Question 1.

Protection of processes' memory is a method to manage the access rights of memory in the computer. Most modern operating system and processors use the memory protection. The main goal of memory protection is to restrict a process from accessing memory that has not been allocated to it.

Memory protection prevents a defect or bug within a process from attacking other processes, or the operating system itself. An effort to access unowned memory ends in a hardware fault called a segmentation fault. It usually producing abnormal termination of the process.

Question 2.

(i) Virtual address 130 belongs to page 1.

     So, in the table ****Process ID 1**** and ****Page number 1**** corresponds to ****Frame number 1.****

     Hence this virtual address mapped to ****physical address 130****.

(ii) Virtual address 17 belongs to page 0.

     So, in the table ****Process ID 2**** and ****Page number 0**** don't have any entry.

     Hence this virtual address ****"does not map"****

(iii) physical address 50 belong to frame 0

       In the table Frame 0 is mapped to ****Process ID 1**** and ****Page number 2****

       Hence the ****virtual address 250**** of ****process ID 1**** is mapped to ****physical address 50****.

Question 3.

1.

These are the virtual address ranges that will result in a page fault.

Page 0 : 0 --1023

Page 1 : 1024 --2047

Page 2 : 2048 --3071

Page 4 : 4096 - 5119

Page 5 : 5120 - 6143

2.

1. After applying LRU based page replacement, page 4 will replace the page 7.

Updated Page Table

|  |  |  |
| --- | --- | --- |
| Virtual page # | Physical page # | Valid Flag |
| 0 |  | No |
| 1 |  | No |
| 2 | 2 | Yes |
| 3 | 3 | Yes |
| 4 | 1 | Yes |
| 5 |  | No |
| 6 | 0 | Yes |
| 7 |  | No |

The virtual address (4500)10 can be written as (1000110010100)2. Now this address is divided into the page and offset fields [100][0110010100]2

P O

where P denotes page number and O denotes Page offset. Now page field 100 of the virtual address is replaced by the frame number 01, since page 4 maps to frame 1 and it is converted to the physical address [01][0110010100]2 = (1428)10

1. For 8000 virtual adddress, there will be again page fault for page 7. After applying LRU, the page 7 will replace the page 4.

Update Page Table:

|  |  |  |
| --- | --- | --- |
| Virtual page # | Physical page # | Valid Flag |
| 0 |  | No |
| 1 |  | No |
| 2 | 2 | Yes |
| 3 | 3 | Yes |
| 4 |  | No |
| 5 |  | No |
| 6 | 0 | Yes |
| 7 | 1 | Yes |

Suppose the CPU now generates virtual address (8000)10 = (1000110010100)2. Now this address is divided into the page and offset fields [111][0110010100]

P O

Now page field 111 of the virtual address is replaced by the frame number 01, since page 7 maps to frame 1 and it is converted to the physical address [01][ 0110010100]2 = (1428)10

(iii) For virtual address, (3000)10. There will be no page fault. The virtual address (3000)10 can be represented as (0101110111000)2. Now this address is divided into the page and offset fields [010][1110111000]

P O

Now page field 010 of the virtual address is replaced by the frame number 02, since page 2 maps to frame 2 and it is converted to the physical address (1363)10 = [10][1110111000]2

(iv). For virtual address 1100, there will be again page fault for page 1. Page 1 will replace the Page 6.

Updated Page Table

|  |  |  |
| --- | --- | --- |
| Virtual page # | Physical page # | Valid Flag |
| 0 |  | No |
| 1 | 0 | Yes |
| 2 | 2 | Yes |
| 3 | 3 | Yes |
| 4 |  | No |
| 5 |  | No |
| 6 |  | No |
| 7 | 1 | Yes |

The virtual address 110010 can also be written as (0010001001100)2. Now this address is divided into the page and offset fields [001][0001001100].

P O

Now page field 001 of the virtual address is replaced by the frame number 00, since page 1 maps to frame 0 and it is converted to the physical address (76)10 = [00][0001001100]2

Question 4.

Address space = 32 bits

Page size = 4KB = 212 bytes. Total 12 bytes required to represent each page.

Total entries in page table = 232-12 = 220

****a)**** Since physical memory size is not mentioned, so we assume there is enough pages in memory to load the complete process

Max pages per process = ceil(Max memory available to a process / Size of a page)

we use ceil because we can't allocate partial page to process.

Page size = 4KB

****b)**** 100 MB process require (100\*220 )/ (212) pages i.e. 100\*28 pages.

page table size= Number of entries \* size of entry

page table size = 100\*28 \* 4 (4 bytes given in question)

page table size=100\*210 bytes=100KB

****c)**** P1 size=10KB, pages require by P1=ceil(10/4) = 3

page memory required by P1=3\*4KB =12KB

internal fragmentation = page memory-process size = (12-10)KB = 2 KB

P2 size=15KB, pages require by P2=ceil(15/4) = 4

page memory required by P2=4\*4KB =16KB

internal fragmentation = page memory-process size = (16-15)KB = 1KB

Paging doesn't allow external fragmentation since in paging, memory is allocated in pages where page allocated need not to continuous so a process can reside in memory in different pages at different locations. This result in no external fragmentation since no continuous blocks are needed. However internal fragmentation can still be there as in above example.

****d)**** Access Time on hit = TLB time + memory access time = a+b

Access time on miss = TLB time + memory access time + page fault process time = a+b+c

Average Access time = hit\_ratio\*(Access time on hit)+(1-hit\_ratio)\*(Access time on miss)

Average access time = 0.99(a+b)+0.01\*(a+b+c)