Virtual Memory

Virtual memory is a technique that allows the execution of processes that may not be completely in memory.

Here Program can be larger than main memory.

overlays & dynamic loading can work around same like Virtual memory but they need extra effort by programmer and limits the size of program to the size of physical memory.

Entire program not needed or not needed at same time.

Benifits of virtual memory -

(1) A program would no constrained by the physical memory available. Program may be as large as Virtual address space.

(2) each program takes less physical memory, degree of multiprogramming can be increased.

(3) less I/o would needed to load or swap part of programs, so each user program run faster.

#Virtual Memory is implemented -, Paging / demand Segmentation. But Segments are of vaiable sized so demand segmentation is more complex.

Demand Paging - In this a lazy swapper is used, which never swaps a page into memory unless that page will be needed.

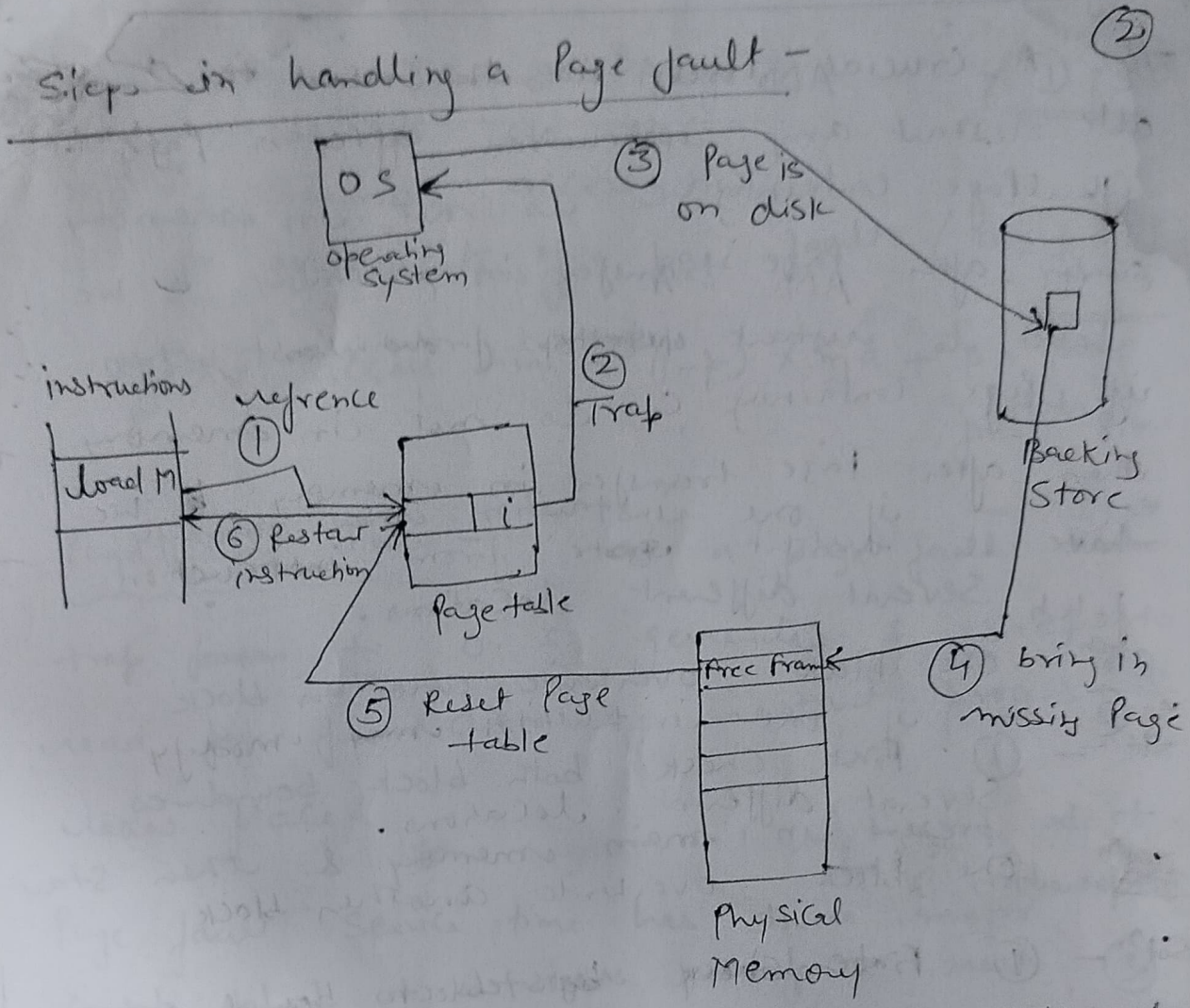
A Process is collection of Pages, when we want to execute Process Pager select Page to load into memory. Pager guess which Page will needed & load only those Pages reducing swap time & needed Physical memory

Valid - invalid bit of Page table is used to define which page is in the memory & which page not loaded in memory.

Valid → Page is on main memory
invalid → either on disk or not in logical address space of Process.

if Page requested that is not in memory is called Page fault.

Steps in handling a Page fault -



after invalid reference Process is interrupted until the desired Page is in memory & resume back its operation from last stage.

H/W to support demand Paging -

1. Page table
2. Secondry memory. (Swap space)

A Crucial issue is the need to be able restart any instruction after a Page fault

Eg ① - ADD A B C
op1 op2 destination

if Page containing C is not in memory then after Page transfer in memory we have to restart ~~ins~~ from instruction fetch.

Eg ② — if one instruction may modify Several different locations.

~~Soln~~ One block overwrite another block

Soln — ① First check both block boundries to be present in main memory & then start operation

② use temporary registers to hold old values.

Performance of Demand Paging –

ma – memory access time
p – probability of a page fault ($0 \le p \le 1$)

effective access time = $(1-p) \times ma + p \times$ page fault time.

Page fault time – as a page fault occurs trap to O.S. generates & steps to read page from disk & restart process takes place.

Page fault servic time has three major components –

1. Servic the page fault interrupt
2. Read in the Page
3. Restart the process

Example – avg. page faul servic time = 25 msec
ma = 100 nsec

effective access time = $(1-p) \times 100 + p \times 25$ msec
$= (1-p) \times 100 + p \times 25,000,000$
$= 100 + 24,999,900 \times p$

eff. access time $\propto p$

effective access time is directly prop. to page fault rate.

if we want less than 10% degradation -

$110 > 100 + 25000000 \times p$ (here $ma = 100$)

$$ma + \frac{ma \times 10}{100} > (\text{effective access time})$$

$10 > 25000000 \times p$.

$p < \frac{10}{25000000} \Rightarrow p < 0.0000004$

means 1 page fault out of 25000000 memory access will slow down 10%.

page fault rate should be low otherwise the effective access time increases and slowing process execution.

Page replacement -

Demand paging concept used to increase -

(a) System throughput
(b) efficient utilization of memory.
(c) increase degree of multiprogramming.

Problem - if all frames are occupied & a process creates page fault then ?

Solⁿ - (1) Terminate Process, not a good idea
(2) Swap a process to disk & free all of its frames
(3) replace a page with new page.

Page fault Servie routine Changed -

(1) find a free frame
   a) free frame available, use it
   b) otherwise, find a victim frame
   c) write victime frame on disk & inquire page on ~~disk~~ main memory
   d) Change page tables & frame table

extra step

#if no free frames, two page transfers are required & page fault service time doubles increasing effective access time.

Sol^{n} - each frame have a bit associated in the hardware called Modify bit (dirty).

i) if frame is modified this bit is set. When page selected as victim its dirty bit checked if it is set, this frame copied on disk. Otherwise it is just overwritten by new page because its unmodified copy is already on disk.

Two major problems must be solved to implement demand paging -

① frame-allocation algo - how many frames to allocate each process in memory.

② Page replacement algo - select the frames (victim) that are to be replaced.

Page Replacement Algo's -

(1) Best Page replacement algo → lowest Page fault rate

evaluation of algorithm is done by running it on string of memory references & counting No. of page faults.

string of memory references is called reference string.

reference string can be generated by -

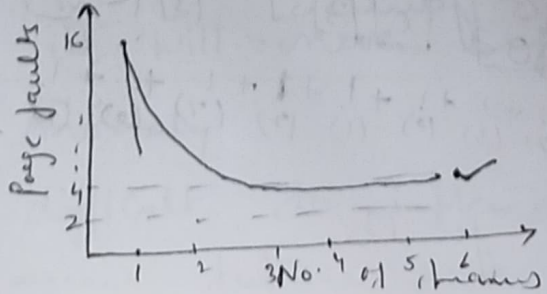
1. Random No. generation
2. tracing & recording a system

to evaluation process we need.

1. Refrence string with only Page No. Is not addresses.
2. No. of frames available

immediate refrence of a page will never cause a page fault.

#



← Page fault decreases with increasing frames

Reference string →

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

Memory with → 3 frames.

① FIFO Algo — ① Simplest
② Easy to implement
③ Performance Not always good

either time is associated with each page & oldest page is replaced

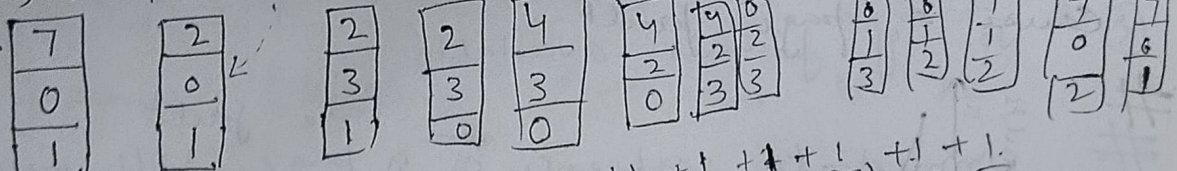
OR

queue is maintained, New pages added at the tail. Page at the head of queue is replaced.

∴ actively used page may be replaced causing extra page fault.

Example –

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

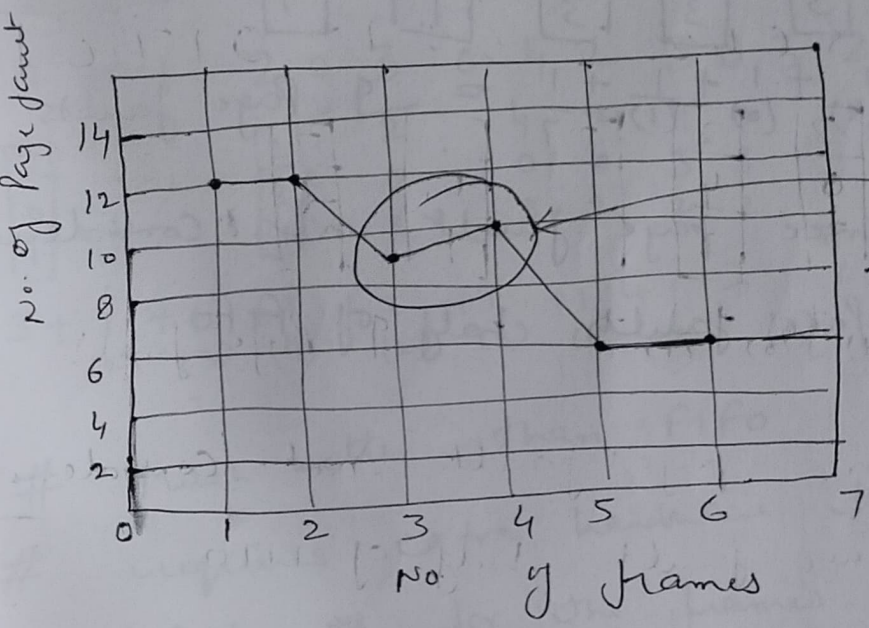


3 Page fault + 1 (2) + 1 (3) + 1 (0) + 1 (4) + 1 (2) + 1 (3) + 1 (0) + 1 (1) + 1 (2) + 1 (7) + 1 (0) + 1 (1)

= 15 Page faults.

Belady's Anomaly – it reflects the fact that for some page replacement algo's, the page fault rate may increase as the No. of allocated frames increases.

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



(9) **Optimal Algorithm** –

(1) it has the lowest Page fault rate of all algo's

(2) it will never suffer from Belady's Anomaly

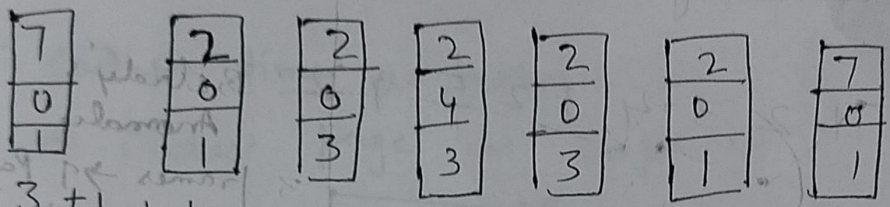
(3) difficult to implement, because it requires future knowledge of reference string

Rule - Replace the Page that will not be used for the longest Period of time.

(4) it is used for Comparison studies.

Example

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



3 + 1 + 1 + 1 + 1 + 1 + 1 = 9 Page faults
(2) (3) (4) (0) (1) 7

if first three page fault not considered it gives Page faults half of Fifo.

3. LRU Algorithm - Replace the page that has not been used for the longest period of time, least recently used approach. (7)

This algo is like optimal algo but looking backward in time, rather than forward.

example -

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

3 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 12 Page faults
(2) (3) (4) (2) (3) (0) (1) (0) (7)

Much better than FIFO

requires extra hardware to determine an Order for the frames acc. to their last time use.

Two Implementations are feasible -

(1) Counters - (1) Associate A time-of-use field with each Page table entry

(2) A counter added to CPU that is incremented with each memory reference

(3) Whenever a page is referenced, Counter value is copied in time-of-use field.

(4) replace the page with smallest time-of-use field value.

∴ Searching for LRU page in page table is required.

∴ Memory access (extra) to write time of use field, is needed.

example CPU Counter = 5 (old) & 5++ (new)
Next time 6 & 6++

Time of use

Page 1	5	← LRU
Page 2	6	← MRU

(2) Stack –

(1) Maintain stack of page No's.
(2) Whenever a page is referenced, put it top of Stack
(3) Bottom of stack denote LRU page.

implement by a doubly linked list with head & tail pointer.
replace the page pointed by tail pointer.

∴ These both LRU implementations need updations of clock fields or stack at every memory reference Cont'd..

Contd.. so it would slow every memory refrence.

NOTE

There is a class of page replacement algo's that never suffers by Belady's Anomaly, called Stack Algo's.

Definition - An Algo for which it can be shown that the set of pages in memory for n frames is always a subset of the set of pages that would be in memory with n+1 frames.

Example - LRU is stack algo because it contains only MRU pages. if frams increases these pages still be in memory because these are most recently used.

---

(4) LRU Approximation Algo -

:) Need less H/w support than LRU Algo.

Refrence bit - # This bit is set(1) if that page is refrenced.

This bit is associated with each entry in page table.

Initially it use will Not be clear

According to reference bit Usage LRU Algo is of three types -

1. Additional - reference - bits Algo.
2. Second - Chance Algo
3. Enhanced Second chance Algo.

(1) Additional - Reference - bits Algo -

For each Page A table entry Aj 8 bit byte is maintained in memory.

Page No.	byte
0	10010011
1	
2	

Table

At regular interval O.S. store reference bit in this table at higher order position, shifted all the bits at right & discarding lower bit.

example - for page 0 reference bit is 1 & previous page entry in table of Aj is 00100111 then OS store New value as -

New value

1 00 1 00 11 1

Inserted reference bit discarded

if table entry for a page is 00000000 (9) then this page is not used from 8 time intervals.

8 bit byte considered as unsigned integer then lowest value gives LRU Page. or victim page.

(2) Second - Chance Algo - (Clock Algo.)

basic Algo is Fifo for this Algo.

Inspect reference bit in Fifo order - if this bit is 0, replace the page but if this bit is 1, give second chance to this page. Means

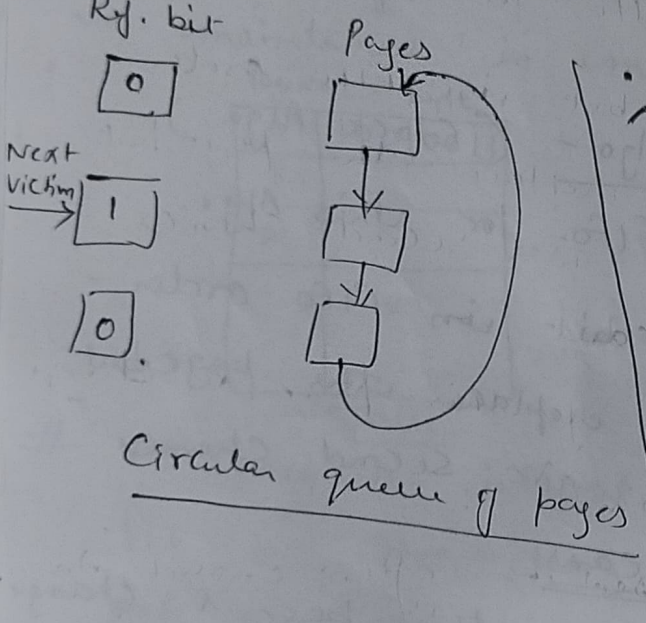
reset(0) the value of this page & change time to current time.

if a page is used ~~to~~ often enough to keep its reference bit set, it will Never be replaced.

implement Algo with Circular queue.

Example - (1) A pointer points which page is to be replaced next

(2) When A frame is needed start from, pointer to search victim if reference bit is 1 give second chance & as soon as reference bit 0 page found, replace that page & pointer will incremented by one because next page is victim for next round.



Circular queue of pages

∴ in the worst case when all the ref. bits are set this algo converts into FIFO because it will give second chance to every page.

(3) Enhanced second chance Algo - (10)

Consider reference bit & Modify bit as pair. Following four classes are possible -

(1) (0,0) neither recently used, nor modified (best Page to replace)

(2) (0,1) Not recently used but Modified (Page will need to written out on disk before replacement)

(3) (1,0) , recently used but Not modified

(4) (1,1) Used & Modified. (Probably used again in future)

Replacement ORDER

Process - Replace the page in order from (1,1) value. ∧ (0,0) Value to (Reset Ref. bit 1 to 0 Works same as Clock Algo)

∴ ∧ We May have to scan Circular queue Several times before we find page to be replaced.

Example - Suppose there is No (0,0) Pair but (0,1) Pair then replace it if (0,1) also not present replace (1,0) Pair page if there is No Such Page then

finally replace page with (1,1) pair.

(∵ Modify bit also concerned to reduce I/O.

(5) Counting Algo -

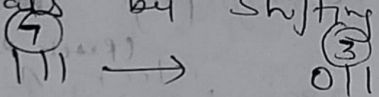
Counter - Counts No. of total references to that page.

(A) LFU Algo - least frequently used page with Smallest Count value is replaced.

Logic - Actively used page have large counter.

Nc Vic ∴ A page used heavily in starting but not in future have larger Counter value but not replaced.

Solⁿ - Counter is decremented at regular intervals by shifting its value right.



before shifting after shifting

(2) MFU Algo - larger Count value page is replaced.

Logic - it is used & Smaller Count value page brought in memory recently & has yet to be used.

∴ Counting algo is expensive because it maintains counter for each page. (11)

(6) Page Buffering Algo -

(1) # Pool of free frames in the system is maintained.

Page replacement Algo find victim but new page is written on frame from pool so process can run fast.
after that victim is copied on disk & added to pool.

∴ Process not wait to writting process of victim to disk to Restart up.

(2) # Maintain list of Modified pages, whenever Paging device is free, Modified pages are written to disk & Modified bit is reset

∴ At replacement time, clean page finding probability will increase.

(3) # frame is added in pool list but remember which page was in each fram

if this page required again first see from pool that frame containing that page is not reused. & use this frame No. I/o is needed.

frame contents are Not Modified when its contents written on disk & added frame to pool.

☺ in FIFO, if actively used page is replaced then it can be used from frame pool without I/o.

There are always some frames should be available in pool list.

Allocation of Frames -

Single user system - After allocating OS, remaining frames available to user but

Multiuser system - frames must be decided in between all the users.

⟹ Minimum No. of frames -

The minimum No. of frames per process is defined by the instruction Set Architecture

if a page fault occur before executing instruction then that instruction must be restarted.

So we want that all the pages needed to run single instruction should be in memory at a time.

Some instruction sets use indirect addressing scheme that want No. of memory references may be at different - different pages.

if a M/c allows 16 level indirection then at least 16 frames should allocated to the process.

Maximum No. of frames allocated to a process is defined by amount of available physical memory.

Allocation Algo -

(1) m frames, n processes

divide equal share among all processes

$\Rightarrow$ m/n frames

this scheme is called equal allocation.

(2) Proportional allocation - (Basis on size of P)

Process $\rightarrow p_i$

virtual size of $p_i \rightarrow s_i$

frames allocated $\rightarrow a_i$

Total virtual size of all current processes $\rightarrow S$

$$S = \sum s_i$$

frames $\rightarrow m$

Then

$$\boxed{a_i = s_i/S \times m}$$

Example

two processes, 62 frames

$s_1 = 10$ & $s_2 = 127$

$S = 10 + 127 = 137$

$m = 62$

$a_1 = \frac{10}{137} \times 62 \simeq 4$

$a_2 = \frac{127}{137} \times 62 \simeq 57$

Equal allocation case

$a_1 = 31$ - wastage

$a_2 = 31$

multiprogramming increase → No. of frames allocated decreases → Page fault increases.

here high priority process & low priority process treated equally so we can change proportional algo to Consider priority also.

Global Vs local Allocation –

① Global Replacement – (A) Process can Replace any frame even allocated to other process.

(B) No. of frames allocated to a process can increase or decrease

(C) i Process can not control its own page fault rate because other process can replace its page that may be needed in future for it.

~~fa~~ Depends on paging behavior of other processes.

(D) gives greater system throughput because less used pages of other process are used

by active pages of another process.

Example - High priority process can replace pages of low priority process & increase its frame No.

(2) Local Replacement - (A) Each process select page to be replaced from only its own set of allocated frames.

(B) Process is affected by only its own pagin behavior.

(C) If pages are not better utilized another process can not use them. (like Internal fragmentation.)

(D) No. of frames Not Changed.

Thrashing -

High paging activity of a process is called thrashing.

A process is thrashing if it is spending more time paging than executing.

Reason - if process have less frames than its minimum requirement then it will quickly page fault because actively used pages are not in memory.

Cause of Thrashing -

Condition - (1) global page replacement is used
(2) OS monitoring CPU utilization.

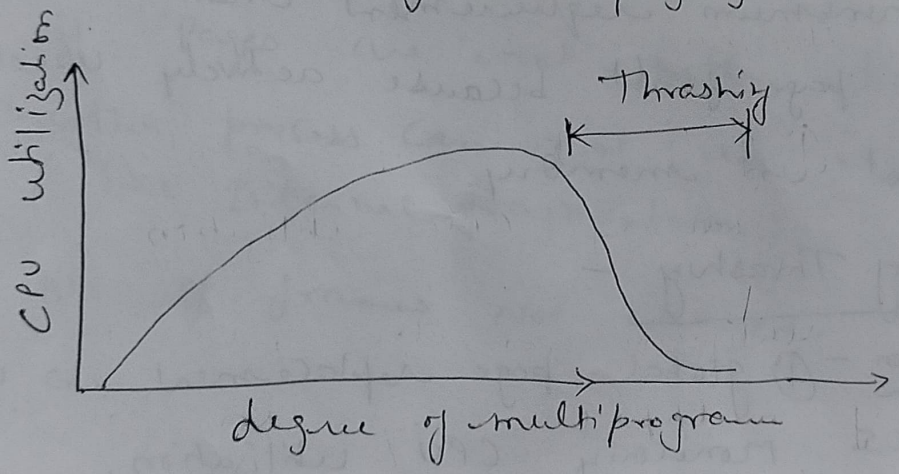
if CPU utilization is low, we increase degree of multiprogramming. New process starts faulting & takes pages from other processes. the page replaced may be active page for that process so that process also fault for that page. it takes frame from other process and so on.

all these processes wait in the paging device queue for ~~pag~~ service

page fault, so ready queue becomes empty & CPU utilization decreases.

CPU Scheduler sees low CPU utilization so it increases degree of multiprogramming. it causes more page fault.

This process goes on & page fault rate increases tremendously. effective access time increases & almost all process spending in paging.



⇒ All thrashing processes wait in paging device queue so, if a process that is not thrashing but create a page fault, its page fault service time also increase, increasing effective access time for process that is not thrashing

local replacement may be a solution to limit the thrashing.

Locality Model – states that as a process executes, it moves from locality to locality.

Locality – is a set of pages that are actively used together.

A program is composed of several different localities that may overlap.

example – A process running Add function need pages containg code of Add, its local & global variable. after execution suppose it calls subtract function now it need different page set means different locality.

allocate ENOUGH frames to contain process locality, otherwise it will thrash for actively used pages.

Working Set Model - it is based on assumption of locality.

Δ (Parameter) → define working set window

Set of pages referenced in Δ page refrences (Recent) is working set (WS).

Example -

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

t_1 t_2

if Δ = 10 then at time t_i

$WS(t_1) = \{1, 2, 3\}$

$WS(t_2) = \{2, 4, 5, 6\}$

Working set Contains only actively used pages

accuracy of working set depends on selection of Δ, if Δ is too small, it will not encompass the entire locality.
if Δ is too large it will overlap several localities.

Working set size (WSS) of a process means No. of pages in working set.

$D = \sum WSS_i$

$D \rightarrow$ total demand of frames

$WSS_i \rightarrow i^{th}$ process needs WSS frames.

$m \rightarrow$ total memory frames

if $D > m$ then thrashing will occur because some processes not getting minimum frames.

OS gives frame to the processes accordig to WSS_i, if 0 extra frames available degree of multiprogramming may be increased OR

if $D > m$ some processes suspended & their free frames available to thrashing processes.

it makes degree of programming as high as possible, increasing CPU utilization.

working set window is moving window

Working set find

① Δ Time
② A Reference bit

Δ is 10000 references than suppose after each 5000 references interrupt occur & reference bit for each page is copied in memory & cleared.

Now if page fault occur we check reference bit of page & previous reference bit stored in memory. if one of these bit is set means page is actively used & present in working set.

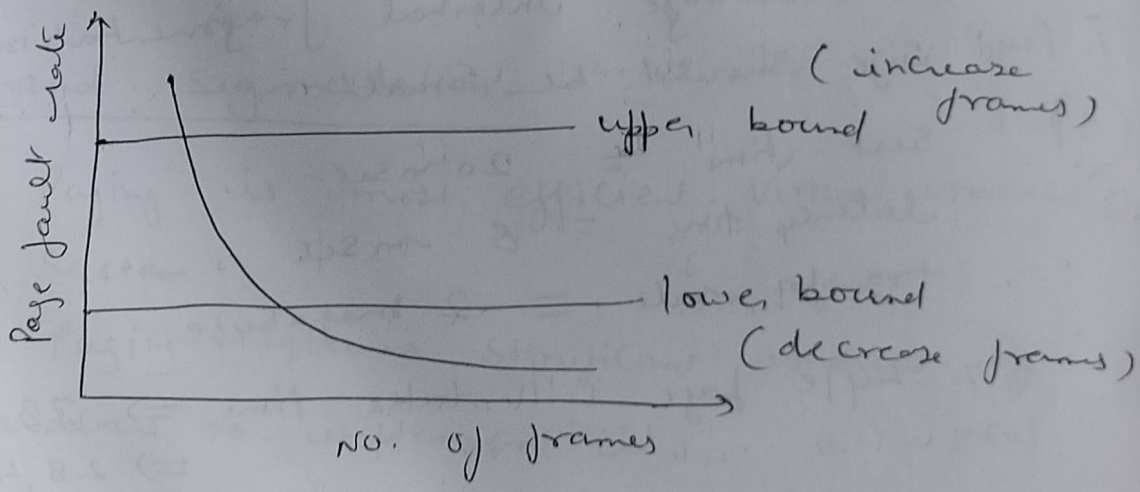
Page fault frequency –

working set Model is complex. to implement.

Concept =

if Page fault rate is too high means process needs more frames.

if Page fault rate is too low means process have extra frames & we may remove a frame from that process.



Page fault rate is more than upper bound & no free frame available than suspend a process & frames of that process allocated to thrashing process.

Prepaging - ① Pure demand Paging Create more page fault.

Prepaging means bring all the pages that will be needed into memory at one time.

needed pages are decided according to working set.

Page Size -

① Page size should be larger so page table size will be small

② To minimize internal fragmentation Page size should be small

③ Seek time = 20 msec.
latency time = 8 msec.
transfer rate = 2 megabyte

512 byte page will take time ⇒ 28.2 msec
1024 " " " ⇒ 28.4 msec.

means large page size will take minimum I/O time.

∴ but larger page size, transfer data needed but also data that is not needed but written on that page & require more memory.

(4) To minimize the No. of page faults, we need to have large page size.

Ex- 2 data written on different page create two page faults while written on one page create one page fault.

Demand Segmentation —

∴ Paging is most efficient virtual memory system.

∴ Paging requires significant amount of Hardware to implement it.

OS/2 — this operating system does not have Paging H/W but have segment H/W so here demand segmentation is implemented.

#OS/2 allocates memory is Segments rather than in pages.

Segment descriptor –

if invalid means Segment fault

Segment size	Protection	location	Valid bit

it is maintained for each process, When Segment addressed, Valid bit is checked if it is invalid means segment is not in memory & (Segment fault) Segment replacement takes place.

⇒ Trap to OS is generated to replace Segment.

Segment replacement –

Accessed bit in segment descriptor work same as reference bit in paging.

it is set whenever that segment is accessed.

After a time slice all the segments with accessed bit set are placed at the head

of the queue (queue contain any ~~is maintained~~ for each segment in memory)

Whenever segment fault occur, segment at the tail of the queue is replaced because of segments at head of queue are segments recently accessed.

Compaction - if segment ~~replace~~ Place vacated have smaller than new segment size than Compaction takes place.

Even now enough space not available than another segment replaced & compaction takes place.

i) demand segmentation requires more overhead. but better than no virtual memory.