Memory Management

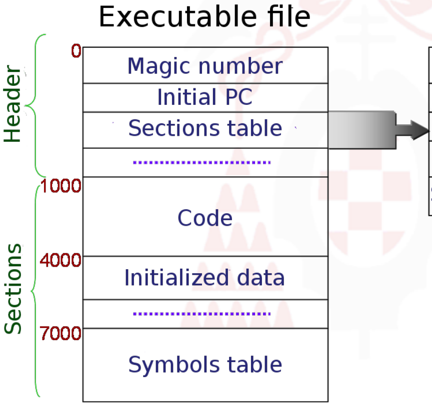
E. Campo M. Knoblauch Ó. López J. Clemente

Departamento de Automática
Universidad de Alcalá

/gso>

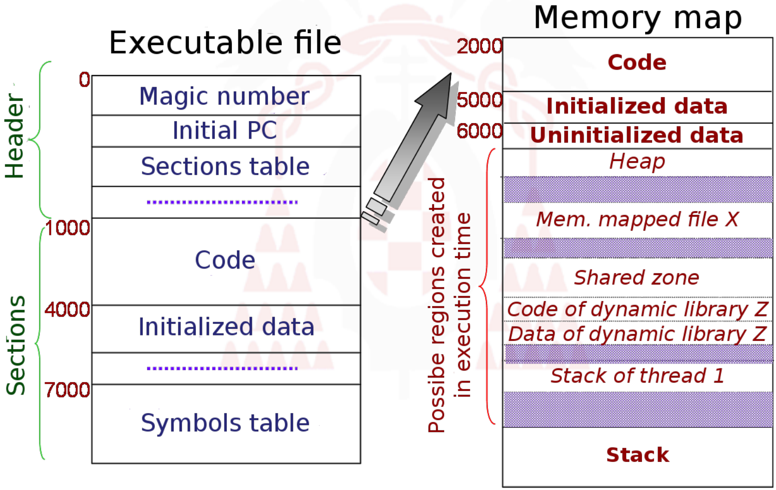
Index

Format of an executable file



	Offset	Size
Code	1000	3000
Init. data	4000	1000
Uninit. data	-	100
....	....	....
Symbols tab.	7000	1000

Memory map of a process



Program and process

```
...
char * progname, int cont = 1;

void Func (int x)
{
   int result = 0;
   char character = 'a';

   if (cont)
      cont = result ++;
   ...
   return;
}

main (int argc, char * argv[])
{
   int i;
   char * progname;

   Func (cont);
   progname = (char *) malloc (1 + strlen(argv[0]));
   ...
   free (progname);
   ...
   exit (0);
}
```

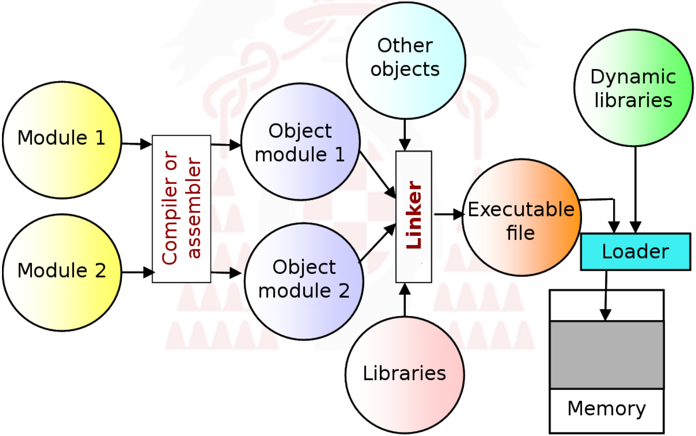
Addressing abstractions

Address space

Set of referentiable addresses

- Virtual address space $\Rightarrow$ independent for every process
- Physical address space $\Rightarrow$ shared amongst all processes
- Processes only reference virtual addresses
- There must be a translation from virtual to physical address, transparent to the process

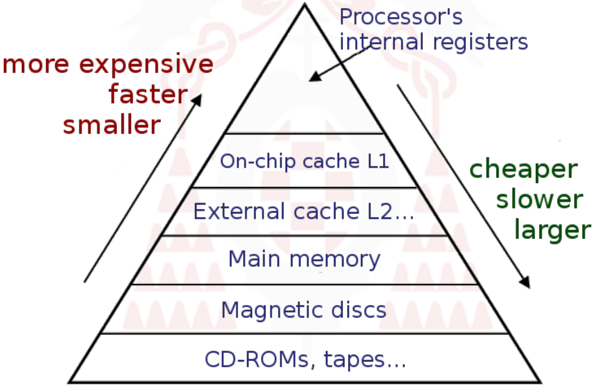
Addressing abstractions



Memory hierarchy in a computer

- The organization of memory into a hierarchy is an attempt to enhace computers' performance
- Based on: programs' locality + technological advances in memory designs
- Fast memories: small capacity, expensive
- Slow memories: large capacity, cheap

Scheme of memory hierarchy



Principle of locality

- During time intervals, processes tend to concentrate references in a subset of their address space

Donald Knuth [1971]:

Programs typically have a very jagged profile, with a few sharp peaks. [..] We also found that less than 4 per cent of a program generally accounts for more than half of its running time.

- It's an empirical property
- There are two types of locality:
  - Spatial locality
  - Temporal locality

Spatial locality

- Once a memory position has been referenced, odds are that near positions will be referenced either.
- Supporting this remark:
  - Sequential execution of code
  - Programmers' tendency to put related variables together
  - Access to data structures like stacks or arrays

Temporal locality

- Once a memory position has been referenced at instant t, odds are that it will be referenced again at instant $t + \Delta t$
- Supporting this remark:
  - Loops
  - Subroutines
  - Stacks

Fragmentation

Fragmentation

Waste (inefficient use) of the available free memory due to the management mechanism employed

- There are two types: internal and external
- Internal fragmentation
  - Caused by the difference of size between the memory partition and the object allocated inside it
- External fragmentation
  - Caused by the inability to use memory between partitions

Relocation

Relocation

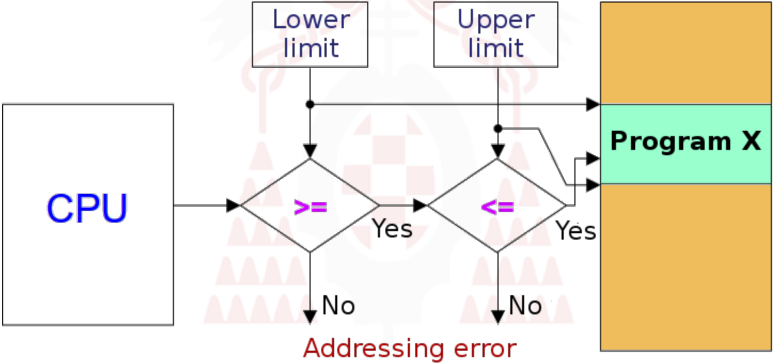
Assigment of addresses to the different parts of a program (code, data, stack...)

- Depending on **when** is fixed the final location, relocation will happen in the compilation stage, in the loading stage, or during the execution stage
- Static relocation
  - Carried out before or while loading the program
  - Once started, programs cannot be moved
- Dynamic relocation
  - The translation from virtual address to real (physical) address is carried out in execution time
  - It requires additional hardware (MMU)
  - Programs can be moved in execution time

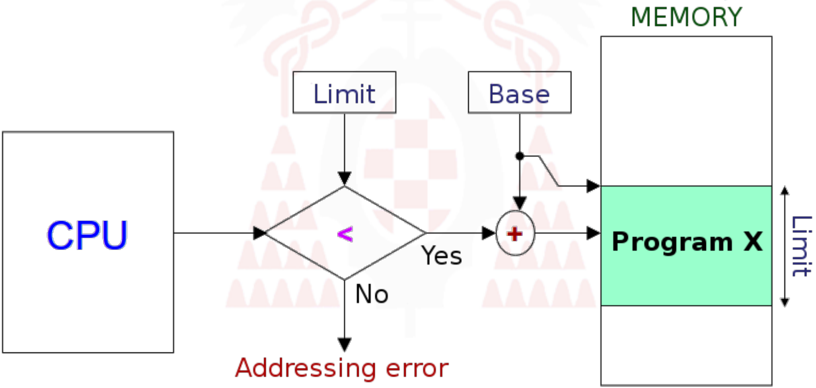
Protection and sharing

- Need to restrict memory access:
  - Operating system
  - User processes
- Protection methods
  - Limit registers
  - Base and limit registers
  - Memory protection bits
  - Access rights in translation tables
  - Where are they stored?
- How to share memory between processes?

Limit registers



Base and limit registers



Historic evolution

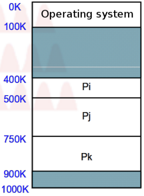
- Bare machine
  - The system provides no service
- Monolithic monitor
  - In addition to the operating system, there is just one process
- Contiguous partitioned memory
  - Multiprogramming with a fixed number of tasks (MFT)
    - Fixed-size partitions
    - Created while booting the system
  - Multiprogramming with a variable number of tasks (MVT)
    - Variable-size partitions
    - Created when processes need them
- Non-contiguous partitioned memory

Non-contiguous partitioned memory

- The contents of a process can be distributed in separated memory partitions
- The memory is organized in partitions:
  - Variable size ⇒ segments
  - Fixed size ⇒ frames

Partitions description table

- Independent for every process
- Built when the process is loaded

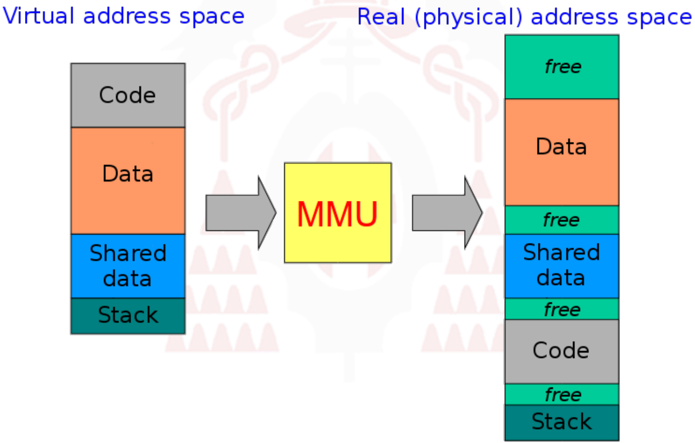


Partition number	Base	Size	State
0	0K	100K	ASSIGNED
1	100K	300K	FREE
2	400K	100K	ASSIGNED
3	500K	250K	ASSIGNED
4	750K	150K	ASSIGNED
5	900K	100K	FREE

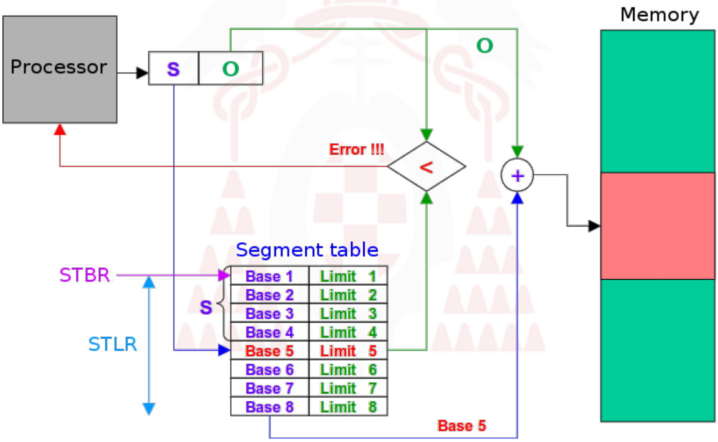
Segmentation

- The physical memory is initially organised as a unique empty block, where variable-size partitions (segments) are created as required
- The virtual address space is organized in segments
- Includes protection mechanism and allows sharing
- Virtual addresses are composed of two elements: segment number and offset
- The partition table is called Segment Table (ST)
- If the ST is too big, it has to be stored in main memory, pointed by a register (STBPR) $\Rightarrow$ every access requires two references to memory

Segmentation: logical scheme



Segmentation: physical scheme



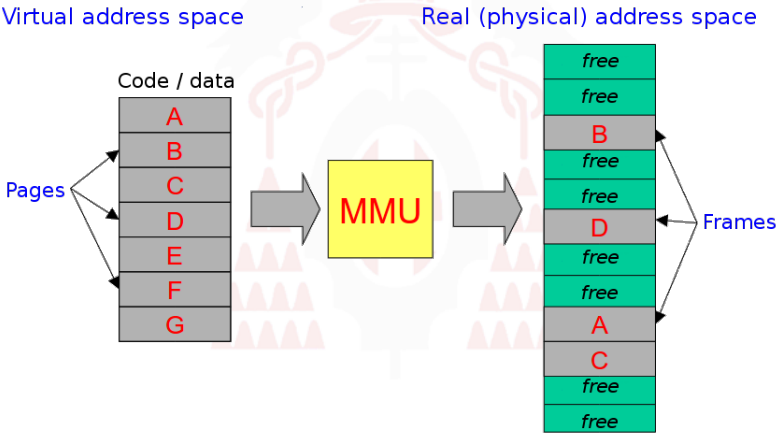
Segmentation considerations

- Advantages:
  - No internal fragmentation
  - Allows dynamic growth of segments
- Drawbacks:
  - Requires memory compacting
  - External fragmentation might occur

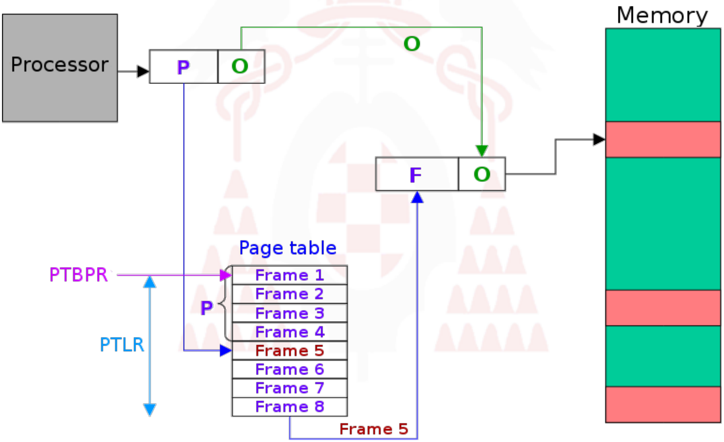
Paging

- The physical memory is initially organised in fixed-size partitions (frames)
- The virtual address space of every process is divided in fixed-size blocks (pages)
- Virtual addresses are composed of two elements: virtual page number and offset
- Includes protection mechanism and allows sharing
- The partition table is called Page Map Table (PMT)
- If the PMT is too big, it has to be stored in main memory, pointed by a register (PTBPR)

Paging: logical scheme



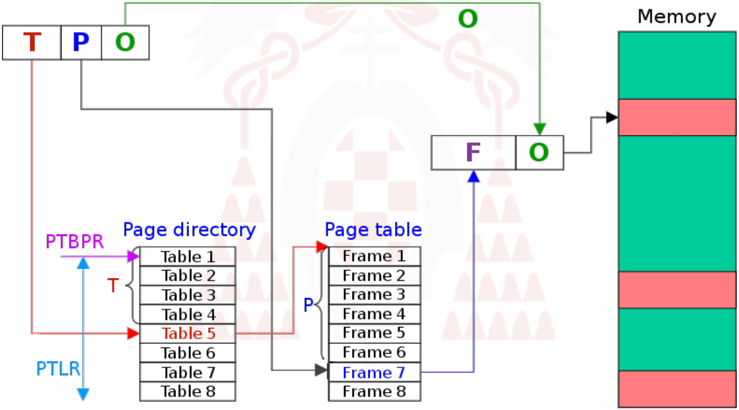
Paging: physical scheme



Paging considerations

- Advantages:
  - No external fragmentation
- Drawbacks:
  - Internal fragmentation might occur
- With large pages internal fragmentation grows but the PMT gets smaller, and vice-versa
- If the number of pages is high, the amount of memory occupied by the PMT can be prohibitive. In such cases, the PMT itself has to be paged.

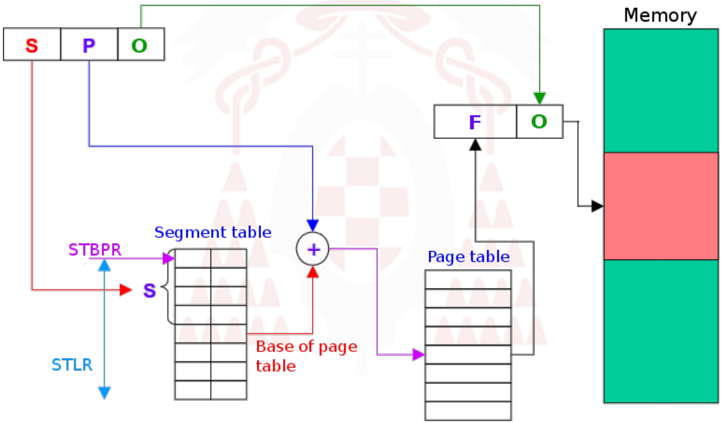
Paged paging



Combination of mechanisms

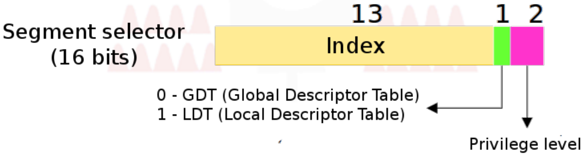
- It is possible to combine the schemes of paging and segmentation
- The advantages of both are obtained, at the price of complicating the hardware
- Possible combinations:
  - Paged segmentation
  - Segmented paging (not used in practice)

Paged segmentation: logical scheme



Pentium's MMU

- The Pentium supports segmentation, paging and paged segmentation (the most usual)
- The logical address is composed of a segment selector (13+1 bits) and an offset (32 bits)
- The segment selector is the value contained in one of the next registers: CS, DS, ES, SS, FS, GS

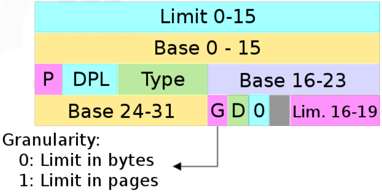


Format of the segment descriptor

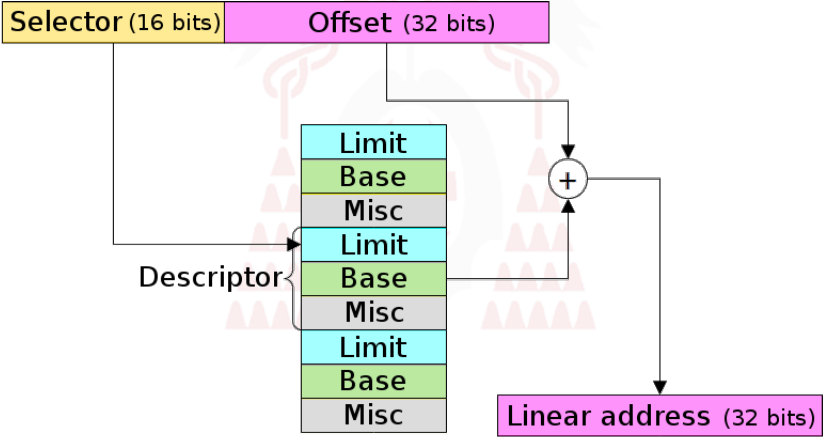
- LDT (Local Descriptor Table) $\Rightarrow$ one per process
- GDT (Global Descriptor Table) $\Rightarrow$ one per system
- Maximum number of entries per table $\Rightarrow 2^{13}$
- Each entry in the segment table is called *descriptor*
- Descriptor size $\Rightarrow$ 8 bytes

Segment descriptor

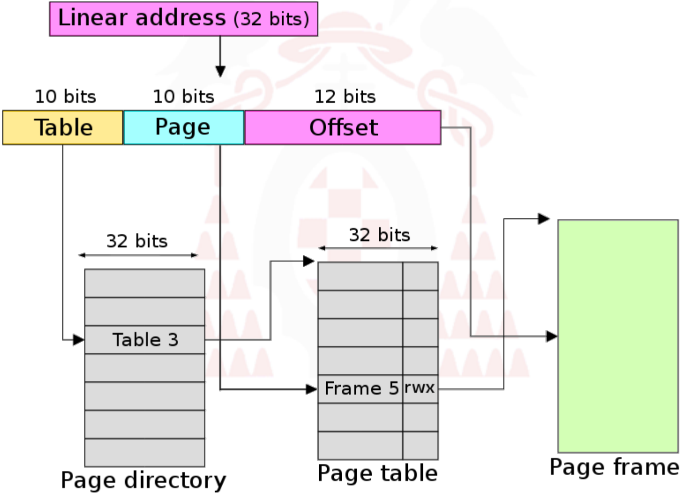
- Base address (32 bits)
- Limit (20 bits)
- Attributes and privileges (12 bits)



Pentium's segmentation stage



Pentium's paging stage



Bibliographic references I

- [Sánchez, 2005] S. Sánchez Prieto.
  Sistemas Operativos.
  Servicio de Publicaciones de la UA, 2005.
- [Tanenbaum, 2009] A. Tanenbaum.
  Sistemas Operativos Modernos.
  Ed. Pearson Education, 2009.
- [Stallings, 1999] W. Stallings.
  Organización y arquitectura de Computadores.
  Ed. Prentice Hall, 1999.
- [Silberschatz, 2006] A. Silberschatz, P. B. Galván y G. Gagne
  Fundamentos de Sistemas Operativos.
  McGraw Hill. 2006

Bibliographic references II

D.E. Knuth.
An Empirical Study of FORTRAN Programs
Software—Practice and Experience, vol. 1, 105–133, 1971.