**OPERATING SYSTEM**

* **Group 14:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Họ và Tên** | **Mã sinh viên** | **Lớp** | **Phân công** |
| 1 | Phan Văn Bằng | 20IT490 | 20IT1 | C3.42 |
| 2 | Nguyễn Thị Như Quỳnh | 20IT016 | 20IT1 | C3.41 |
| 3 | Dương Tuấn Đạt | 20IT488 | 20IT1 | C3.40 |
| 4 | Nguyễn Minh Tân | 20IT929 | 20IT9 | Không phản hồi |

**CHAPTER 3. DEADLOCKS**

**40) Which strategies are designed for speeding up Paging? (Choose 2)**

**a) Page table is loaded into main memory.**

**b) Using TLB.**

**c) Page table is loaded into disk.**

**d) Page table is divided into several levels.**

Answer: a, c

Explain:

a. Because age tables are said to be stored in the kernel-owned physical memory. However page tables can get awfully big since each process have their own page tables (unless the OS uses inverted paging scheme). For even a 32 bit address space with a typical 4KB page size, we shall require a 20 Bit virtual page number and a 12 bit offset. A 20 bit VPN(Virtual Page Number) implies that there would be 2^20 translations. Even if each translation i.e the Page Table entry requires 4 Bytes of memory, it amounts to 4x(2^20)= 4MB of memory, all just of address translations, which is awful. Hence modern OSes place such large page tables in virtual kernel memory which is the Hard Disk, and swaps them to the physical memory whenever required. Thus page table is virtualized the same way each page is virtualized.

c. The Translation Lookaside Buffer is a memory cache;

Cache used to reduce the time to map virtual to physical addresses;

• Part of the chip’s memory-management unit (MMU);

• Stores the recent translations of virtual memory to physical memory;

Consists of a small number of entries: rarely more than 256

• Each entry contains information about one page, including:

• virtual page number;

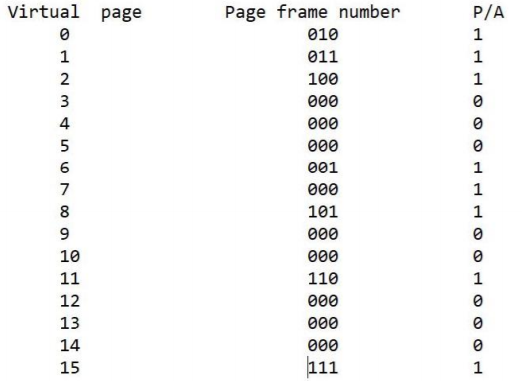
• modified bit;

• protection bits (read/write/execute permissions);

• physical page frame;

• valid bit: indicates whether the entry is valid (i.e., in use) or not.

**41) Suppose that a computer with 16 bits virtual address use a single level page table and 15 bits physical memory address is given by the following page table. Which is the physical address that the MMU will map with the virtual address 1000 0000 0010 0100 onto?**

****

**a) 101 0000 0010 0100**

**b) 011 0000 0100 0100**

**c) 110 0000 0100 0000**

**d) DISK**

Answer: d

Explain:

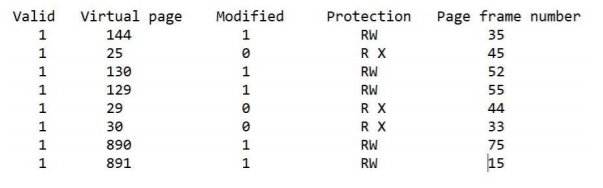
It is given that virtual address is 16 bit long. Hence, there are 2^16 addresses in the virtual address space. Page Size is given to be 4KB ( there are 4K (4 \* (2 ^ 10) )addresses in a page), so the number of pages will be ( 2^16 ) / ( 2 ^ 12 ) = 2 ^ 4.

4 VNP and 12 offset. Page size (in the virtual address space ) is always same as the frame size in the main memory. Hence the last 12 bits will remain same in the physical address as that of the virtual address.

VNP= 0x4

=> it's on disk

**42) Consider that a TLB (Translation Lookaside Buffer) is given by the following table. What will be happened if the instruction is trying to write on the virtual page number 130th?**

****

**a) A protection fault is generated**

**b) A page fault occurs**

**c) The page frame 52 will be deleted**

**d) The page frame 52 will be modified**

Answer: b

Explain:

Because Write fault: A write was attempted but no entry matched.

Occurs when the page accessed by a running program is not present in physical memory. It means the page is present in the secondary memory but not yet loaded into a frame of physical memory.

The interesting case is what happens when the virtual page number is not in the TLB. The MMU detects the miss and does an ordinary page table lookup. It then evicts one of the entries from the TLB and replaces it with the page table result in a TLB hit rather than a miss. When an entry is purged from the TLB, the modified bit is copied back into the page table entry in memory. The other values are already there, except the reference bit. When the TLB is loaded from the page table, all the fields are taken from memory