLB

→ Paging also have a overhead as page table.

Scenario

To reduce or avoid external fragmentation → Paging

for paging, page table is overhead

↓

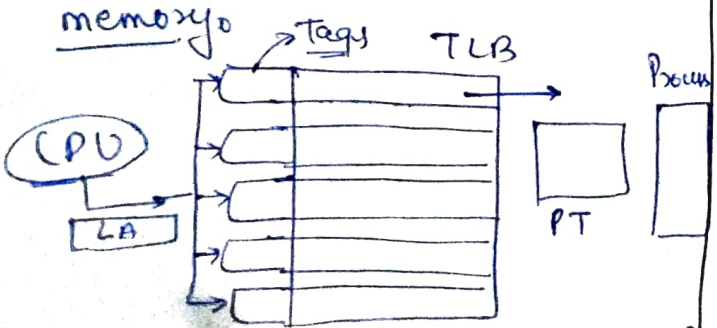
To reduce page table size we increase Page size → It might leads to internal fragmentation

So we come with the concept of multi-level paging and it is also time consuming.

To make it more efficient we store the base address of table in register, which is directly in contact with CPU → But register are very costly and context switching time arises

So, we Introduce Cache, because cache is cheaper than register and size greater than register and also faster than memory. It introduces the concept of 'TLB. [Translation Lookaside buffer]'.

TLB also called as associate memory.



CPU gives the logical address to tags in TLB and if they have same no regarding page no then it will send further. This way we reduce the time taken in accessing page table.

→ TLB also contains page tables entries that is currently used.

→ (TLB size) > register
∧ lesser than
(main memory)

→ Accessing TLB is faster as compare to accessing page table.

→ But if entry is not in TLB then we should go in page table and get it and we will also insert the entry in TLB if space is available But if there is no space in TLB then we have page replacement algo. or some other methods.

If entry found in TLB called as 'TLB hit.'

Say hit rate = 80%.

then, miss rate = 20%

effective m/m access time

$= 0.8(t_a) + (0.2)t_b$

t_a → time taken by mapping if hit

t_b → time taken by mapping if miss.

Ex → time taken in searching page inside TLB = 20ns

time taken in searching inside page table = 100ns

Hit rate = 0.8. find effective access time

Sol^n miss rate = 1 − 0.8 = 0.2

If process hit in TLB −:

$t_a = 20 + 100$

↙ for searching in TLB to get frame no

↘ for accessing frame no. in main memory

If process miss in TLB −:

$t_b = 20 + 100 + 100$

↙ for searching in TLB for a miss

↓ for searching in page table

↘ for access frame no. in main m/m

Access time = $0.8(t_a) + 0.2(t_b)$

$= 0.8(120) + 0.2(220)$

$= 96 + 44$

$= 140$ ns.

→ Searching time in page table and accessing time of frame is same bcoz both task occur in main m/m.

So, access time is 140ns but if we don't have TLB then each time we check page table and then access frame means $(100 + 100) = 200$ns.

So, somehow TLB is better approach.

Note → Above discussion is for single level.

Generalise,

Efficient m/m access time = EMAT

hit ratio = p

miss ratio = (1−p)

TLB time = t

main m/m time = m

(↓ for both page table and frame access)

then, for single level paging.

EMAT = $p(t+m) + (1-m)(t+m+m)$

for n-level paging,

EMAT = $p(t+m) + (1-p)(t+nm+m)$

$= p(t+m) + (1-p)(t+m(n+1))$

gate

gate → we are neglecting the time to update the entry if hit doesn't occur.

gate → Sometime $m >> t$, So, we can neglect t. It's completely depends on questions.

Tags → Tags used to contain the page no.

→ If nothing is given consider single level paging.

Q TLB Questions

TLBA	MA	TLBH	PTLevel	EMAT
20ns	100ns	80%	1	140ns
20ns	100ns	80%	2	160ns
20ns	100ns	80%	3	180ns
20ns	100ns	90%	1	130ns
20ns	100ns	60%	1	160ns
20ns	100ns	50%	1	170ns

Complete the table.

Note → Do all the question during practice by self.

TLB Summary

→ It might be given miss rate instead of hit rate.

TLB access = t

M/m access = m

TLB miss rate = p

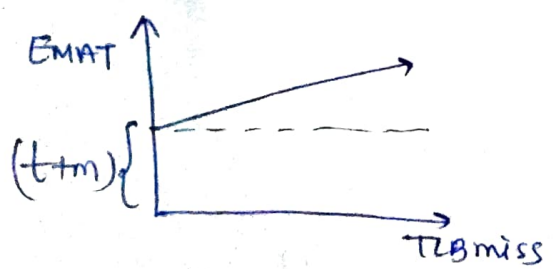
↳ not hit rate

EMAT = p(t+m+m) + (1-p)(t+m)

EMAT ⇒ (t+m) + p(m)

y = c + x a

y = ax + c



If TLB miss ↑ EMAT ↑. So make less miss page replacement Algo. should be good.

So,

EMAT = (t+m) + pm

↳ for one-level page table

and p → miss ratio

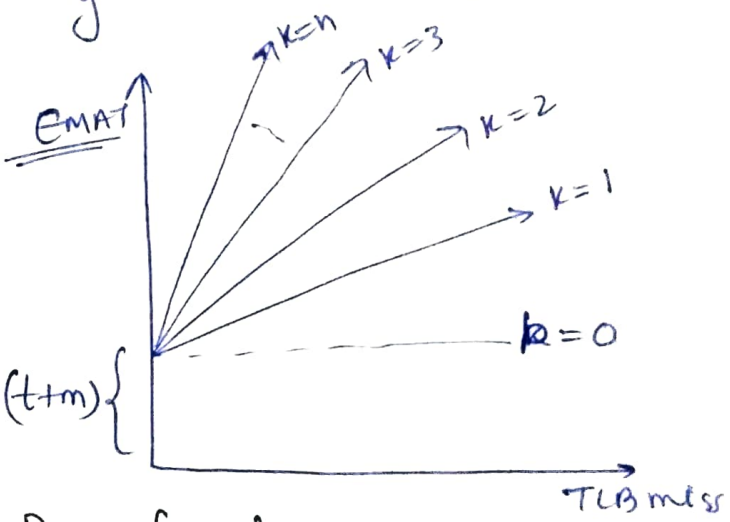
for multi-level paging

No. of page table level = k

EMAT = p(t + km + m) + (1-p)(t+m)

⇒ EMAT = (t+m) + p(km)

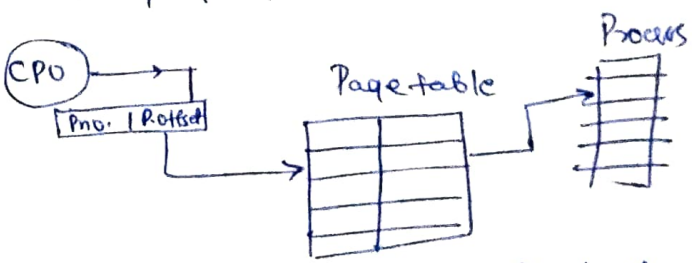
y = c + m a



Page fault

Demand paging: We don't load any page in main m/m until required.

If page not present in m/m then page-fault occur.



TLB miss and page fault both are different things.

If entry not present in TLB called TLB miss and if entry not present in table due to concept of virtual m/m called as page fault.

If page fault occurs then,

① Procss called OS to load the page from secondary m/m called Context switching and its takes (usec).

② After getting CPU, OS loads the data from secondary m/m to page table takes (msec)

③ OS again calls Procss to run via context switching takes (usec).

④ Now Procss will find the add. in page table takes (ns).

and if no. page fault it will only takes step ④ time as (ns).

Thrashing → If large no. of page faults occur called as thrashing.

To avoid thrashing we have to come with algo. that can fulfill the future requirement but its not possible.

So, we can perform good page replacement algo.

Not final formula.

EMAT = p*(Service time or page fault time) + (t)

p → page fault rate

EMAT: Effective m/m access time

EMAT = [p (service time + ma)] + [(1-p) ma] (t)

ma → m/m access time.

So, final formula of EMAT is,

EMAT = p(st + ma) + (1-p)ma

st → service time

→ Graph and complete formula

Gate 2011 page fault

Given Service time → 10ms

ma = 20ns

p = 10^{-6}

EMAT = $10^{-6}(20 + 10000000) + (1-10^{-6}) \times 20$

⇒ $20 \times 10^{-6} + 10^{1} + 20 - 20 \times 10^{-6}$

⇒ 20 + 10

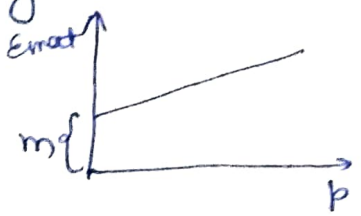
= 30 ns

Emat = p(ps+m) + (1-p)m

⇒ p(ps) + pm + m - pm

⇒ Emat = p(ps) + m → graph

y = x m + c



Above question can be soloed as

EMAT = p(ps) + m

⇒ 10^{-6} (10ms) + 20ns

⇒ 10ns + 20ns

= 30ns

Gate-1998. ps = j, m = i, p = 1/k

EMAT = j/k + i

Effective m/m accesstime and effective instruction time both are same. So,

$$\boxed{Eit = Tlk + i}$$

Gate 2000. [Convertion of value of p is in p.]

St = 10m sec
m = 1 u sec
$p = \left(\frac{1}{100}\right)\% = \frac{1}{10000}$] Remember

$EMAT = 10^{-2}(10ms) + 1us$

$\Rightarrow 10^{-4} \times 10 \times 10^{3} us + 1us$

$\Rightarrow 1us + 1us$

$\Rightarrow 2us$ (1.999 usec)

Or

$EMAT = p(PS+m) + (1-p)m$

$= p(PS + \cancel{m}^{0}) + (1-p)m$

Reason
We can neglect 'm' as compare to 'PS' but in $(1-p)m$ m is independent,

$\Rightarrow p(ps) + (1-p)m$

$= 0.0001(10^{-2}) + (0.9999)10^{-6}$

$= \underline{1.999 \text{ usec}}$

Here exacts ans is 2 us and approximate ans is 1.999 us.

→ But you will get the two closest ans in both approaches and both are correct.

2mb
Gate 2007 [New Concept]

GIVEN:
St = PS = 300 units (Dirty)
m = 1 unit
probability page fault = p (Both cases)
PS = 100 units (Not Dirty)
probabibility of dirty pag = p

So,
$PS = (1-p)100 + p(300)$
$= (100 + 200p)$ unit
m = 1 unit
$Emat = m + (p)PS$
$3 = 1 + p(100+200p)$

$\Rightarrow 100p + 200p^2 = 2$

$\Rightarrow 100p^2 + 50p - 1 = 0$

$\Rightarrow \boxed{p = 0.0194 \text{ units}}$

but option given as 0.194, so they might miss a 0.

→ In case if dirty bit is set or page is dirty then, some time will take in updating the info. to secondary m/m.

So, if process have some pages in page table, they might be of both types either,
→ Dirty
→ Not Dirty.

Good. [first do it by reading the soln]

Gate 2003 [TLB and paging]

VA = PA = 32 bits

VA

10	10	12

PTE(e) = 4B

→ find the given info. in questions of 2003 and solve it.

Soln

TLB and cache both uses as saving page table entry and process data respectively.

TLB hit = 0.96
Cache hit = 90% = 0.9.

No, page fault means we have no need of sevice time.

Cont—

Gate 2004, 2014 on TLB

↳ Do it by self and watch the video in downloaded video if require

Inverted page table

This concept is not widely popular because it saves space but waste the lot of time

→ We make a page table for each process and if m/m have large no. of process then we have large no. of page tables in main m/m

Say we have two process P_1 & P_2 as –

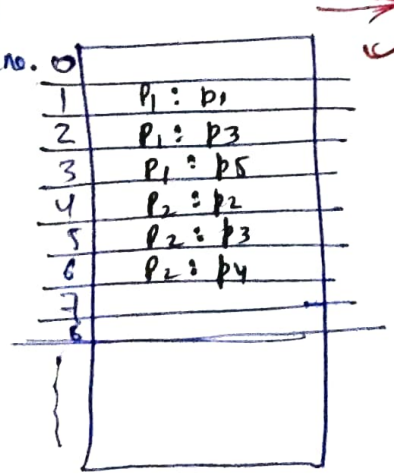
Page table of P_2

0	X
1	X
2	f4
3	f5
4	f6
5	X
6	X
7	X

Page table of P_1

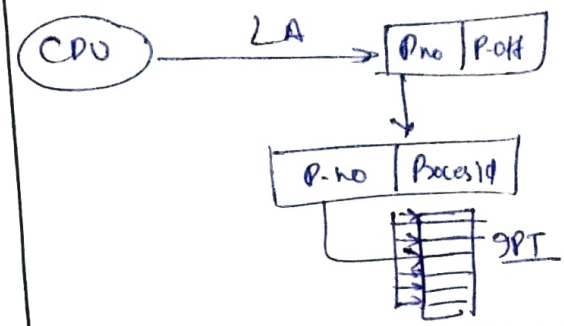
0	X
1	f1
2	X
3	f2
4	X
5	f3
6	X

So instead of making lot of page tables that contains lot of vacant entries we make a single inverted page table as,



→ Index are frame no. here instead of page no. in page table

We maintain both page no. and process id in inverted page table



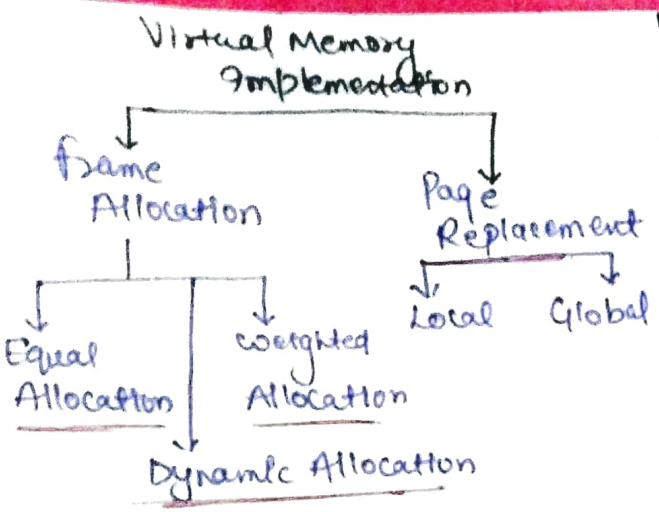
In inverted page table all the entried checked simultaneously at a time

Ex PA = 32 bits
PS = 4KB
(IPT) entry size = 4B
what is the size of IPT?

PS = 2^{12} B
No. of frames = 32-12
= 20
(size of IPT) = $2^{20} \times 4$
= 2^{22} B
= 4MB

Importance of frame allocation and page replacement

If there is no virtual m/m then each time only a single process will get the main m/m, but in virtual m/m multiple process can run simultaneously so it is difficult to maintain that each process get a fair chance to run bcoz all the process want to access a shared common main m/m.



→ Each process is allow to access only certain number of frames in main m/m. Once a process get its max. limit and if it needs to load more page then it will take help of page replacement.

→ If we got page fault for every instruction then program time will increase exponentially called thrashing.

→ So, we have a min. value and max. value for frame allocation

min. value → min. no of frame required to run a instruction.

max. value → size of process (no. of pg.)

So, we choose a number between them.

① Equal allocation : no. of frames = 30
no. of process = 3
frames per process = 10 frams.

② Weighted allocation
no. of frames = 30
3 process with (10, 100, 1000 pages)
then we can't do equal allocation.

Ex: P_1 = 20 pages P_2 = 30 pages
P_3 = 50 pages
no. of available frames = 10 frams.
no. of required pages = (20+30+50) = 100
So, P_1 get $\left(\frac{20}{100}\right) \times 10 = 2$ frames
Similarly P_2, P_3 get 3, 5 frames.
→ This is weighted allocation.

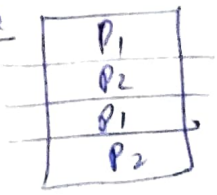
Generalise

If P_i → Process and S_i → size of process
then, $S = \sum_{i=1}^{n} S_i$
assume no. of frame available = f
$\Rightarrow$ $(P_i \text{ gets}) = \left(\frac{S_i}{S}\right) \times f$ frames.

→ Till now both the methods are Static, means we can't change once we allot.

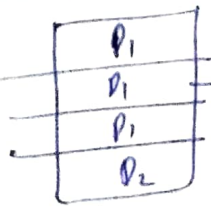
③ Dynamic allocation → We can change the allocation as they want according to process priority.

Assume

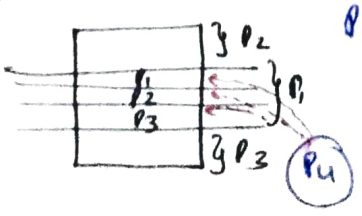


Say Priority ($P_1 > P_2$)

and P_1 needs one more page then



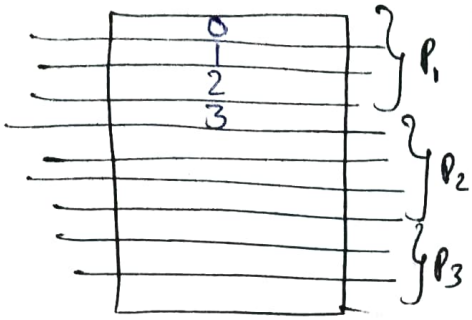
Page replacement



P_1 gets its 3 pages but it wants to load then it needs to replace.

which page will get replaced?

① **Local (PR):**



Replacement will occur in frames alloted to P1 process only.

② **Global (PR):** But in global page of another process can also get replaced if they are free, using some parameters as priority.

→ Local is more preferred over global.

→ If nothing explained about type of algo. then go for local algo. in GATE exam.

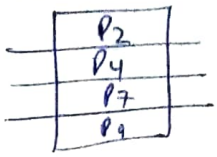
Demand paging → we will not load the page until its required

→ **Pre-paging** is just opposite to demand paging.

Reference String → Strings or order of pages requires by the CPU to execute a process.

Ex → 0145716..
Shows page no.

Pre-paging → Assume a process with 15 pages got 4 frames in main m/m and it allot or filled the frams as

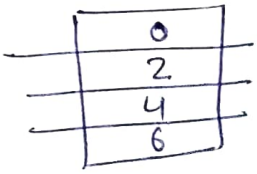


means pre-paging.

bcoz, we don't know which page is required in future we filled the frames with any pages and in future if new page is require then we replace it.

Demand paging → Assume a process with 10 pages got 4 frames in main m/m and initially the frams are empty. then it has a reference string as-

0246753 2 ——



Initially it is empty so page fault occur and '0' get loaded similarly four page fault occurs in order to fill first four frames, but all the pages are necessary.

Page-replacement Algo.

① **Optimal:** Page that not being refered for longest time replace it.

② **LRU (Least recently used):** Replace the page which has not been referred for a long time.

③ (FIFO) : first In first Out

Optimal algo is for reference bupose. we compare each algo. with optimal algo. for benchmarking.

→ In optimal we search page that not reffred 'for long in future' but in LRU we replace the page that not been referred 'from long in basto'

Question

Demand paging used find page fault?

① 2014 (Framesize = 5)

4, 7, 6, 1, 7, 6, 1, 2, 7, 2

→ Optimal:

4 7 6 1 7 6 1 2 7 2

↑	↑	↑	↑				↑		

4	1
7	2
6	7

No. of page fault = 5

Bcoz initally all frames were empty so, 4, 7, 6 algo get pagefault.

→ LRU

4 7 6 1 7 6 1 2 7 2

↑	↑	↑	↑				↑	↑	

4	1
7	2
6	7

No. of page fault = 6

→ FIFO

4 7 6 1 7 6 1

↑	↑	↑	↑			
	2	7		2		
	↑	↑				

4	1
7	2
6	7

No. of pg. fault = 6

→ Sometimes we also maintain a queue in case of FIFO as,

4	7	6	1	2	7	

Replaces

→ No any algo. can beat optimal.

② 1995 grp

0100, 0200, 0430, 0499, 0510, 0530, 0560, 0120, 0220, 0240, 0260, 0320, 0370.

100 records per page present with 1 free main m/m frames

Convert address no. to page no. first

[0001 – 0100) → Page 00
[0100 – 0200) → Page 01
[0200 – 0300) → Page 02
[0300 – 0400) → Page 03
[0400 – 0500) → Page 04
[0500 – 0600) → Page 05

Reference string on given address.

01, 02, 04, 04, 05, 05, 05, 01, 02, 02, 02, 03, 03

1 frame

01	02	04	04	05
↑	↑	↑		↑

05	05	01	02	02	02
		↑	↑		

03	03
↑	

Page fault rate = 7/13
or miss rate

No. of page fault = 7

→ There is no choice so every algo. will give the same result.

Gate 2009 (frame no = 3)

1, 2, 3, 2, 4, 1, 3, 2, 1, 1

Gate 2007 (frame no = 3)

1, 2, 1, 3, 7, 4, 5, 6, 3, 1

Gate 2014 (frame no = 3)

1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6

Do all the question by applying FIFO, LRU and Optimal Algo.? find no. of page faults.

Nok → Do it by self on rough copy and verify the answers

Belady's Anomaly

Reference string

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

① No. of frame = 3

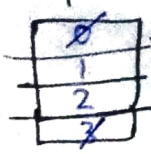
→ Optimal

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

No. of page faults = 7

② No. of frame = 4

→ optimal



0 1 2 3 0 1 4 0

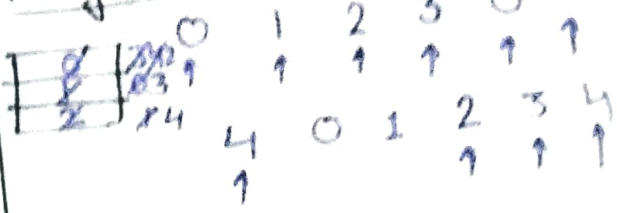
1 2 3 4

No. of page fault = 6

→ More the frames contributes lesser number of page faults, except Belady's anomaly.

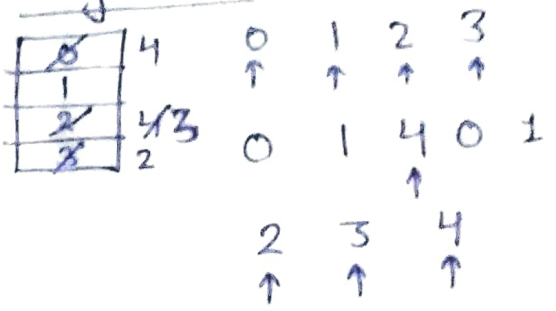
② LRU

No. of frame = 3



No. of page faults = 10

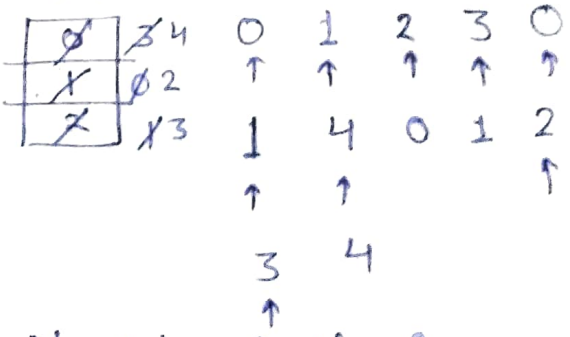
No. of frame = 4



No. of page faults = 8

③ FIFO

3 frames



No. of page fault = 9

4 frames

No. of page fault = 10

grap

On this example on increase no. of frames page fault also increase called as 'Belady's anomaly'.

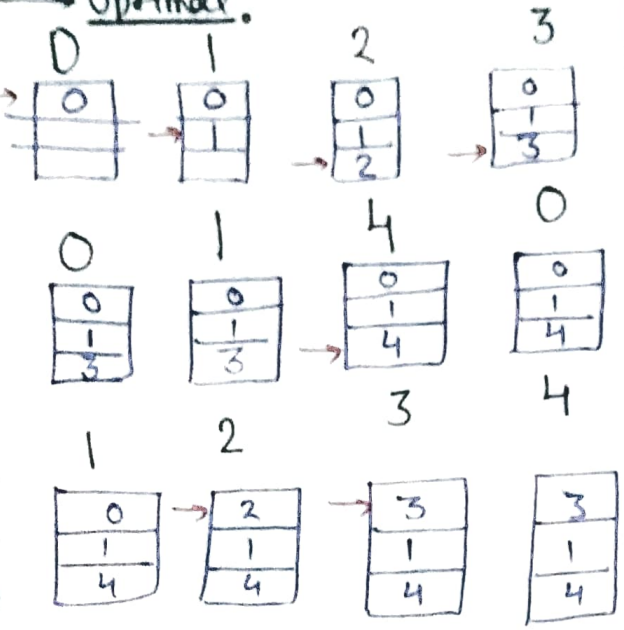
→ So, for this scenario we have stack algorithms that doesn't follow belady's anomaly.

→ Belady's anomaly applies on FIFO page replacement algorithm.

Stack Algorithms

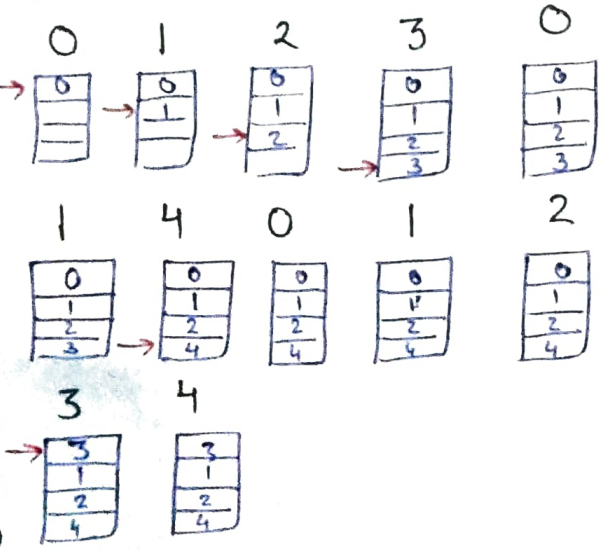
→ Reason for belady's anomaly is stack property

→ **Optimal**.



→ Here we use optimal algo and represent the state after each page allocation.

for 4 frames

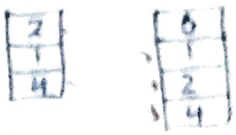


→ (In both the case (3 frames / 4 frames) if we have multiple choice to replace page we go for first one here).

Observation

Here, we can observe that the pages present in 3 frames are also the part of 4 frames corresponds to particular page

Ex ① 2 that comes 2nd time



② 4 that comes first

Similarly, we can observe for each page

→ Order might be different but all are present.

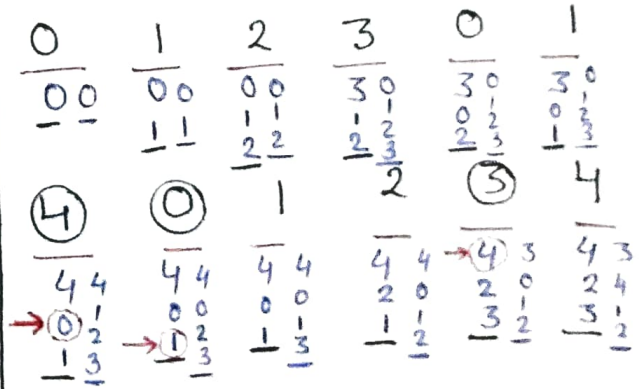
→ This property is called stack property.

→ Pages (3) ⊆ Pages(4)
Pages (m) ⊆ Pages(m+1)

Algorithm that doesn't fail in belady's anomaly' are called as stack algorithm.

→ **FIFO**

On doing same kind of state observation in 3 frames and 4 frames.



with 3 and 4 frames combines states.

→ FIFO doesn't follow stack property, so it has belady's anomaly.

and Similarly LRU also follows stack property, so it is stack algorithm.

Q A memory page containing a heavily used (most frequent) variable that was initialized very early (longest) back and is in constant use (now is refered) is removed when ____ is used.

① LRU ② FIFO ③ LFU ④ none

Not LFU (least frequently used) bcoz it is heavily used.

Not LRU bcoz it is in constant use.

But its FIFO bcoz it was declared very earlier so it will get replaced, and it can also replace the page that is in constantly use.

Gate 2007 on FIFO (P & Q statement)

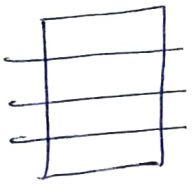
→ Some programs do not exhibit the locality of reference.

→ Both the statements are right but Q is not the reason for P.

→ Reason for Belady's anomaly is Stack property absence.

Gate-2010 Q mp

If we do following demand paging then on 'n' pages we will have n page faults.



If we access 1-100 pages first at last 97, 98, 99, 100 will be in the frames.

and when we access in reverse order 100, 99, 98, 97 pages are inside so they have no page fault and below 96 all have page fault.

So, total page fault = 100 + 96 = 196

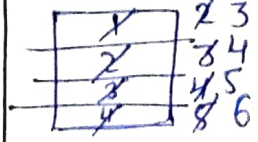
→ Very Good and tricky Question above one.

Some interesting behaviour of optimal page replacement algo.

→ If looping follows in reference string as,

1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 ... ①

Say we have optimal algo. and 4 frames.

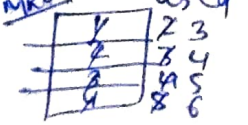


→ Optimal behaves as MRU

Here we observed that we replace the page that is most recently used.

→ In case of most recently used

MRU



same pattern as above

If we use LRU on same string then

1	8	3	1	5
2	6	4	2	6
3	1	8	3	1
4	2	6	4	2

So LRU works as worst performance

Similarly FIFO also did its worst performance. [All page faults]

Note → If replacement called means (no. of frames < no. of pages) in case of looping then optimal and MRU is going to give best performance

Sequence ②

1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 1

R → ← Reverse(R)

In this case optimal work as LRU and FIFO and all the 3 will give same no. of page fault, but MRU give its worst performance and no. of page fault equals to no. of pages in string [all page faults].

Gate 2014 Question on LRU, MRU, LIFO, FIFO and optimal →

→ Simple and trick question So, Do it by self and write the trick if got (any) on the sticky chit notes.

Working Set Algorithms

All the algo. we discussed till now have limitation that we had fix the number of frames called Static algorithm.

But in working set algo. we assume that the nearest future is close approximation of recent past.

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

Assum window size A = 4 then working sets are –

$w = \{1\}$	$w = \{1,2,3\}$	$w = \{1,2,4\}$
$w = \{1,2\}$	$w = \{1,2,3,4\}$	$w = \{1,2,4,5\}$
$w = \{1,2,3\}$	$w = \{1,2,4\}$	$w = \{1,2,5\}$
$w = \{1,2,3\}$	$w = \{1,2,4\}$	$w = \{1,2,5\}$
	$w = \{1,2,4\}$	$w = \{1,2,4,5\}$
		$w = \{1,2,4\}$

So, number of frames or pages required of a process is increasing or decreasing. So if process require all four we allot to it and if it requires less than 4 we allot the other pages to other processes.

Average frame requirement

$$= \frac{1 + 2 + 3 \times 10 + 4 \times 3}{15}$$

(15 → no. of working sets)

$$= \frac{3 + 30 + 12}{15} = \frac{45}{15} = \lceil 3.0 \rceil = 3$$

Page-fault limiting or decreasing Algo.

→ It is the modification of working set algo.

Gate 1992 Question → Do it by self.

Page replacement Algo. Implementation

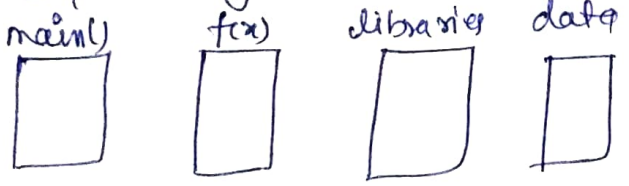
NOT IMPORTANT FOR GATE

Segmentation

Suppose that we have process that have alloted 4 frames in main m/m but process have 8 pages then system can manage the things using virtual m/m but in user point of it is not considerable. User might think that how it works.

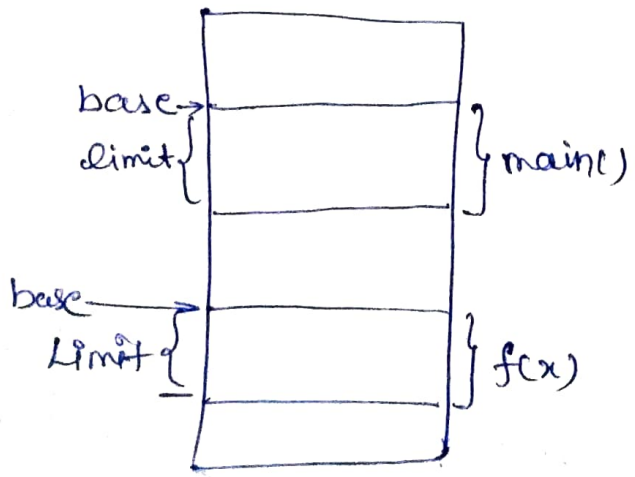
→ So, paging is a Concept in System point of view and Segmentation is concept for user point of view.

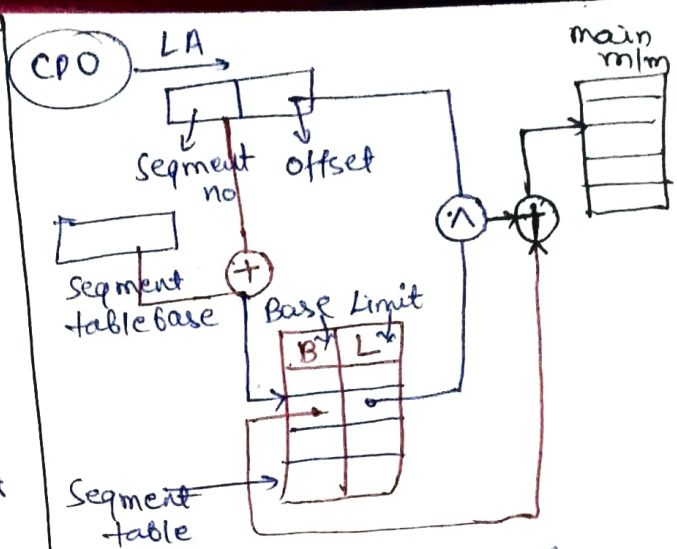
In segmentation we devide the segments of a process as -



This way we form the segment for each program.

In main m/m we alloted space for each segment as



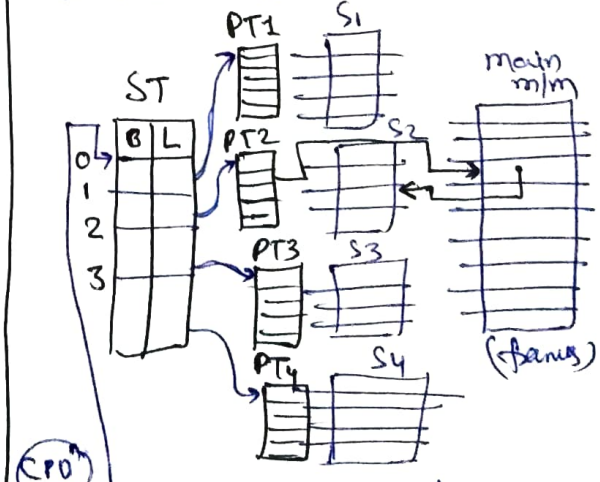


If value of 'offset < limit' then add the value of offset to Base value and search it in main m/m.

Generally, segmentation doesn't used alone, we use segmentation and paging in a combine way.

Segmented paging:

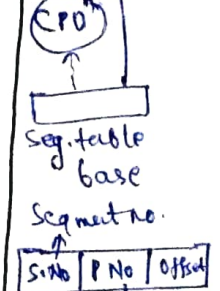
Segments need not be of same size but all the segments divided to same size pages



Segments (pages).

PT → Page table for each segment.

ST → we have just one segment table.



CPU → (LA) → Segment table

Main m/m ← Page table ← Segments

→ Each Page table take 1 frame and segment table also take 1 frame.

→ Base register pointed to frame in which segment table is present (or segment table base).

Todo.

Gate 1999 → [Do it by seeing question]

PAS = VAS = 2^{16} B

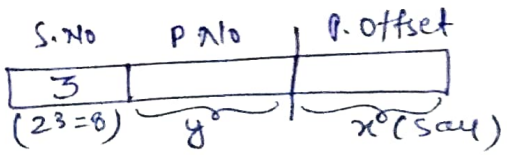
PA = VA = 16 bits

Do by self first to

→ 8 non-overlapping equal size segments.

→ Segment table have PA of Page table.

→ Page table entry = 2B.



Page size = 2^x

No. of pages = 2^y

$x+y+3 = 16$

$x+y = 13$ — ①

Ans

Page size = 2^7

To fit page table in 1 page.

Page table size = No. of entry * size

$\Rightarrow 2^x = 2^y * 2$

$\Rightarrow 2^x = 2^{y+1}$

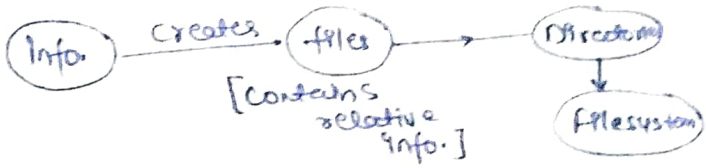
$\Rightarrow x-y = 1$ — ②

Using eqn ① and eqn ②

$x = 7$ $y = 6$

File system, I/O and protection

Attributes and operations on file

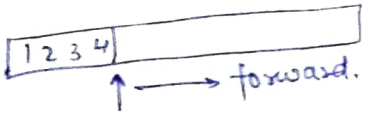


File is a abstract data structure that have following attributes –

① Name
② Identifier (.exe, .doc, .xls, .mp4)
③ Type (Text, video, Audio, etc)
④ Location [In case of more than two Hard disk]
⑤ Size (in bytes)
⑥ Protection [Password protected]
⑦ Time and Data

File operations

① Creating [Some space allot in HD]
② Writing [Add some data in file]
Operating system maintain a pointer to maintain the last ended position of writing. So that we can start from forward to pointer.



③ Reading [Read the file sequentially]
[It also have a read pointer]
Both the pointers maintain by the OS.
④ Repositioning [Change the position of pointer]
⑤ Deleting [Delete the data and file attributes both]
⑥ Truncating [Delete the data of file only]
↳ Redo file

Cont.. →

Open file Table

Information required to access a file –

1. File pointer
2. File open counter
3. Disk location of a file
4. Access right

Those files which are using by process very frequently. It is a good thing to maintain the info. about those files in main m/m.

Assume we have 2 process, P_1 P_2

FileName	FD	AR

FileName	FD	AR

Each process have its own file table contains info. about file name, file pointer (FD) and access right (AR as read/write/exe).

Both the process might access the same file at a time and it is not to worry.

→ OS maintain this file table for each process along with this OS also maintain a open file table that contain location of file and open counter.

file name	Location	Open county

As the process open the file, open Counter get increment and as file entry gets deleted in process file table (means process is not using file) file counter decremented.

→ If the value of open counter is '0' then entry gets deleted from open file table.

Accessing Files

Access methods –:

1. Sequential
2. Direct
3. Indexed

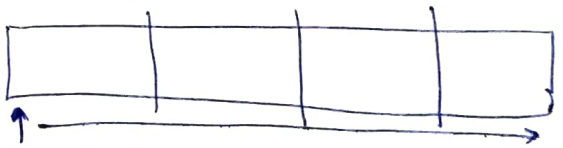
We devide the file into blocks called logical blocks.

In general file size is 512 B.

We divide HD into blocks called physical blocks (sector). Its size is also 512 bytes.

→ file might not always stored in the sequence.

file logical blocks



Access pointer

If you reading ~~reading~~ or accessing bit by bit and incrementing the pointer sequentially called sequential access.

for direct access we have to maintain some entries like

→ No. of records per block
→ No. of records per file
→ Block size.

Indexed access uses by B and B+ trees generally.

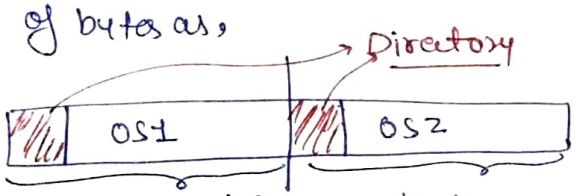
→ We access the record in file using index of record.

→ Most of OS provided sequential access.

Directory Structure

We can have more than one file system in same computer bcoz we are using more than one OS in a single system.

Assume HD is continuous array of bytes as,



Minidisk / Partition / Volumes

→ Divide the HD into various part and allocate to various file system.

→ Each OS have different file system.

Ex. Windows have abc.dox
Linux have abc.odt

Each partition have special file called directory, that converts the file name to required info. for access.

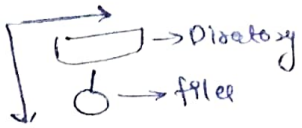
→ Directory contains metadata about files.

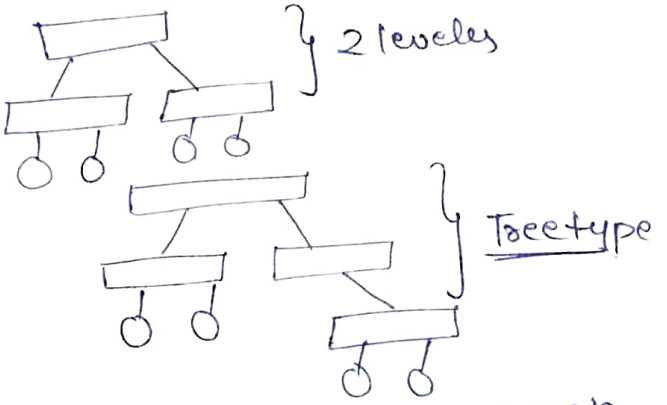
Operations on directory:
i) Search
ii) Create a file
iii) Delete a file
iv) All the files in entry
↳ List all the files
v) Traverse
vi) Rename

We generally have single level directory.

But if there is large no. of files then we sometimes need multi level directory as—

i) 2 level directory
ii) Tree type

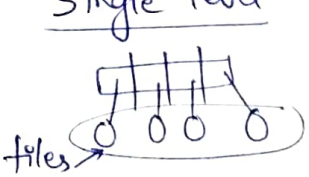




We also have acyclic graph and graph directory. [Tree without cycle]

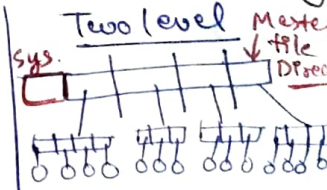
Single level v/s two level directory

Single level



→ Simple implementation
→ Creating and searching is simple
→ Naming of file is difficult (Same names not allowed)
→ Access security is difficult

Two level



→ Master contain user info.
→ Each user have different directory so searching is easy.

→ Two users can have file with same name.
→ OS takes care of access rights to particular directory.
→ One user can't access the system file of another user. ☹ [Soln]

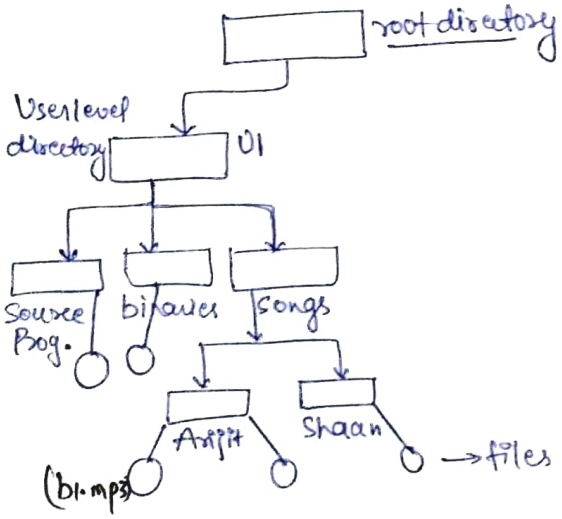
Soln → Create a separate directory in master that contain only system files. [Sys]

→ Two level is difficult to implement

Tree structured Directory

Extension to 2-level directory.

→ In this directory user can also group same kind of files.



→ files can be obtain at any level.

→ Here, path use to access the file

root/U1/songs/Arijit/bi.mp3

Absolute path name → from [root to bi.mp3]

Relative Path name: If root/U1 taken from present working directory then we can do the same using the given path as,

songs/arijit/bi.mp3

We have variable that maintain the state of present working directory. It can be named varying from OS to OS.

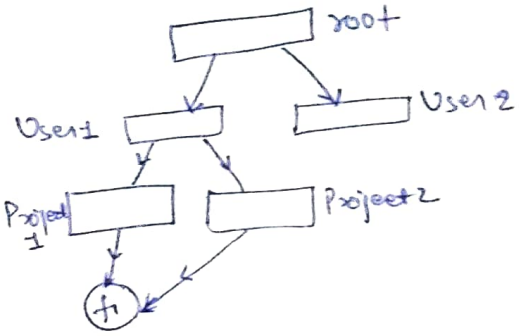
→ We can also change the present working directory if OS provide as permission using some System Calls.

Permission required → Read/write/Execute Permission.

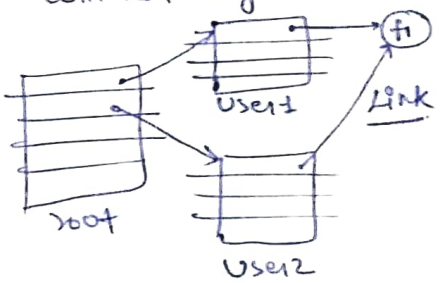
→ Traversing is not easy.

Acyclic Graph

How files can be shared between two directories.



→ Simplest way to share the file is make copies in both but if one user is update it will not reflect in another copy



This ways using links we can share files. This ways directories form a acyclic form (cycle not formed).

→ Problem

Assume User1 delete the file and delete it but user2 is also pointing the same file that doesn't exist there. It arise the situation same as dangling pointer.

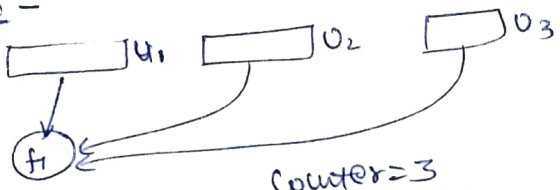
→ Same file can have different path name.

→ We can also share directory so, links can be created for directory too.

To delete a file there is problem bcoz many links are connected to single file.

So, make variable as counter that have value equals to no. of references to file and if all the references get deleted then delete the file.

Ex -



File Systems

File systems provides the mechanism for on-line storage and access to file contents including data and programs.

File-system is a software that deals with files.

File system deals with following issues :-

1. File structure
2. To allocate disk space
3. Recovering freed space
4. To track location of data
5. Interface other parts of OS to secondary storage.

File system Structure

File sys. provides efficient and convinient access to the disk by allowing data to be stored, located and retrieved easily.

We generally follows layer approach.

Benefit of layer approach is - If we change in any layer it will not reflect in another layer.

Application Program
⇓ ask for file to access
Logical File System (Meta data, manage Directory str.)
⇓
File Organization Module (logical block → physical blocks, free space management)
⇓
Basic File System (Commands to I/O control buffering)
⇓
I/O Control (Consist device drivers, interrupt handlers)
⇓
Devices

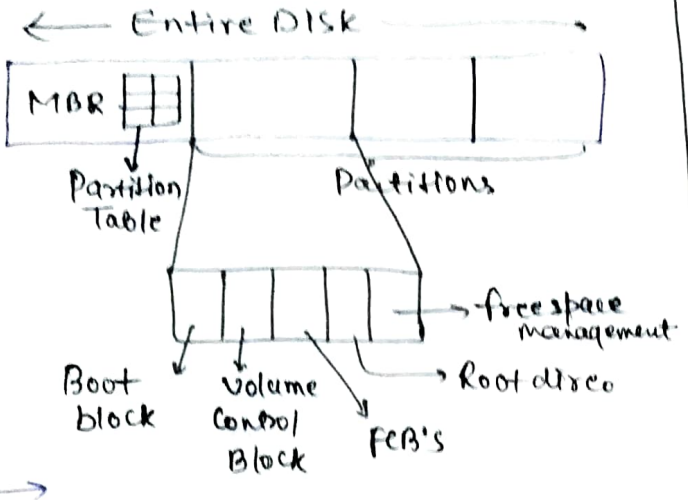
→ Logical file system manages directory str. and identifies which file is asked.
→ It also divide the file into logical blocks.

→ file organization module map logical block with physical block and allot the free space to required files.

→ Store the block in main m/m in buffer that (maintain temporarily).

Basic file system issues command to I/O contol and device drivers get the data using those commands.

MBR (Master Boot Record)



→ When you turn ON the system the RAM have nothing to execute. so a special program BIOS (Basic Input output system) get loaded from ROM to RAM to start a procedure called as booting.

→ As BIOS get activated it loads the MBR in RAM and due to this you get options in front screen as-

→ Windows
→ Ubuntu
→ Mc OS

If you have multiple OS then you get the options and select from your choice. MBR also have a partition table that contains starting of each partition.

→ Each system have an active file system (OS). If you have only one file system by defeult that is active otherwise you have to select it.

'On disk' data structure uses in file system implementation

Each file system uses different data structures.

Several 'in memory' and 'on disk' structures are used to implement a file system.

These structures vary depending on the OS and file system but general principles apply.

Boot Control block (per partition)

It contains info need by system to boot an OS from that partition.

→ In Unix file system called as boot block.

→ In NTFS called as partition boot sector

NTFS → New Technology file Sys.

→ Every OS have suppose to maintain Boot control block.

Volume Control block: It

Contains volume details such as no. of blocks and its size in partition.

In UFS called as Super block.
In NTFS called as master-file table.

→ You can also find about the empty blocks.

Directory str per file system

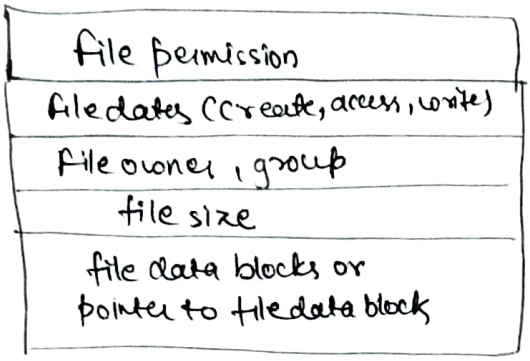
It is used to organize the files

In UFS this includes file

names and associated 'inode' Numbers.
[Index Block]

File Control Block (FCB):

It contains details about files.



file control block ↑

'In memory' data structures in file system implementation

The in-memory is used for both the file system management and performance improvement via caching. This data loaded at 'mount time' and discarded at 'dismount time'

Mounting → Create new partition.
Ex → As you connect you pen-drive system creates a new partition for you.

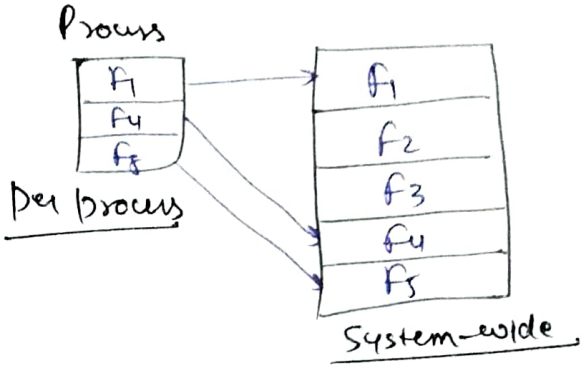
(i) **In-memory mount table:** It contains information about each mounted volume.

(ii) **In-memory directory str. Cache:** It holds the directory info of recently accessed directory.

(iii) **System-wide Open file table:** It contains the info. of each open-file by OS and contains FCB.

(iv) **per process open file table.**

It contains pointer to appropriate entry in the system wide open file table.

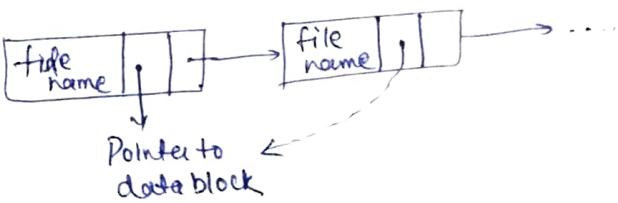


Directory Implementation

The selection of directory allocation and directory management algo. significantly affects the performance and reliability of file system.

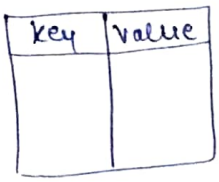
Algorithms:

1. **Linear List**



Here, search operation is going to take lot of time.

2. **Hash table.** It have Linear list and hash table both.
→ It makes the searching easy in O(1) time.



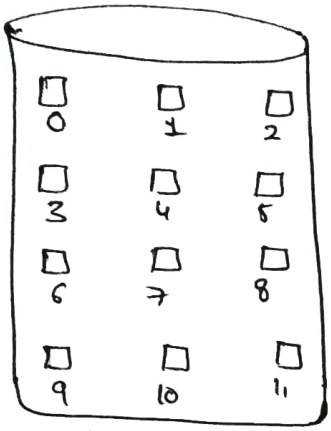
→ It have fixed size only and also have collision.

→ We can also use B and B+ trees for implementation.

Allocation methods

→ How to allocate space to the files so that disk space is utilized effectively and files can be accessed quickly?

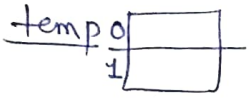
Contiguous Allocation

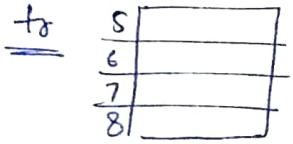


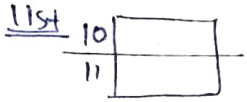
Hard Disk

Directory

file	start	length
temp	0	2
tr	5	4
list	10	2

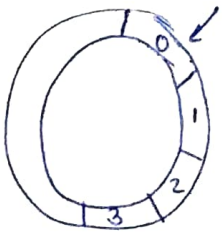






So, we are using contiguous allocation

→ Read performance is excellence due to sequential allocations



Hard Disk track

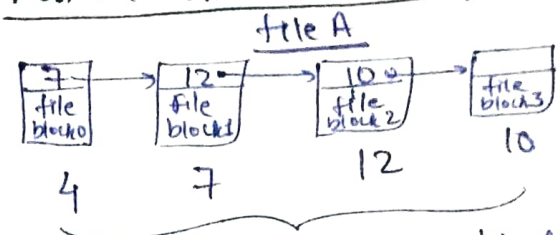
→ Disadvo: Disk become fragmented

→ Internal fragmentation can be possible as if process required 2.5 block.

But we can't avoid it.

We can avoid external fragmentation using non-contiguous methods.

Linked list allocation



(4, 7, 12,) are physical blocks in which file blocks are saved.

Advo → ① Every disk block can be used.

② No external fragmentation, so no space lost (except internal fragmentation).

Disadvo → Random access is very slow.

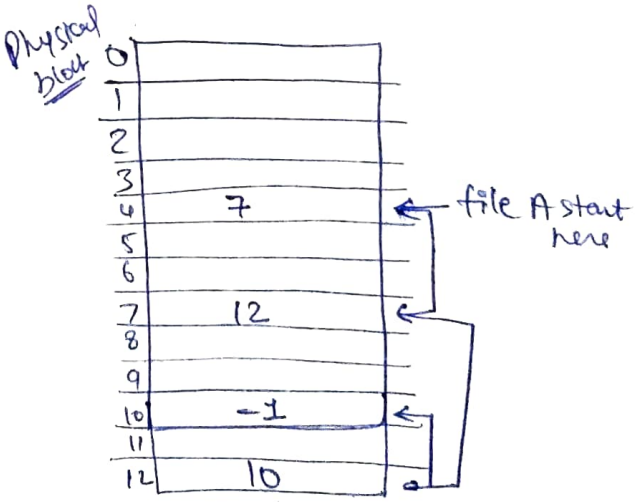
→ Pointers takes two bytes.

[9mb. concept to learn.]

File Allocation Table

Solution

If you want to access randomly in linked list allocation then use file allocation table.



Each block contain index of next file block.

→ But its size might be large or very large.

→ But it provide random access.

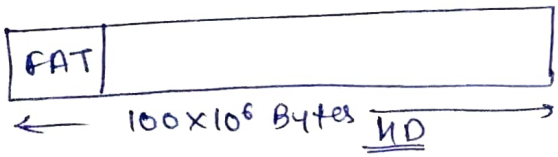
Hard Disk = 20GB (say)
Size of block = 1KB
No. of block = $\frac{20GB}{1KB}$ = 20M
If blocksize = 4B
then
size of table = 20M × 4B
= 80MB

→ This table is also a (in-m/m) data structure, so we can do random access quickly.

[Must do by self first]

Gatelly on File Allocation Table

Disk size = 100×10^6 bytes
Data block size = 10^3 B
FAT entry size = 4B



FAT = no. of entries * size of entry
No. of blocks in HD = $\frac{100 \times 10^6}{10^3}$
= 105
No. of blocks = No of entries in FAT
FAT = $4 \times 105 = 0.4 \times 10^6$ B

Space available for file
= $100 \times 10^6 - 0.4 \times 10^6$
= 99.6×10^6 bytes

Ans- 99.6

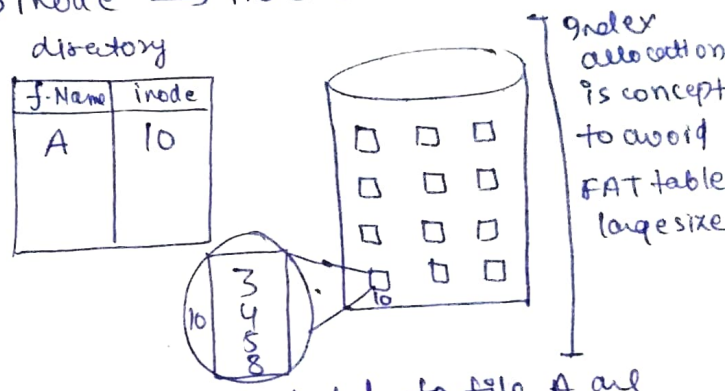
Indexed allocation

→ Linked allocation solves the external fragmentation and size declaration problem of contiguous allocation.

→ In the absence of FAT, linked allocation doesn't support direct access efficiently.

→ Indexed allocation solve the problem by bringing all the pointers together into one location : the inode

⇒ inode → Index block are same

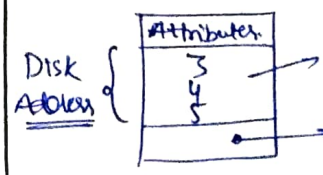


The blocks alloted to file A are 3,4,5,8 in physical m/m (HD)

→ If index block is not completely filled then internal fragmentation occur and we can't manage this.

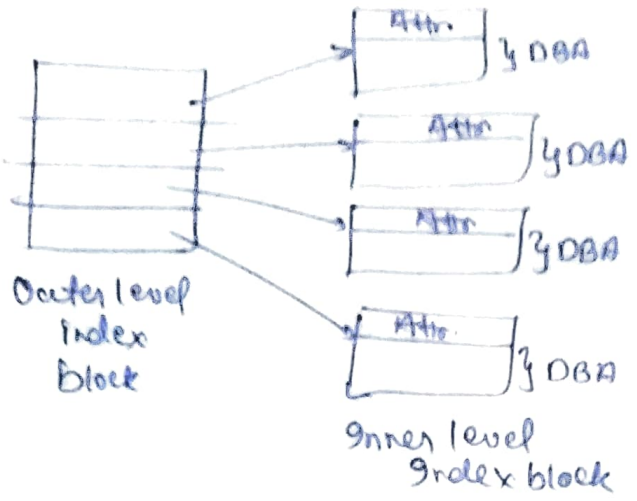
→ But if it is too small in size, however it will not be able to hold enough pointers to hold for a large file.

① Linked schema : we can link several index blocks.



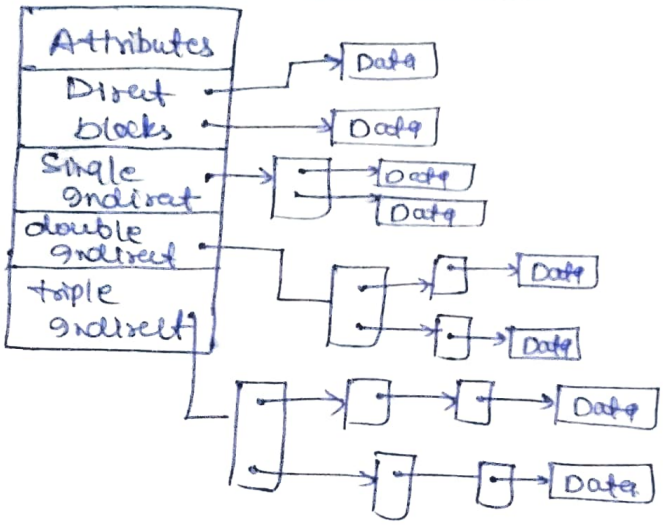
→ Maintain multiple inode for a single file.

② Multilevel Index



→ Inner Index block contains attributes and Disk Block Addresses.

③ Combined scheme → Used in UNIX [Remember the diagram]



This way we can use store huge index file in just one block.

Gate 2004 [Solve it using diagram] [Do it by reading question]

Direct blocks → 10 entries points

Single indirect → $\frac{1024B}{4Bbits} = \frac{1024B}{4B}$

$= 170$ pointers

Double Indirect $= (170 \times 170)$

triple Indirect $= (170 \times 170 \times 170)$

→ No of data blocks = a

$a = (10 + 170 + 170^2 + 170^3)$

→ (size of file) max

$= a \times 1024B$

Note → Above question is necessary to do. It might ask this time.

Gate-2012 : Ques 5.29 in OS Made easy PYQ

Freespace Management

The first responsibility of file system is allocate blocks to each file and keep track of it.

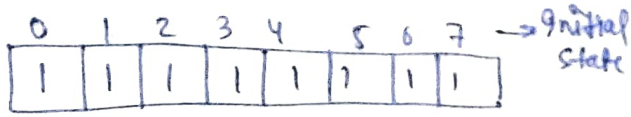
The second responsibility is to manage how manage blocks are free for next allocation. The method used is -

① Bit Vector [Array of bits]

The free space list is implemented as a bit map or bit vector.

Each block is represented by 1 bit.

If the block is free bit is 1 else bit is 0.



Say 2, 4, 5 blocks get filled then,