Chapter - 9 Chandrika. C.P.

Virtual Memory.

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

Advantages:-

1) A program doesn't depends on the physical memory that is available. users would be able to write programs for an extremely large virtual-address space, simplifying the program task.

2) Since user program is taking less physical memory, more programs could run at the same time which increases the CPU utilization & throughput.

3) less I/O would be needed to load or swap each user program into memory, so each user program would run faster.

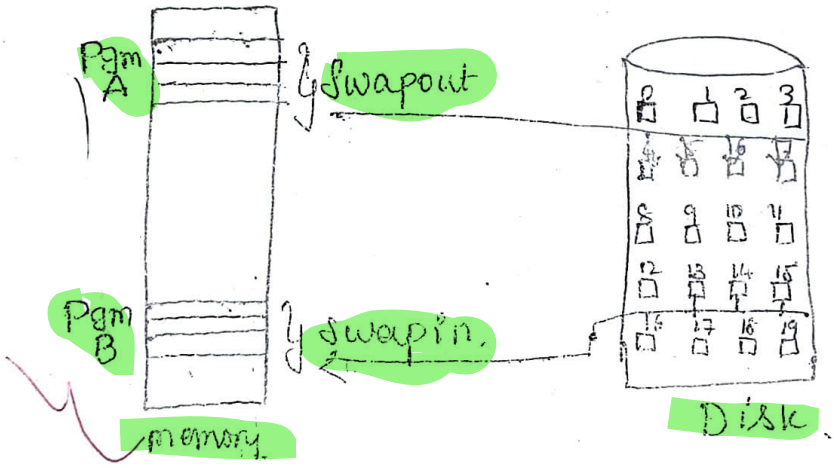
Thus virtual memory is the separation of user logical memory from physical memory.

Virtual memory is commonly implemented by demand paging.

With a help of neat diagram briefly explain Demand Paging.

* Demand paging:

Demand paging is similar to the paging system with swapping Processes reside on secondary memory (which is usually a disk) When we want to execute a process, we swap it into me which is shown in the fig below.

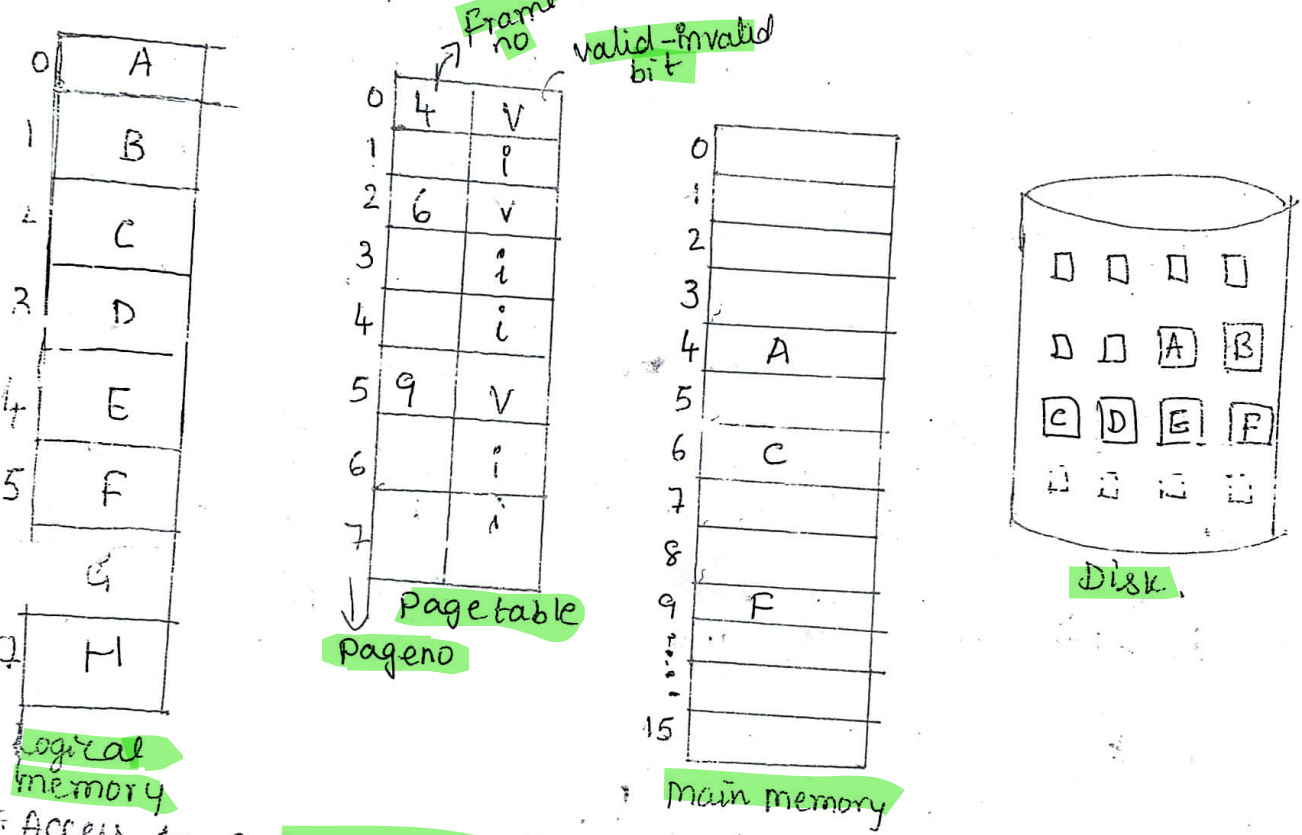


* To distinguish b/w the pages that are in memory & those are in the disk, we use valid-invalid bit scheme.

* When the bit is "valid" indicates that the page is in memory & is a legal page. If the bit is invalid indicates that the page either is not valid i.e not in the logical address space of the process or is valid but not currently on the disk.

* The page table shows which pages are valid & which are invalid

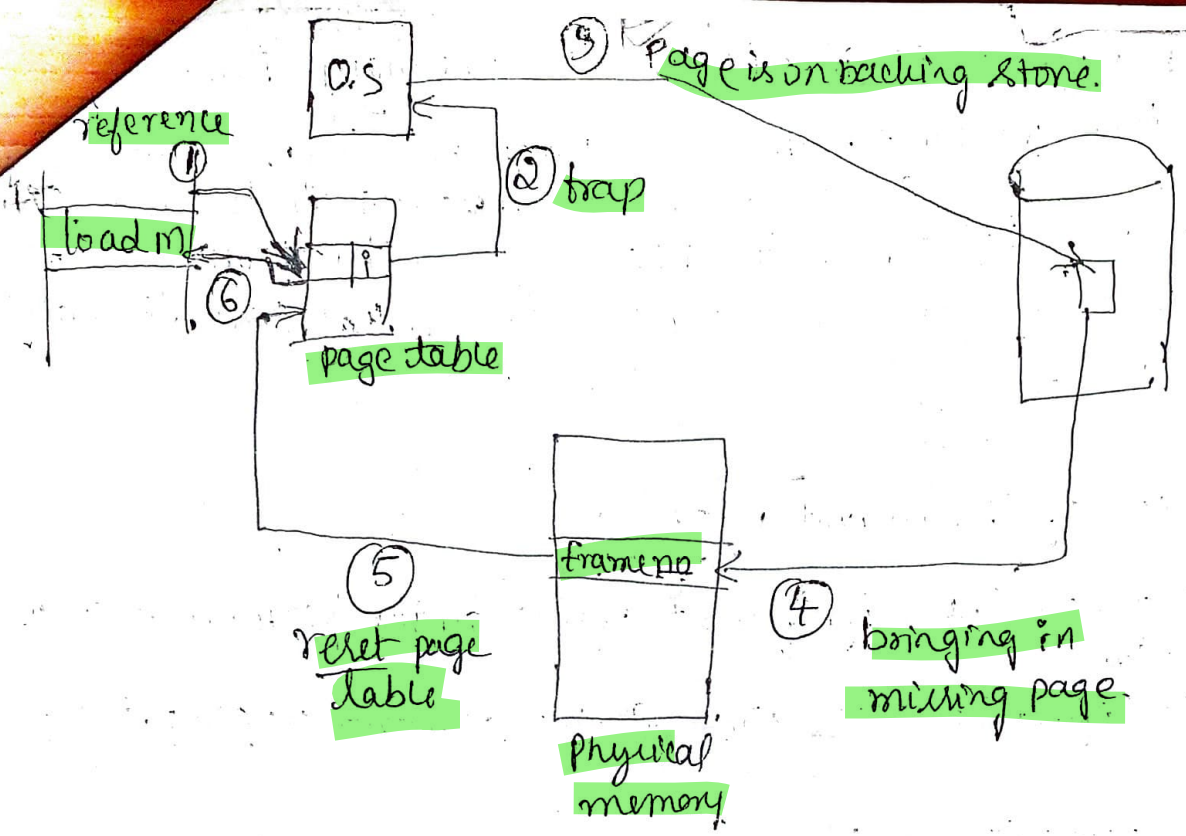
* The fig below shows the page table when the pages are not in memory



* Access to a page marked as invalid causes a page fault trap.

The procedure for handling page fault is as follows.

1) we check internal table for this process to determine whether reference is valid or invalid memory access.
2) If reference is invalid the process terminates. If the reference is valid but not yet brought in that page but later we can brought
3) we find free frames
4) we schedule a disk operation to read the desired page into the newly allocated frame.
5) When the disk read is complete, we modify the internal table kept with the process & the page table to indicate that the page is now in memory.
6) We restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory.



Pure demand paging:- If all the pages of a process are placed in disk not in memory, when we execute that process, then it leads to page fault & this page fault continues until all the pages that are in disk are brought into memory. This is known as pure demand paging.

hardware

H/w Support for demand paging:- The h/w to support demand paging is the same as the h/w for paging & swapping.

1) Page table:- This table has the ability to mark an entry invalid through a valid-invalid bit or special value of protection

2) Secondary memory:- This memory holds those pages that are not present in main memory. The secondary memory is usually a high speed disk, known as the swap device. The disk used for this purpose known as swap space or backing store.

software

S/w Support for demand paging:- In addition to H/w, S/w support is also needed which includes some constraints (conditions) that must be imposed on demand paging.

a) A page fault could occur at any memory reference. If the page fault occurs on the instruction fetch, we can restart by fetching the instruction again. If a page fault occurs while we are fetching an operand, we must re-fetch the instruction, decode it again & then fetch the operand.

then we have to bring the page from disk
correct the page table & restart the instruction, in which
this restart will start from the begining of instruction.
∴ the sw has to put some constraints so that it won't
from the begining.

Performance of Demand paging :-

a) Demand paging has the significant effect on the performance of a computer system

b) To calculate effective access time for the demand paging, the formula is

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

where p is the probability of page fault $(0 \leq p \leq 1)$
ma is the memory access time
Page fault time :- is a time needed to service a page fault

c) A page fault causes the following sequence to occur
1) Trap to the O.S
2) Save the user regr & process state
3) Determine that the interrupt was a page fault
4) Check that page reference was legal & determine the location of the page on the disk
5) Issue a read from the disk to a free frame
*) Wait in a queue for this device until the read request is serviced
*) Wait for the device seek / latency time
*) Begin the transfer of a page to a free frame

d (next point after 12)

There are 3 major components of the page-fault service time
1. Service the page-fault interrupt.
2. Read in the page
3. Restart the process.)

6.) While waiting, allocate the CPU to some other user
7) Interrupt from the disk

for the CPU to be allocated to this process again.
& restore the user regr, process state & new page table, then resume the interrupted instruction.

c) for eg if page fault service time = 25 milliseconds
& memory access time = 100 nanoseconds

then effective access time = (1−P) x 100 + P (25 milliseconds)
= (1−P) x 100 + P x 25,000,000
= 100 + 24,999,900 x P

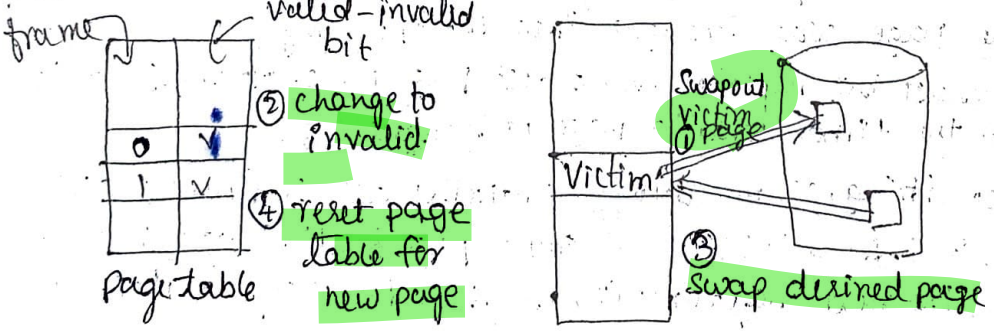
i.e effective access time is directly propotional to the page fault rate.

With a neat diagram, demonstrate the need for page replacement during paging activity.

* Page Replacement :-

Page Replacement takes the following approach :

1) If no frame is free, we find one that is not currently being used & free it.
2) we can free a frame by writing its contents to swap space & changing the page table to indicate that the page is no longer in memory.
3) The freed frame can now be used to hold the page for which the process faulted.
4) The page-faulted service routine is now modified to include page replacement as follows :



a) Find the location of the desired page on the disk.
b) Find a free frame
   i) If there is a free frame, use it.
   ii) Otherwise, use a page replacement algorithm to select a victim frame.
   iii) Write the victim page to the disk; change the page & frame tables accordingly.
c) Read the desired page into a free frame; change the page & frame tables

5) Notice that, if no frames are free, two page transfer (page-out) are required. This situation effectively doubles the page service time & will increase the effective access time accordingly.

6) ~~This situation effectively doubles the page fault~~

This overhead can be reduced by the use of modify bit. The modify bit for each page is set by the HW: whenever any word or byte in the page is written into, indicates that page has been modified. When we select page for replacement, we examine its modify bit.

7) If the bit is set, we know that page has been modified since it was read in from disk. In this case we must write page to the disk.

8) If the bit is not set page has not modify since it was read into memory. This avoid writing the memory page to the disk if it is already there.

9) Page replacement is basic to demand paging, it completes the separation b/w logical memory & physical memory with this mechanism, a very large virtual memory can be provided for programmers on a smaller physical memory.

Page Replacement Algorithm:-

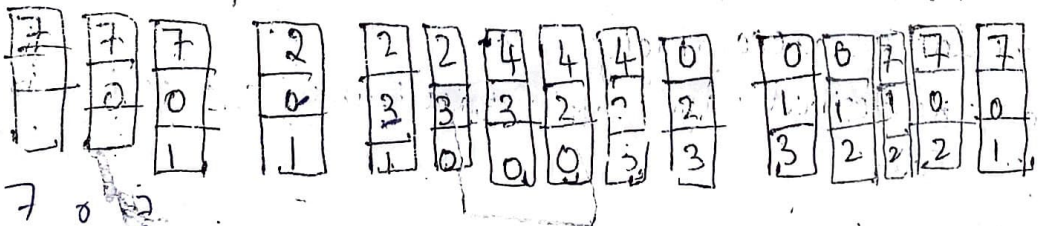
FIFO Algorithm:- 1) The simplest page-replacement algorithm is a FIFO algorithm. The FIFO replacement algorithm is associates with each page the time when the page was brought into the memory.

a) When a page must be replaced, the oldest page is choosen, so it is necessary to record the time when a page is brought in.

3) For eg: Consider reference string given below. Intially all 3 frames are empty. The first 3 references (7,0,1) cause page fault & are brought into these 3 frames. The next reference (2) replace the page 7 because page 7 is brought in first. Since 0 is the next reference it is already in memory ∴ therefore no fault for this reference. The reference to 3 replaces the page 0 & reference to 0 replaces the page 1 & so on.

∴ we get 15 faults altogether

7 0 1 2 (0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1) — Reference string



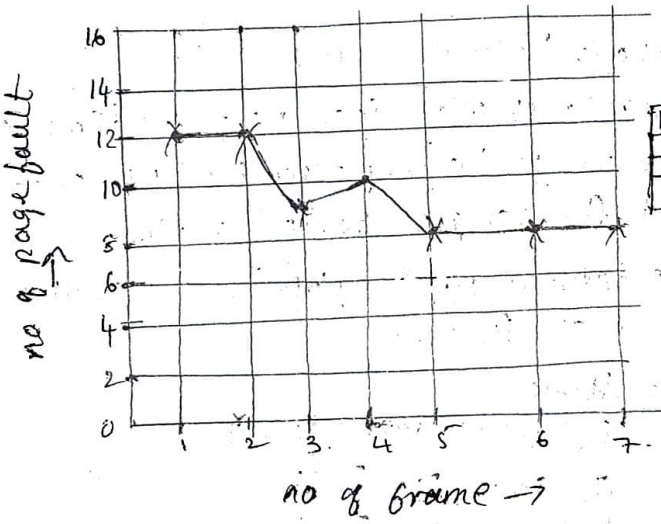
Advantage :- FIFO page replacement algorithm is easy to understand program.

Disadvantage :-

1) Its performance is not always good.

2) The fig below shows the page fault curve for FIFO, ~~the curve shows that~~ the curve indicates page fault vs no of available frames.

The no of faults for 4 frames (10) is greater than no of faults for 3 frames (9). This result is unexpected & is known as Belady's anomaly. Belady's reflect the fact that for some page-replacement algorithm the page fault may increase as the no of allocated frame increases.



This is for the reference string

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

if available frame = 4, then

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

the no of page fault is 10

Optimal algorithm :-

*) One result of ~~discover~~ Belady's anomaly leads to the discovery of optimal page replacement algorithm. optimal page replacement will not suffer from Belady's anomaly. An optimal page replacement algorithm has the lowest page fault rate of all algorithm.

*) OPT states that

Replace the page ~~fault~~ that will not be used for longer time

*) for eg

Consider the reference string

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

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

algorithm would yield nine page faults. as shown in fig. The

3 reference causes faults that fill the 3 empty frames. The reference page 2 replace page 7, because 7 will not be used until reference whereas page 0 will be at 5, & 1 at 14. Reference to page 3 replaces page 1 at 14. Reference to page 3 replaces page 1 as page 1 will be last of the 3 pages in memory to be referenced again with only nine page faults.

Advantage:- *) Optimal algorithm is better than a FIFO algorithm which had 15 faults, where optimal with only 9 page fault.

Disadvantage:- *) Optimal page replacement algorithm is difficult to implement, because it requires future knowledge of reference string.

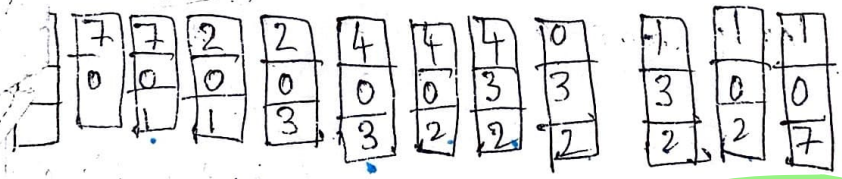
LRU Algorithm:- (Least Recently used)

* LRU replacement associates with each page the time of that page's last use. When a page must be replaced the LRU chooses the page that has not been used for longest period of time.

This strategy is the optimal page-replacement algorithm looking backward in time, rather than forward.

the same reference string

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



The LRU algorithm produces 12 faults. The first 5 are as same as the optimal replacement. When reference to page 4 occurs LRU replacements sees that of the 3 frames in memory, page 2 was used least recently. The most recently used is page 0 & ~~therefore that page~~ just before page 3 was used.

* LRU algorithm places page 2, not knowing page 2 is about to be used. When it then faults for page 2 the LRU algorithm replaces the page 3.

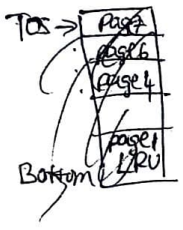
¥. LRU algorithm is considered to be quite good.

* LRU page-replacement algorithm may require substantial hardware assistance. The problem is to determine an order for the frames ~~determine an order~~ by the time of last used.

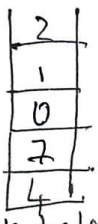
can be implemented by.

counters:- In the simplest case, we associate with each page table entry a time of use field, & add to the CPU or logical clock. counters. The clock is incremented for every memory reference. whenever a reference to a page is made, the contents of clock register are copied to the time-of-use field in the page table for that page.

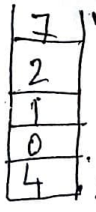
Stack:- Another approach to implementing LRU replacement is to keep a stack of page no. whenever page is referenced, it is removed from the stack & put on the top. The top of the stack is always the most recently used page & the bottom is the LRU page. & top of the stack will be the most referenced page.



reference string 4 7 0 7 1 0 1 2 1 2 7 1 2



Stack before a



Stack after a.

LRU Approximation Algorithm:-

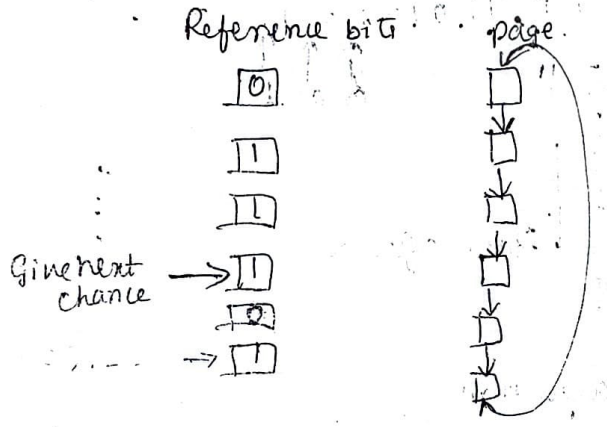
1) Addition-reference-bits algorithm
2) Second chance algorithm
3) Enhanced second chance algorithm

Addition Reference bits Algorithm:-

* we gain additional ordering information by recording reference bits in regular intervals.
* In regular intervals, a timer interrupt transfer the control to the O.S
* The O.S shifts the reference bit for each page into higher order 8 bit byte, shifts remaining bits to right one bit & discarding the low-order bit.
* These 8 bit shift register contain history of page used for last 8 time periods. If 8 bit shift register contain 00000000 then the page is not used for 8 time periods. If page is used atleast once each period would have a shift register val of 1111 1111

Second chance algorithm:-

* The basic algorithm of second chance is FIFO replacement algorithm
* When a page is selected we check the reference bit. If reference bit is ~~selected~~ set to 0 we proceed to replace the page. If reference bit is set to 1 give a second chance to page & move to select next FIFO page.
* once page gets second chance, its reference bit is cleared & its arrival time is reset to current time.
* one way to implement the second chance algorithm is to use the circular queue. The pointer indicates ~~within~~ which page is to be replaced next



* when frame is needed, pointer advances until it finds a page with reference bit 0 as it advances it clear the referencebit
* once a victim page is found, page is replaced & the new page is inserted in the circular queue in that position.
* Second-Chance algorithm degenerates to FIFO replacement if all bits are set

Enhanced Second chance algorithm:-

*) Second chance algorithm can be enhanced by considering both reference bit & modifying bit as an ordered pair.

*) With these 2 bits following four classes are possible
a) (0,0) neither recently used nor modified - best paged to replace.
b) (0,1) not recently used but modified - quite not good because the page will need to be written out before replacing it.
c) (1,1) recently used & modified probably will be used again

This algorithm is used in the Macintosh operating system i.e in virtual memory management scheme.

Counting Algorithm:- In a counting algorithm we keep a counter to count the no of references that have been made to each page which help in developing the following 2 schemes of replacement algorithm.

1) LFU Algorithm:- *) LRU algorithm requires that the page with the smallest count be replaced.

*) The reason for this selection is that actively used page should have a large reference count.

*) This algorithm suffers from the situation in which a page is used heavily during the intial phase of a process, but then is never used again. Since it was used heavily, it has a large count & remains in memory even though it is no longer required.

*) One solution for this algorithm is to shift the counter content right by one bit.

2) MFU:- replace most efficiently used page keeping in mind that the page with smallest count just now brought into main memory

Eg: 93 frames

Allocation of Frames:-