



Sistemi Operativi

Bruschi Monga Re

Memoria virtuale
Memory mapping
Gestione della memoria

Sistemi Operativi¹

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

a.a. 2016/17

¹© 2008–17 M. Monga. Creative Commons Attribuzione — Condividi allo stesso modo 4.0 Internazionale. <http://creativecommons.org/licenses/by-sa/4.0/deed.it>. Immagini tratte da [?] e da Wikipedia.

1



Sistemi Operativi

Bruschi Monga Re

Memoria virtuale
Memory mapping
Gestione della memoria

Lezione XIX: Gestione della memoria in JOS

350



Sistemi Operativi

Bruschi Monga Re

Memoria virtuale
Memory mapping
Gestione della memoria

Lab 2

```
cd solab-jos
git pull
git checkout lab2
```

Attenzione: la rete della macchina virtuale deve essere attiva.
(La branch lab2 incorpora già le mie soluzioni di lab1)

351



Sistemi Operativi

Bruschi Monga Re

Memoria virtuale
Memory mapping
Gestione della memoria

Indirizzi

Nei manuali x86 si parla di 3 tipologie di indirizzi

virtuali quando sono relativi ad un segmento: un puntatore C è un *offset*

lineare selettore di segmento + offset permette di calcolare un indirizzo nello spazio di indirizzamento (virtuale) lineare 0–4GB

fisico l'indirizzo lineare è "mappato" su un indirizzo fisico dalla MMU (che non può essere saltata!)

352



Segmentazione e MMU non possono essere saltati: il programmatore "vede" esclusivamente indirizzi virtuali.

- JOS configura tutti i segmenti (in boot/boot.S tramite la prima GDT) in 0-0xffffffff (0-4GB), quindi il segmento può essere ignorato
- Quando serve manipolare indirizzi fisici (che non possono essere dereferenziati) devono essere usati *numeri* che sarà utile contrassegnare con il tipo `physaddr_t`
- Un numero che può essere dereferenziato (perché si tratta di un indirizzo virtuale) verrà contrassegnato con `uintptr_t` e per dereferenziarlo come T va interpretato come T*.



I kernel sono generalmente caricati a un indirizzo (lineare) alto, p.es. 0xf0100000 (3,75GB), che potrebbe perfino non esistere nello spazio fisico.

- il programmatore del kernel (e il programma!) usa 0xf0100000 (virtuale)
- il boot loader carica il kernel all'indirizzo 0x00100000
- il boot loader istruisce la MMU perché mappi 0xf0100000 → 0x00100000



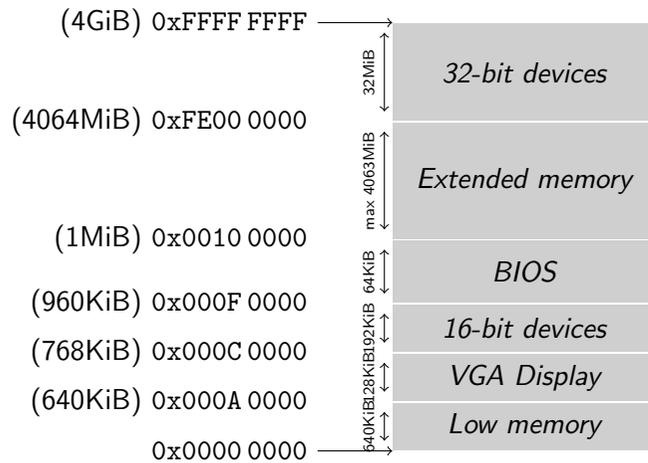
- La page table 'zeresima' in boot/boot.S configura il mapping *identità*, quindi indirizzi lineari uguali a fisici.
- La prima vera page table è in kern/entrypgdir.c

lineare	fisico
0xf0000000 (KERNBASE)	0x00000000
...	...
0xf0400000	0x00400000 (4MB)
0x00000000	0x00000000
...	...
0x00400000	0x00400000 (4MB)
*	eccezione



0xf0000000 == KERNBASE → 0x00000000
 0xf0100000 == KERNBASE + 1MB
 0xf0400000 == KERNBASE + 4MB → 0x00400000

Alla fine del lab2 verranno mappati 256MB. Si noti che esiste una relazione semplice fra fisico e lineare: quando serve il programmatore può calcolare l'indirizzo lineare aggiungendo KERNBASE al fisico. Per farlo meglio usare KADDR (e PADDR per l'inverso) che controllano che il numero cui si applica sia sensato.



Sistemi Operativi
Bruschi Monga Re

Memoria virtuale
Memory mapping
Gestione della memoria



```

// Physical page state array
struct PageInfo *pages;
// Free list of physical pages
static struct PageInfo *page_free_list;
    
```

(Lo static garantisce che page_free_list sia "privata" del file kern/pmap.c. Analogamente la variabile nextfree è privata alla funzione boot_alloc, anche se la durata del suo valore è analoga a quella di una variabile globale: si mantiene fra una chiamata e l'altra)

- ❶ L'array npages viene allocata inizialmente con boot_alloc
- ❷ Viene inizializzata con page_init; una pagina è libera se fa parte della lista collegata page_free_list
- ❸ L'allocazione poi deve avvenire sempre con page_alloc

Il reference count di una pagina (quante pagine virtuali vengono mappate su di essa) è aggiornato da page_insert. Per altri usi occorre farlo a mano.

Sistemi Operativi
Bruschi Monga Re

Memoria virtuale
Memory mapping
Gestione della memoria