**Computer Science & Engineering Department**
**I. I. T. Kharagpur**

**Operating System: CS33007**
*3rd Year CSE: 5th Semester* (*Autumn 2005 - 2006*)
*Lecture XII* (**Memory Management**)

Goutam Biswas *Date:* 11th September, 2006

Memory Management: Paging

1. The main memory may be divided into fixed size *page frames*.
2. A program along with data is divided into same size *pages*.
3. The size of a *page frame* depends on several factors e.g. the size of the disk block, the time taken to transfer one block of data from the disk to the main memory etc.
4. Different pages of a process are loaded in different page frames of the main memory.

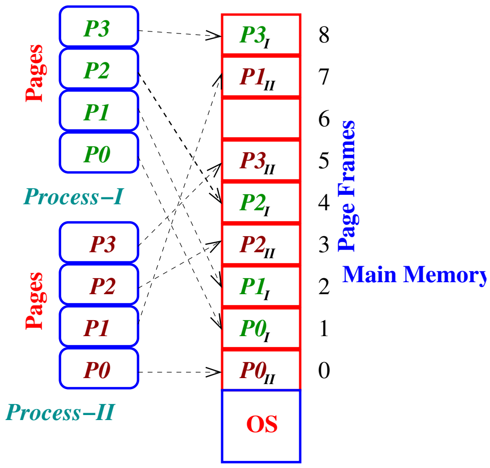


Figure 1: **Paging**

5. Page and page frame - an example
   - Size of the main memory is 16 MBytes.
   - The size of code + data is 122KBytes (from the load module).
   - Size of a *Page (page frame)* is 4 KBytes.
   - Total number of page frames are $\frac{16K}{4} = 4K$.
   - The code will occupy $\frac{122}{4} = 31$ pages.
6. Internal fragmentation - In the previous example the last page frames (30th frame) is not full. Some memory space within the last frame is wasted. This is called an internal fragmentation.

7. Page table - translation of logical address to physical address.

   - For each process a table, called the page table, is used to translate the *virtual* or *logical* address to the *main memory address*.
   - A *logical address* has two parts -

| Page Number | Offset within Page |
|---|---|

   - The *page number* is used as an *index* to the page table.
   - The *page frame number* of the main memory where the page is loaded is obtained from the *page table*.

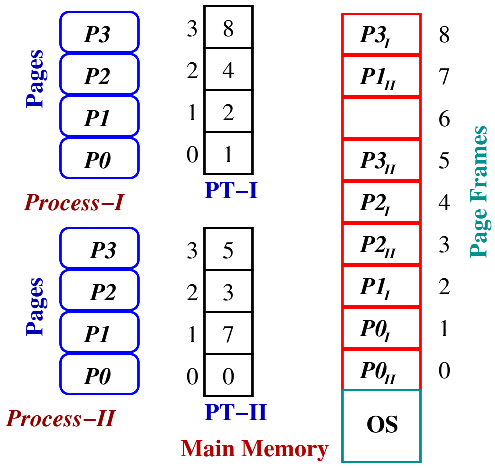


Figure 2: **Page Table**

   - There are questions related to the page table -
     – What is the size of the page table?
     – Where does the OS stores the page table?

8. Size of the page table, a 32-bit Pentium example -

   - The logical address is 32-bits.
   - There are $4M = 2^{20}$ entries in the page table for each process.
   - Assuming 4-Bytes per table entry, each process needs **4 MBytes** space only for the page table. This not an acceptable solution.

9. Two level page table -

   - A page table is also divided into *pages*.
   - There is a table of the page-table pages called the *page directory*.
   - For each process there is a *page directory*.
   - The 32-bit logical address is divided in i *three parts*

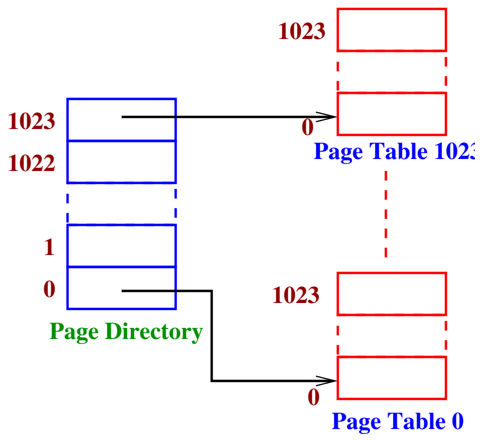


Figure 3: **2-Level Page Table**

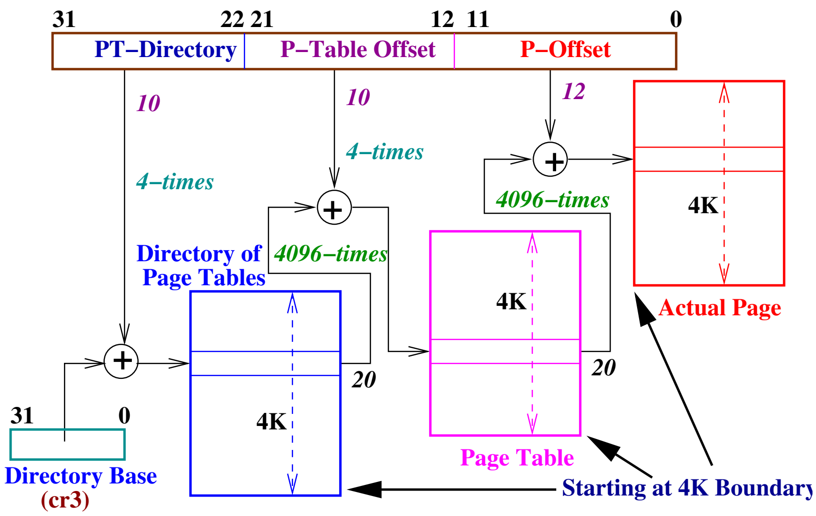


Figure 4: **2-Level Translation**

| Directory Index (31-22) | Page Table Index (21-12) | Offset within a Page (11-0) |
|---|---|---|

- Address translation -
  - The *page directory base register* (cr3 in Pentium) is loaded by the OS when a process is scheduled.
  - The address bit *31-22* is multiplied by 4 (4-Bytes per entry) and added to the content of *cr3* to get the physical address of the directory entry.
  - The directory entry provides the *base address* of the corresponding page (if it is present).
  - The *base address* is specified by *20-bits*. The size of the *directory*, the size of a *page table* and the size of a *page* are each *4 KBytes*. Each one is loaded at the *4 KByte* boundary.
  - The base address of the page table is multiplied by *4K* and added to the address bits *21-12* (after multiplying by 4).
  - This gives the address of the *page table entry* which holds the *base address* of the page (if it is present).
  - The *20-bit* base address of the page is obtained from the page table.
  - It is multiplied by *4K* and added to the *12-bit Offset* (logical address bits 11-0) to get the *physical address* of the location specified by the *logical* or *virtual address* (Intel call it *linear address*).

10. Address translation in 2-level paging - an example. Assume that the OS has assigned a *virtual space* of size **0x1000 0000 - 0x1001 FFFF** to a process.

    - The size of the *virtual space* is 128KBytes.

    | Directory | Table | Offset |
    |---|---|---|
    | 0001 0000 00 | 00 000X XXXX | XXXX XXXX XXXX |

    where $X \in \{0, 1\}$

    - There is only one *valid entry* in the *page directory* (out of 1024 entries). Its *index* is i $0x00\ 0100\ 0000 = 64_D$.
    - All other entries of the page directory are nil.
    - There is only one page table with *valid entries* ranging from the index $\mathbf{00\ 0000\ 0000 = 0}_D$ to $\mathbf{00\ 0001\ 1111 = 31}_D$ i.e. there are *32 pages*, each of size *4 KBytes*.

11. Extra memory for two level paging -

    - One *page directory* of size *4 KBytes* with only one valid entry.
    - One *page table* of size *4 KBytes* with 32 valid entries.
    - *8 KBytes of extra memory* is required for paging.

12. Other data in the *page directory* and the *page table*.

- **Valid/Present Bit**: If *set (1)*, it indicates that the page is valid and present in the main memory. If it is *zero (0)*, the page is not valid or in case of a virtual memory system it is not present.

  For an invalid page the *logical address* is saved in some *control register* (cr2 in Pentium) and a *memory exception* (*page fault* in case of virtual memory) (`int 14` in Pentium) is generated. In this case the other bits may be used by the OS.
- The *page is present* -
  - *20-bits* are used as the *base address*.
  - *Dirty Bit* - Indicates whether the page has been modified. Its copy in the disk may be *stale*.
  - *Permission bits* - Indicates whether the page is *read-only* or *writable*.
  - *Access privilege* - Indicates the *privilege level* required to access the page etc.

13. Protection and sharing using page table -

- Protecting a process or the OS from another process is easy. The instruction to modify the *page directory base register* (cr3) is to be a *privileged instruction* and is not available to a user process.

```
/***************************************
 * The page table base register cr3 is    *
 * not available to a user process        *
 * ***************************************/
#include <stdio.h>

int main() {
asm (
"movl $1, %%eax \n\t"
"movl %%eax, %%cr3 \n\t"
:
:
:"%eax"
    ) ;
return 0;
}
```

- It is also easy to *share* pages under the *paged memory management*.
- If two processes want to share a page, the *base address* of the shared page-frame in the main memory is entered in page tables of both the processes.
- It is also easy to have different set of permissions on the same page for two different processes.

14. problems of paged memory management -

- Instruction fetch or data access through a 2-level page-table system will first look-up the *page directory*, then the *page table*, and finally will fetch the actual instruction or data from the main memory.

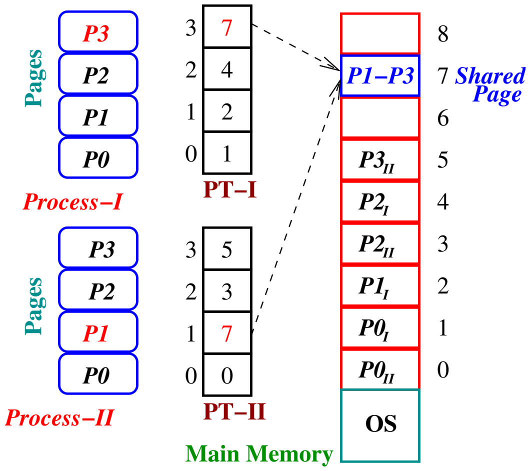


Figure 5: **Shared Page**

- Every access to a logical address will give rise to three (3) access to the physical memory. This is not acceptable as the access to physical memory is slow compared to the processor speed.

15. A solution from architecture - translation look-aside buffer (TLB) cache -

- The *memory management unit (MMU)* has a *fully associative cache memory* know as i the *translation look-aside buffer (TLB)* cache for fast translation of virtual to physical address.

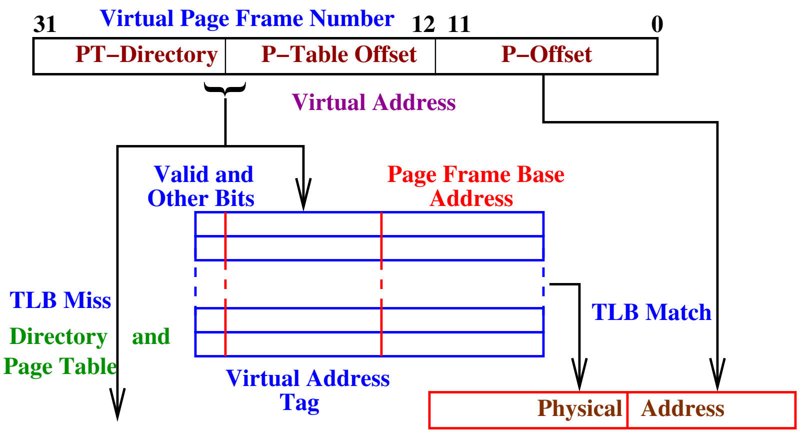


Figure 6: **TLB**

- When a process gets scheduled, the *page directory base register* is loaded with the appropriate *base address*. All entries of the *TLB cache*[1] may be *invalidated* (there are other possibilities as well).

[1]Not to be confused with the cache memory of the memory hierarchy.

- If some of the old entries of the TLB have their *dirty bits* set, the corresponding page table entries are to be properly updated.
- When the scheduled process starts its execution, there is no valid entry in the TLB. The first logical address will be translated to the physical address with the help of the page table system.
- Once the translation is known, the MMU loads an entry of the TLB with the following information.
  - The virtual page frame number (31-12) is loaded in the *tag* field.
  - The corresponding *page frame base address* is loaded in the corresponding cache line.
  - The valid and other control bits are set accordingly.
- Any subsequent reference to the same virtual page will be translated using the TLB.
- The virtual page frame number from the logical address is compared in parallel with the *tags* corresponding to all the *valid entries* of the TLB. This is called an associative search.
- If there is a match, the page-frame base address is obtained immediately, and the memory is accessed.

16. The system of TLB works due to the fact that the memory references are clustered. And the cluster changes slowly.

    There is a *working set* of pages (instruction as well as data), where most of the references are made for a time period much larger than the time of execution of an instruction.

    There is spatial and temporal locality in the stream of instruction execution.

17. TLB Miss -

    - Typically there are 16 to 128 entries in a TLB (that is good enough to cover 64 KBytes to 512 KBytes of memory)[2].
    - If there is a TLB miss, and there is a *free slot* in the TLB, the first translation is through the page table. But the new translation entry is loaded in the available slot of the TLB.
    - But if all entries of the TLB are *valid*, then it is necessary to replace one of those to accommodate the translation of the referenced page.
    - The question is which entry to replace and what is to be done if the *dirty bit* is set.
    - Simple replacement policy is implemented on hardware. If the dirty bit is set, the bit is to be written in the corresponding page table entry while replacing.

18. Printing the content of page directory base register in Pentium - **cr3**

19. Three level paging - DEC Alpha example -

    - 64-bit architecture - The address space is *64-bit*. But the most significant 21-bits are always set to zero (0). We are left with 43-bits.

[2] Systems with larger TLB may not be fully associative.

- The 2-level paging scheme does not work for such a big address space.
- An example -
  - Let the page size be 8 KBytes. The least significant 13-bits will give the offset within the page.
  - Remaining 30-bits may be divided in two groups, each of size 15-bits.
  - Each table will have 32K entries. Even if the size of each entry be 32-bits, the size of each table will be 128KBytes. There will be at least two such tables for each process i.e. 256KB of table space per process. This is too big.
- Three level paging -
  - The page size 8 KBytes. Bit 12-0 is of the logical address is the page offset.
  - The base address of directory$_1$, the top-level directory, whose base address is available from an MMU register. It is indexed by higher order 10 bits (bit 42-33) of the logical address.
  - The base address of directory$_2$, the middle directory, is obtained from the directory$_1$ and is indexed by the next 10 bits (bit 32-23) of the logical address.
  - The base address of a page table is obtained from the directory$_2$ and is indexed by the next 10 bits (bit 22-13) of the logical address.
  - Size of each table entry may be 8 Bytes.

20. Inverted Page table - a solution to a very large address space.

    (a) An alternate to a system of page table per process, is a *global page table* where each entry corresponds to a *page frame* of the main memory.

    (b) The index of the table, after multiplying and adding two constants, gives the base address of the page frame.

    (c) Each entry of the page table contains, other than the protection information, the process ID and the logical page number.

    (d) It is clear that though this page table reduces the total memory requirement for the table, the translation through the table is costly as it involves search through the table.

    (e) But most of the time the translation takes place through the TLB cache and the table search is not necessary. The searching can be reduced by using hashed inverted table.

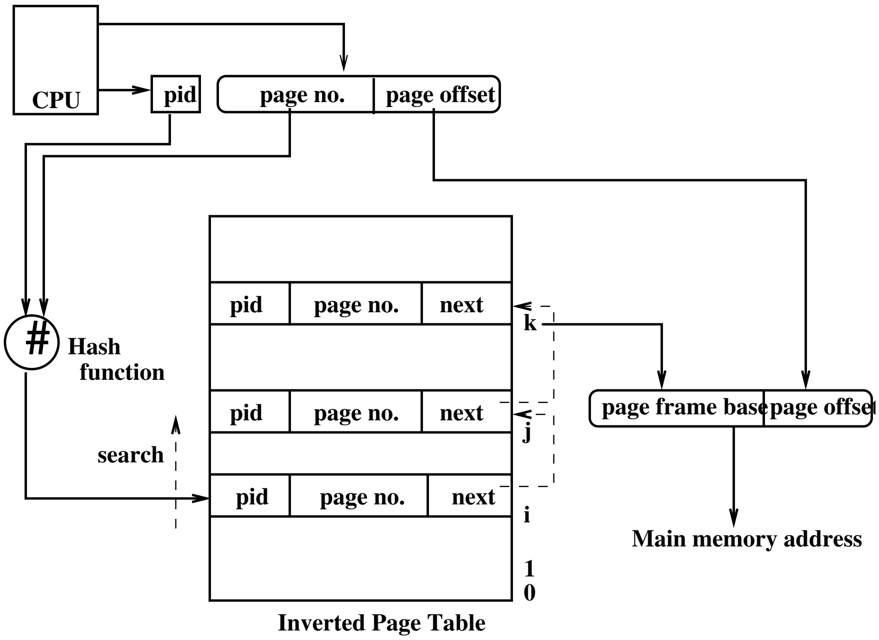


Figure 7: **Inverted Page Table**