190031187 | NERELLA VENKATA RADHAKRISHNA

Operating Systems Design - 19CS2106S, 19CS2106A
Home Assignment - CO-III

1. Assume that memory contains only three frames. Initially all three frames are empty. The page reference string is 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1. Find how many page faults occurs using FIFO, **Optimal Page replacement, and Least Recently Used** algorithms. Pictorially show which pages are replaced.

190031187
Radhakrishna

Operating System Design
Home Assignment-3

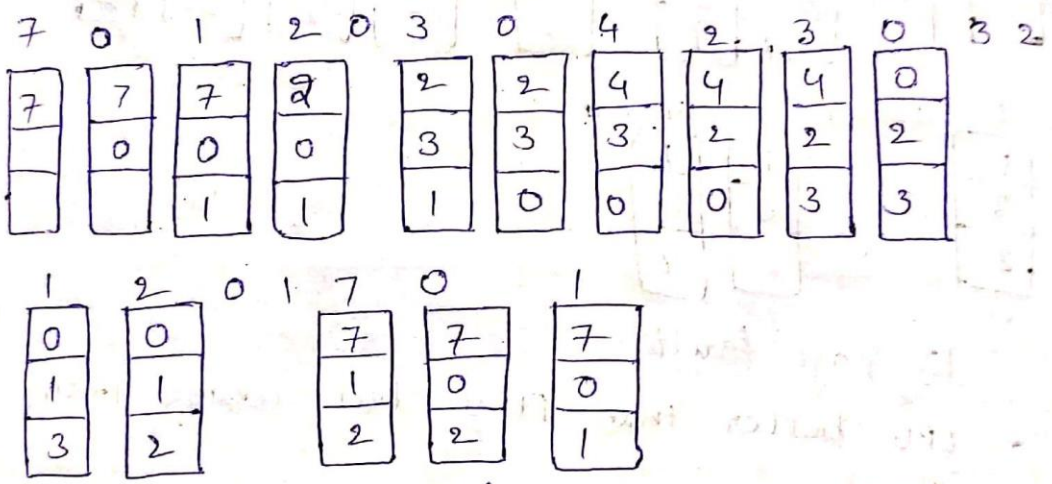
1. If there are free page frames. If not must list some thing. This is called page replacement

Given Reference string:

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

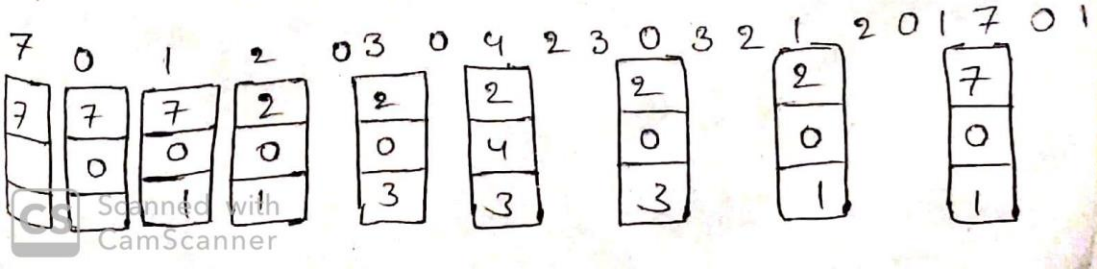
FIFO

→ 3 frames (3 pages can be in memory at a time per process)



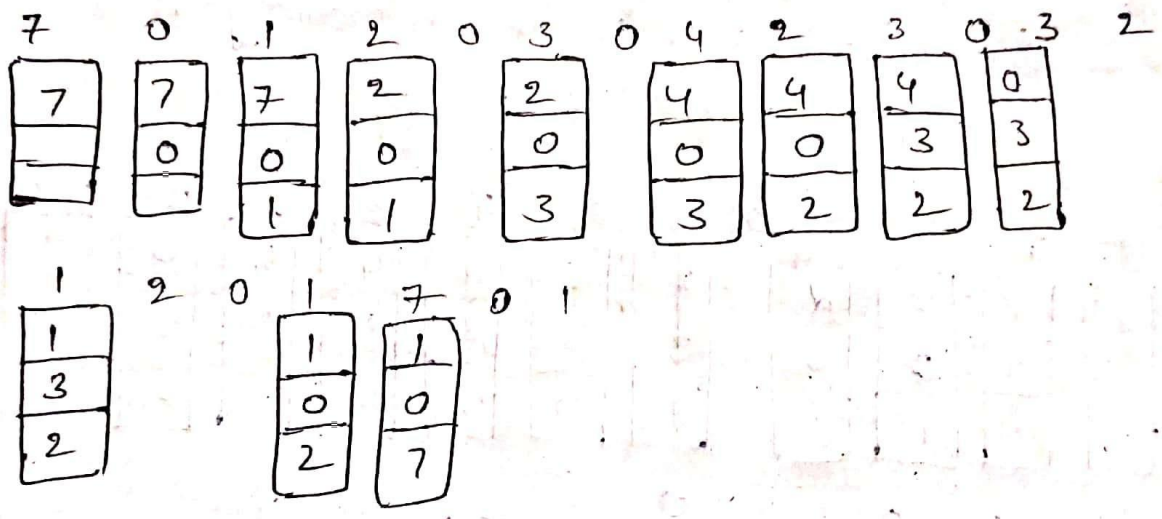
15 page faults

Optimal: Replace page that will not be used for longest period of time



LRU (Least Recently Used)

Reference string:



12 page faults

→ LRU better than FIFO but worse than optimal.

2. Consider the following page reference string:
   1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
   How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.
   - LRU replacement
   - FIFO replacement
   - Optimal replacement
   - Clock replacement

2. Given page reference string:

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

LRU ⟶ Frame-1

20 page faults

Element	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
Frame-1	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6

Frame-2

18 page faults

Element	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3
Frame-1	1		3		2		5		2		NF		7		3		1		3
Frame-2		2		4		1		6		1		3		6		2			

Frame-3

15 page faults

Element	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
Frame-1	1			4			5			1			7			2		NF		
Frame-2		2			NF			6				3			NF				NF	
Frame-3			3			1			2		NF			6						6

Frame-4, 10 page faults

Element	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
Frame-1	1					NF				NF				6						
Frame-2		2			NF				NF		NF					NF		NF		
Frame-3			3				5					3			NF				NF	
Frame-4				4				6					7				1			

Element	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
Frame-1	1					NF				NF							NF			
Frame-2		2			NF				NF		NF					NF		NF		
Frame-3			3									NF			NF				NF	
Frame-4				4																
Frame-5							5													
Frame-6								6						NF						NF
Frame-7													7							

like above we will calculate faults for FIFO, OPTIMAL & CLOCK Replacement pages

	No. of faults.			
	LRU	FIFO	OPTIMAL	CLOCK
Frame-1	20	20	20	20
Frame-2	18	18	15	17
Frame-3	15	16	11	14
Frame-4	10	14	8	10
Frame-5	8	10	7	10
Frame-6	7	10	7	7
Frame-7	7	7	7	7

190031187 | NERELLA VENKATA RADHAKRISHNA

Frame-5, 5 page faults

element	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
Frame-1	1					NF				NF										
Frame-2		2			NF				NF		NF					NF				
Frame-3			3									NF			NF		NF			
Frame-4				4			5						7							
Frame-5								6						NF					NF	

Frame-6, 7 page faults

Element	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
Frame-1	1					NF				NF							NF			
Frame-2		2							NF		NF					NF		NF		
Frame-3			3									NF			NF				NF	
Frame-4				4																
Frame-5							5						7							
Frame-6								6						NF						NF

Frame 7

7 page Faults

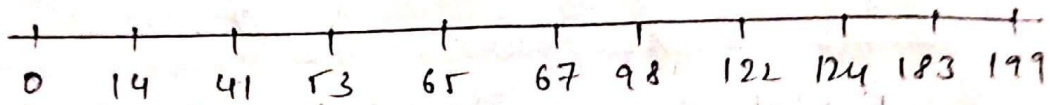
3. Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

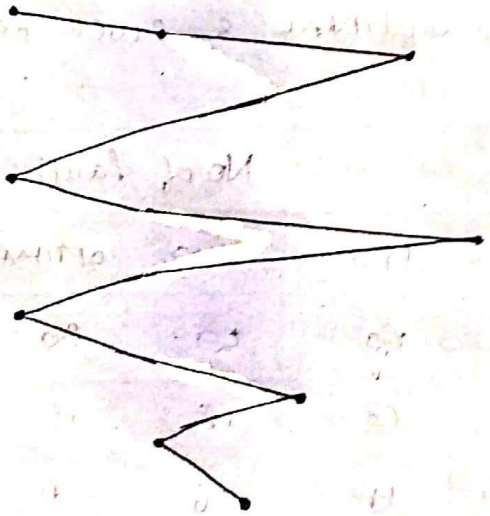
3. Total head movements increased while serving this requests

$= (98-53) + (183-98) + (183-41) + (122-41)$
$+ (122-14) + (124-14) + (124-68) +$
$(67-65) +$

$= 48 + 85 + 142 + 81 + 108 + 110 + 59 + 2$

$= 632$





4. SCAN is an algorithm that will service requests to the nearest end. Usually tracks start at 0 and the end track is given.

For example, if the head is currently at 180 - it will go to 199 and service all requests in that direction.
queue = 98, 183, 37, 122, 14, 124, 65, 67 start = 53 end = 199 For the given data, calculate the total head movement

4. SCAN

queue : 98, 183, 37, 122, 14, 124, 65, 67

Head starts at 53

Total head movements

= (53-37) + (37-14) + (14-0) + (65-0) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124)

= 236

5. C-SCAN algorithm is very similar to SCAN in the sense that it will still go and hit the closest end but it "jumps" to the other end and services going in the opposite direction.
For example, if the head starts at 100 - it will service all requests down to 0. But then "jump" up to 199 and service all requests going back down to 0. You should not include the distance of the "jump" in the total head movements.
queue = 98, 183, 37, 122, 14, 124, 65, 67
start = 53 end = 199
For the given data, calculate the total head movement

5. C-SCAN

queue : 98, 183, 37, 122, 14, 124, 65, 67

Head starts at 53

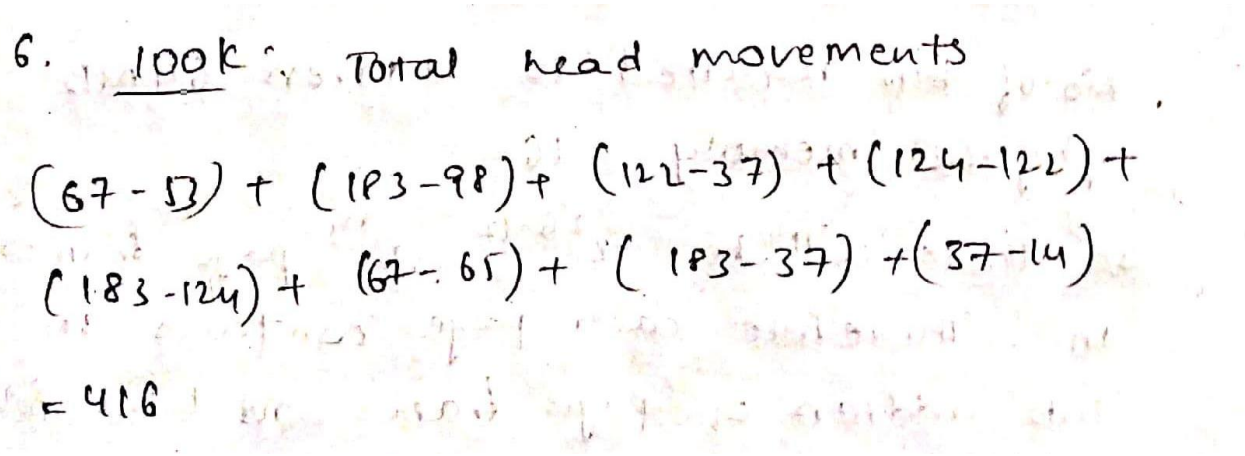
Total head movements

= (65-53) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124) + (199-183) + (199-0) + (14-0) + (37-14)

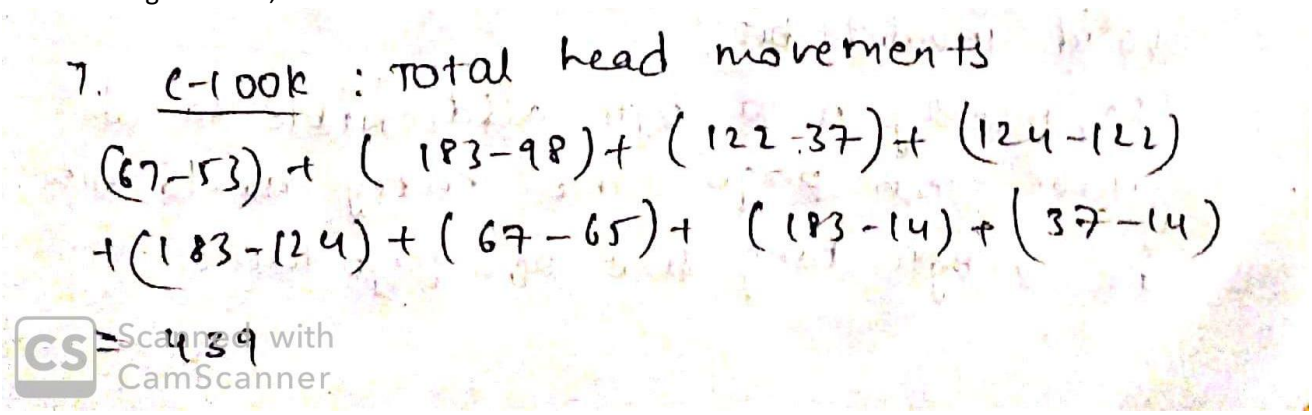
= 382

6. The LOOK algorithm is similar to SCAN but allows you to "look ahead" and not require you to end the end or beginning. This will save you head movements.
queue = 98, 183, 37, 122, 14, 124, 65, 67
start = 53 end = 199

For the given data, calculate the total head movement



7. The C-LOOK algorithm works similar to C-SCAN where it "jumps" but just like in LOOK it will not go to the end. It will "look" ahead to determine where it needs to go and stop.
   queue = 98, 183, 37, 122, 14, 124, 65, 67
   start = 53 end = 199
   For the given data, calculate the total head movement



8. What if you wanted to run a program that needs more memory than you have? Consider a machine with 64 MB physical memory and a 32-bit virtual address space. If the page size is 4KB, what is the approximate size of the page table?

8. If any computer program needs more RAM than there is a variable it starts to use so called "page file" a substitute of RAM that is actually a file on the harddisk. As the HDD is usually much ~~better~~ slower than RAM, the program performance get worse, but it still works.

No. of entries in pagetable
= (virtual address space size) / (page size)

using above formula, we can see that there will be

$(2^{32-12}) = 2^{20}$ entries in pagetable.

No. of bits required to address 64MB physical memory = 16

so there will be $2^{26-12} = 2^{14}$ page frames in. Therefore each page contains 14 bits address of page frame and 1 bit for valid - invalid bit

So, memory is a byte addressible so we take each page table entry as 16 bits i.e 2 byte long size of page table

(Total no. of page table entries) *
(Size of pagetable entry)

$2^{20} \times 2 = 2\ MB$

9. Consider a simple system running a single process. The size of physical frames and logical pages is 16 bytes. The RAM can hold 3 physical frames. The virtual addresses of the process are 6 bits in size. The program generates the following 20 virtual address references as it runs on the CPU: 0, 1, 20, 2, 20, 21, 32, 31, 0, 60, 0, 0, 16, 1, 17, 18, 32, 31, 0, 61. (Note: the 6-bit addresses are shown in decimal here.) Assume that the physical frames in RAM are initially empty and do not map to any logical page.
   (a) Translate the virtual addresses above to logical page numbers referenced by the process. That is, write down the reference string of 20 page numbers corresponding to the virtual address accesses above. Assume pages are numbered starting from 0, 1, ...
   (b) Calculate the number of page faults genrated by the accesses above, assuming a FIFO page replacement algorithm. You must also correctly point out which page accesses in the reference string shown by you in part (a) are responsible for the page faults.
   (c) Repeat (b) above for the LRU page replacement algorithm.
   (d) What would be the lowest number of page faults achievable in this example, assuming an optimal page replacement algorithm were to be used? Repeat (b) above for the optimal algorithm.

9.(a) For 6 bit virtual addresses and 4 bit page offsets (page size 16 bytes) the most significant 2 bits of a virtual address will represent page number

So Reference string is 0, 0, 1, 0, 1, 2, 1, 0, 3 (repeated again)

(b) page faults with FIFO = 8. Page faults on 0, 1, 2, 3 (replaced 0), 0 (replaced 1) 1 (replaced 2), 2 (replaced 3), 3

(c) page faults with LRU = 6. page faults on 0, 1, 2, 3 (replaced 2), 2 (replaced 3), 3.

(d) The optimum algorithm will replace page least libely to be used in future, and would like LRU above

10. Consider a system with only virtual addresses, but no concept of virtual memory or demand paging. Define total memory access time as the time to access code/data from an address in physical memory, including the time to resolve the address (via the TLB or page tables) and the actual physical memory access itself. When a virtual address is resolved by the TLB, experiments on a machine have empirically observed the total memory access time to be (an approximately constant value of) t_h. Similarly, when the virtual address is not in the TLB, the total memory access time is observed to be tm. If the average total memory access time of the system (averaged across all memory accesses, including TLB hits as well as misses) is observed to be t_x, calculate what fraction of memory addresses are resolved by the TLB. In other words, derive an expression for the TLB hit rate in terms of t_h, t_m, and t_x. You may assume $t_m > t_h$.

10. we have $t_x = h \times t_h + (1-h) \times t_m$

$$\text{to } t_h = \frac{t_m - t_x}{t_m . t_h}$$

11. Consider a system with a 6 bit virtual address space, and 16 byte pages/frames. The mapping from virtual page numbers to physical frame numbers of a process is (0,8), (1,3), (2,11), and 4 (3,1). Translate the following virtual addresses to physical addresses. Note that all addresses are in decimal. You may write your answer in decimal or binary.

11. $90 = 010100 = 110100 = 52$

$40 = 101000 = 10111000 = 184$

12. Consider a system with several running processes. The system is running a modern OS that uses virtual addresses and demand paging. It has been empirically observed that the memory access times in the system under various conditions are: t1 when the logical memory address is found in TLB cache, t2 when the address is not in TLB but does not cause a page fault, and t3 when the address results in a page fault. This memory access time includes all overheads like page fault servicing and logical-to-physical address translation. It has been observed that, on an average, 10% of the logical address accesses result in a page fault. Further, of the remaining virtual address accesses, two-thirds of them can be translated using the TLB cache, while one-third require walking the page tables. Using the information provided above, calculate the average expected memory access time in the system in terms of t1, t2, and t3.

12. $0.6 \times t_1 + 0.3 \times t_2 + 0.1 \times t_3$

13. Consider an operating system that uses 48-bit virtual addresses and 16KB pages. The system uses a hierarchical page table design to store all the page table entries of a process, and each page table entry is 4 bytes in size. What is the total number of pages that are required to store the page table entries of a process, across all levels of the hierarchical page table?

13. page size = 2^{14} bytes

so no.of pagetable entries = $\frac{2^{48}}{2^{14}} = 2^{34}$

Each page can store $\frac{16\text{KB}}{4} = 2^{12}$ pagetable entries

so, no of innermost pages = $\frac{2^{34}}{2^{12}} = 2^{22}$

Now pointers to all innermost pages must be stored in next level page table so, next level of page table has $2^{22}/2^{12} = 2^{10}$ pages

Finally a single page can store all 2^{10} page table entries. so, outermost level has one page.

so, total no.of pages the stole page table entries is $2^{22} + 2^{10} + 1$

14. Consider a 64-bit system running an OS that uses hierarchical page tables to manage virtual memory. Assume that logical and physical pages are of size 4KB and each page table entry is 4 bytes in size.

14. Ceil ((64 - 12) / (12 - 2)) = 6

2, 10, 10, 10, 10, 10 (starting from most significant to least)

Innermost level has 2^{42} PTE's, which fit in 2^{42} pages. Next level has 2^{42} PTE's. which requires 2^{32} pages & so on

Total pages = $2^{42} + 2^{32} + 2^{22} + 2^{12} + 2^{2} + 1$

15. Consider a system with 16 bit virtual addresses, 256 byte pages, and 4 byte page table entries. The OS builds a multi-level page table for each process. Calculate the maximum number of pages required to store all levels of the page table of a process in this system.

15. No. of PTE's per process = $\frac{2^{16}}{2^8} = 2^8$

No. of PTE's per page = $\frac{2^8}{2^2} = 2^6$

No of inner page table pages = $\frac{2^8}{2^6} = 2^2$

which requires one outer page directory

so total pages = 4 + 1 = 5

16. Consider a virtual memory system with physical memory of 8GB, a page size of 8KB and 46 bit virtual address. Assume every page table exactly fits into a single page. If page table entry size is 4B then how many levels of page tables would be required.

16. page size = 8KB = 2^{13} B

virtual address space size = 2^{46} B

PTE = 4B = 2^{2} B

no of pages or no of entries in pagetable

= (virtual address space size) / (page size)

= 2^{46} B / 2^{13} B

= 2^{33} B

Size of page table

= (no. of entries in page table) * (size of PTE)

= $2^{33} * 2^2 B = 2^{35} B$

To create one more level, size of page table > page size

no. of page table in last level,

= $2^{35} / 2^{13} = 2^{22}$

Base address of these tables are stored in page table (2^{nd} last level)

Size of page table [second last level]

= $2^{22} * 2^2 = 2^{24} B$

To create one more level,

size of page table [2^{nd} last level] > page size

No of page tables in second last level =

$2^{24} / 2^{13} = 2^{11}$

Base address of these tables are stored in page tab (3^{rd} last level)

Size of page table [3^{rd} last level]

= $2^{11} * 2^2 B = 2^{13} B$ = page size

∴ 3 levels are required

190031187 | NERELLA VENKATA RADHAKRISHNA

17. A certain computer system has the segmented paging architecture for virtual memory. The memory is byte addressable. Both virtual and physical address spaces contain 216 bytes each. The virtual address space is divided into 8 non-overlapping equal size segments. The memory management unit (MMU) has a hardware segment table, each entry of which contains the physical address of the page table for the segment. Page tables are stored in the main memory and consists of 2 byte page table entries. Assume that each page table entry contains (besides other information) 1 valid bit, 3 bits for page protection and 1 dirty bit. How many bits are available in page table entry for storing the aging information for the page? Assume that page size is 512 bytes.

17. Given

virtual address space = process size = 2^{16} B

physical address space = Main memory = 2^{16} B

process is divided into 8 equal size segment

pagetable size entry = 2 bytes = 16 bits

pagetable entry besides other information contains 1 valid bit, 3 protection bits, 1 dirty bit,

page size = 512 bytes

No of frames in main memory

No of frames in main memory =

(size of main memory) / (page size)

= 2^{16} bytes / 512 bytes

= 2^{7} bytes

Thus no of bits required for frame identification in pagetable entry = 7 bits.

No of bits available for storing aging info

= no of bits in pagetable entry − (no of bits req for frame identification) + 1 valid + 3 protection + 1 dirty bits.

= 16 bits − (7+1+3+1) bits

= 4 bits.

18. A certain computer system has the segmented paging architecture for virtual memory. The memory is byte addressable. Both virtual and physical address spaces contain 216 bytes each. The virtual address space is divided into 8 non-overlapping equal size segments. The memory management unit (MMU) has a hardware segment table, each entry of which contains the physical address of the page table for the segment. Page tables are stored in the main memory and consists of 2 byte page table entries. What is the minimum page size in bytes so that the page table for a segment requires at most one page to store it? give the division of virtual address.

18. Given

Virtual address space = 2^{16} bytes

physical address space = 2^{16} bytes

page table entry size = 2 bytes

let page size = n bytes

Since page table has to be stored into single page, we must have size of page tables <= page size

Size of each segment = process size / no of segments

$\therefore 2^{16}$ bytes / 8 = 2^{13} bytes

= 8 KB

no of pages of each segment = $\frac{\text{size of segment}}{\text{page size}}$

= 8B/n = (8k/n) pages

size of each page table = no of entries * page table entry size

= no of pages that segment is divided * 2 bytes

= 8 (k/n) * 2 bytes

= (16k/n) bytes

page size = 16k/n bytes <= n bytes

= 16k/n <= n

$n^2 >= 16k$ =>

$n^2 >= 2^{14}$

$n >= 2^7$

MM page size = 2^7 bytes

= 128 bytes

Division of virtual address:

no of segments the process is divided = 8

$= 2^3$

no. of bits = 3

no. of pages a segment is divided =

segment size / page size

= 8 KB / 128 bytes

= 2^{13} bytes / 2^7 bytes

= 2^6 pages

no. of bits = 6

no. of bits req for page offset

page size = 128 bytes = 2^7 bytes

no. of bits = 7

Thus virtual address is divided as