



Sistemi Operativi

Bruschi  
Monga Re

Memoria virtuale

Memory mapping

Gestione della memoria

# Sistemi Operativi<sup>1</sup>

Mattia Monga

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

a.a. 2018/19

<sup>1</sup> © 2008–18 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 [2] e da Wikipedia.

1

360



Sistemi Operativi

Bruschi  
Monga Re

Memoria virtuale

Memory mapping

Gestione della memoria

# Lezione XIX: Gestione della memoria in JOS

## Lab 2



Sistemi Operativi

Bruschi  
Monga Re

Memoria virtuale

Memory mapping

Gestione della memoria

```
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)

361

## Indirizzi



Sistemi Operativi

Bruschi  
Monga Re

Memoria virtuale

Memory mapping

Gestione della memoria

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!)

362

## Come manipolare gli indirizzi



Sistemi Operativi

Bruschi  
Monga Re

Memoria virtuale

Memory mapping

Gestione della memoria

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*`.

363

## Il mapping iniziale



Sistemi Operativi

Bruschi  
Monga Re

Memoria virtuale

Memory mapping

Gestione della memoria

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`

364

## le prime page table



Sistemi Operativi

Bruschi  
Monga Re

Memoria virtuale

Memory mapping

Gestione della memoria

- La page table ‘zeresima’ in `boot/boot.S` configura il mapping *identità*, quindi indirizzi lineari uguali a fisici
- La prima (*trivial*) page table l’allestisce `kern/entry.S`
- La page table è in `kern/entrypgdir.c` (scritta esplicitamente riga per riga!)

lineare	fisico
<code>0xf0000000</code> (KERNBASE)	<code>0x00000000</code>
...	...
<code>0xf0400000</code>	<code>0x00400000</code> (4MB)
<code>0x00000000</code>	<code>0x00000000</code>
...	...
<code>0x00400000</code>	<code>0x00400000</code> (4MB)
*	eccezione

365

## Macro che sostituiscono la MMU



Sistemi Operativi

Bruschi  
Monga Re

Memoria virtuale

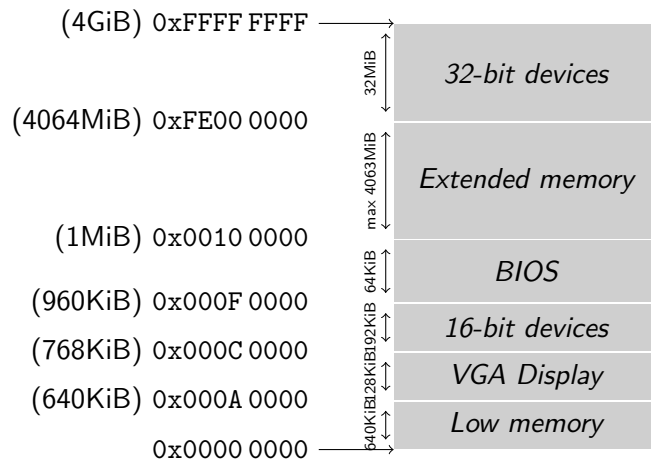
Memory mapping

Gestione della memoria

```
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.

366



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)

- ① L'array pages viene allocata inizialmente con boot\_alloc
- ② Viene inizializzata con page\_init; una pagina è libera se fa parte della lista collegata page\_free\_list
- ③ 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