Sommario

- Working set
- Memoria segmentata
- Architettura x86
- Gestione della memoria in Minix

Working Set theory (Peter Denning)

At any given time, each process has a working set of pages; that is, the pages that it is actually using.

The working set is usually a small portion of the entire address space of the process. The working set may change over time, triggering page faults, but these will not occur frequently.

The working set theory predicts that **if** the operating system can keep every process's working set in main memory, there will be few page faults and the system will perform well.

Locality of Reference

The **locality principle** states that processes tend to reference memory in patterns, not randomly. Memory references tend to be clustered.

- If a page is referenced, it is likely that the same page will be referenced again in the near future (temporal locality)
- If a page is referenced, it is likely that nearby pages will also be referenced (spatial locality)

Si verifica sperimentalmente che molti programmi esibiscono questa località

Gli algoritmi che esibiscono località sono da preferire

Thrashing

Se l'insieme dei working set dei processi in esecuzione è più grande della RAM disponibile

Vengono generati page fault ogni poche istruzioni

Il sistema passa il suo tempo nel memory manager

Non viene fatto lavoro utile ⇒ Thrashing (agitarsi in maniera scomposta)

Stima del working set

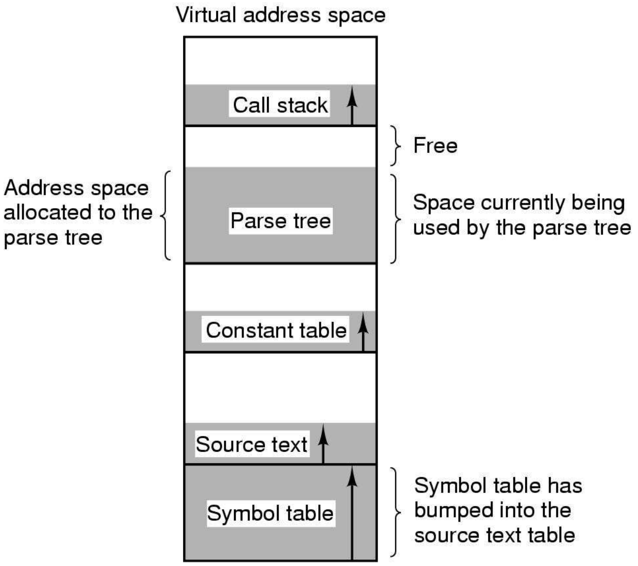
Working set: pagine accedute nell'intervallo di tempo $(t, t-\delta)$

wss_i : working set size for process i

$D = \sum_i wss_i$

obiettivo: tenere $D <$ numero di pagine fisiche

posso ridurre D swappando alcuni processi


Virtual address space
Call stack
Free
Address space allocated to the parse tree
Parse tree
Space currently being used by the parse tree
Constant table
Source text
Symbol table
Symbol table has bumped into the source text table

Memoria segmentata

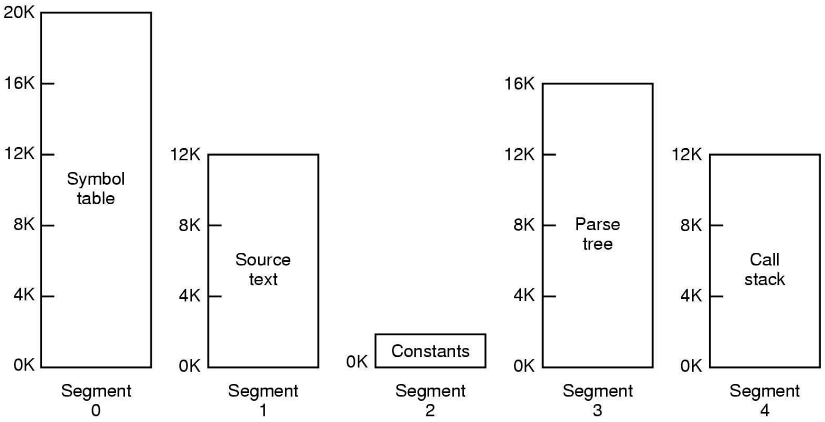
Lo spazio dei processi è suddiviso in più *segmenti*

A discrezione del programmatore

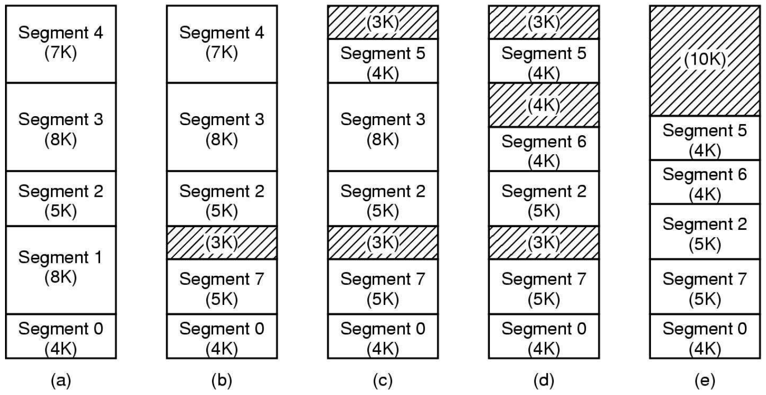
I segmenti hanno dimensione diversa ⇒ problema: la gestione della memoria diventa complicata

Occorre una *segment table*

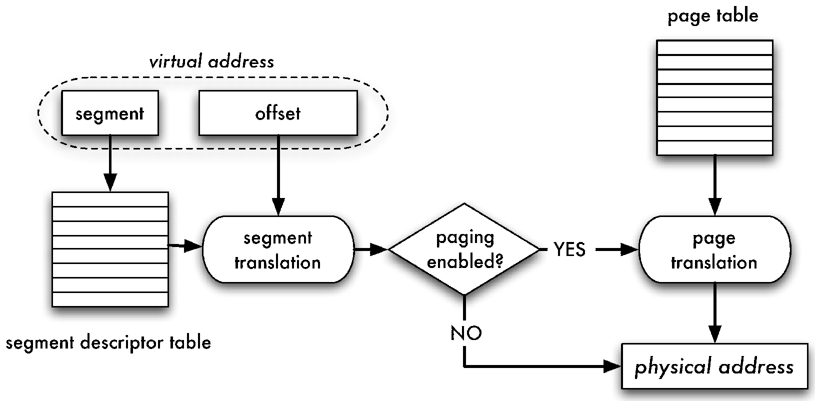
Per ogni segmento:
- base
- limit
- permissions



Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection



Memoria segmentata e paginata: l'architettura Intel x86



Memoria segmentata e paginata: l'architettura Intel x86

Spazio di indirizzamento: max 2^{16} segmenti, ciascuno di 4GB

Local Descriptor Table (LDT) per processo

- testo, stack, dati, ...

Global Descriptor Table (GDT) unica

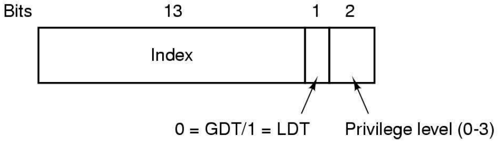
- segmenti di sistema

I registri del x86 che gestiscono i segmenti

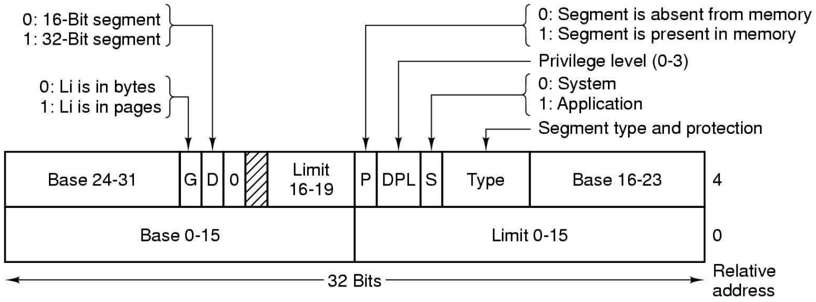
CS: registro che contiene un selector per il segmento codice

DS: registro che contiene il selector per il segmento dati

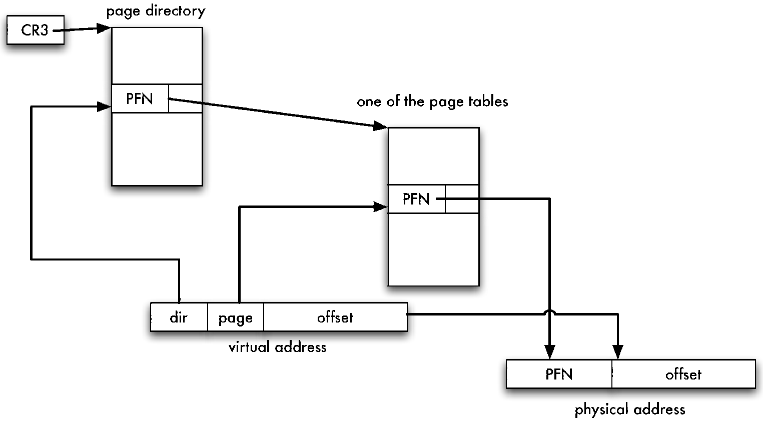
Un segment selector è un registro di 16 bit



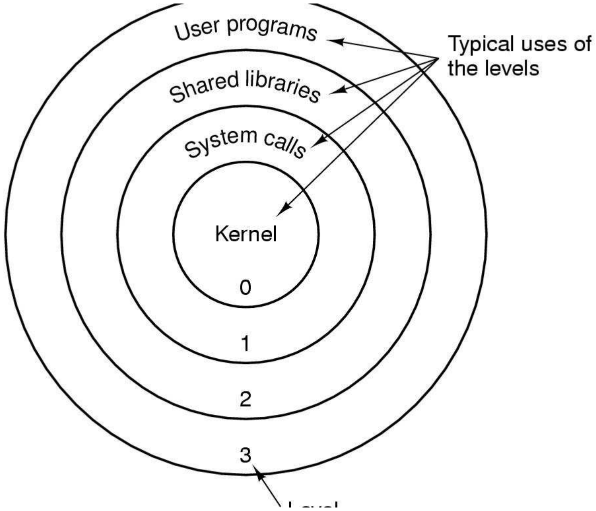
Il segment descriptor del x86



Traduzione di un indirizzo (architettura x86)



Livelli di protezione nell'architettura x86



Monitorare l'efficienza del MM

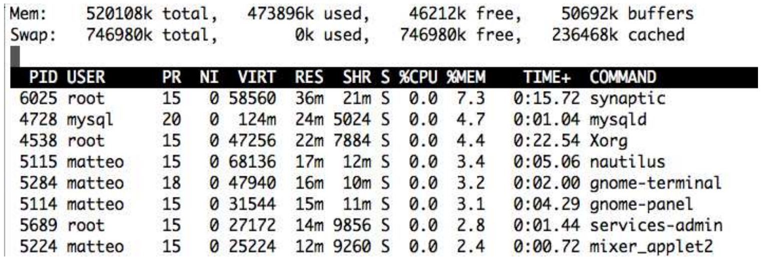
il comando /usr/bin/time(1)

il comando top(1)

il comando vmstat(1)

il comando free(1)

Il comando top(1)



```
Mem:    520108k total,   473896k used,    46212k free,    50692k buffers
Swap:   746980k total,        0k used,   746980k free,   236468k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6025 root      15   0 58560  36m  21m S  0.0  7.3   0:15.72 synaptic
 4728 mysql     20   0  124m  24m 5024 S  0.0  4.7   0:01.04 mysqld
 4538 root      15   0 47256  22m 7884 S  0.0  4.4   0:22.54 Xorg
 5115 matteo    15   0 68136  17m  12m S  0.0  3.4   0:05.06 nautilus
 5284 matteo    18   0 47940  16m  10m S  0.0  3.2   0:02.00 gnome-terminal
 5114 matteo    15   0 31544  15m  11m S  0.0  3.1   0:04.29 gnome-panel
 5689 root      15   0 27172  14m 9856 S  0.0  2.8   0:01.44 services-admin
 5224 matteo    15   0 25224  12m 9260 S  0.0  2.4   0:00.72 mixer_applet2
```

RES Resident size (kb): The non-swapped physical memory a task has used.

VIRT Virtual Image (kb) The total amount of virtual memory used by the task. It includes all code, data and shared libraries plus pages that have been swapped out. VIRT = SWAP + RES.

%MEM Memory usage (RES) A task's currently used share of available physical memory.

SHR Shared Mem size (kb) The amount of shared memory used by a task. It simply reflects memory that could be potentially shared with other processes.

Il comando vmstat(1)

```
$ vmstat 2
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0  0      0  86736  50936 234604    0    0   147   101  287  546  5  4 83  7
 0  0      0  86736  50936 234604    0    0     0     0  126   39  0  0 100  0
 0  0      0  86728  50936 234624    0    0     0     0  132   56  0  0 100  0
 0  0      0  86728  50936 234624    0    0     0     0  128   38  0  1 99  0
                     ⋮                   ⋮                   ⋮
  Memory
      swpd: the amount of virtual memory used.
      free: the amount of idle memory.
      buff: the amount of memory used as buffers.
      cache: the amount of memory used as cache.
      inact: the amount of inactive memory. (-a option)
      active: the amount of active memory. (-a option)

  Swap
      si: Amount of memory swapped in from disk (/s).
      so: Amount of memory swapped to disk (/s).

  IO
      bi: Blocks received from a block device (blocks/s).
      bo: Blocks sent to a block device (blocks/s).

  Svstem
```

Il comando /usr/bin/time

Da non confondersi con il built-in "time" di bash(1)

```
$ /usr/bin/time echo ciao
ciao
0.00user 0.00system 0:00.01elapsed 100%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+217minor)pagefaults 0swaps
```

Memory regions in Linux

```
$ cat /proc/1/maps
08048000-0805d000 r-xp 00000000 08:01 2730805    /sbin/init (executable code)
0805d000-0805e000 rwxp 00015000 08:01 2730805    /sbin/init (data)
0805e000-081ab000 rwxp 0805e000 00:00 0          [heap]
b7e7e000-b7e7f000 rwxp b7e7e000 00:00 0
b7e7f000-b7fba000 r-xp 00000000 08:01 2861752    /lib/libc-2.5.so (code)
b7fba000-b7fbb000 r-xp 0013b000 08:01 2861752    /lib/libc-2.5.so (code)
b7fbb000-b7fbd000 rwxp 0013c000 08:01 2861752    /lib/libc-2.5.so (data)
b7fbd000-b7fc0000 rwxp b7fbd000 00:00 0
b7fcd000-b7fcf000 rwxp b7fcd000 00:00 0
b7fcf000-b7fe8000 r-xp 00000000 08:01 2861615    /lib/ld-2.5.so (code)
b7fe8000-b7fea000 rwxp 00019000 08:01 2861615    /lib/ld-2.5.so (data)
bfa05000-bfa1a000 rw-p bfa05000 00:00 0          [stack]
# start  end      perm offset   major i-node
#                               minor
```