



Sistemi
Operativi

Bruschi
Monga

JOS

La gestione della
memoria

Paginazione

JOS

La gestione della
memoria

Sistemi Operativi¹

Mattia Monga

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

a.a. 2019/20

¹ © 2008–19 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.



Sistemi
Operativi

Bruschi
Monga

JOS

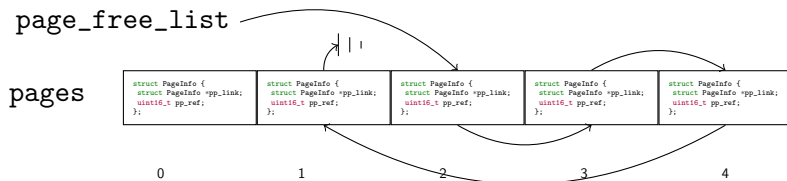
La gestione della
memoria

Paginazione

JOS

La gestione della
memoria

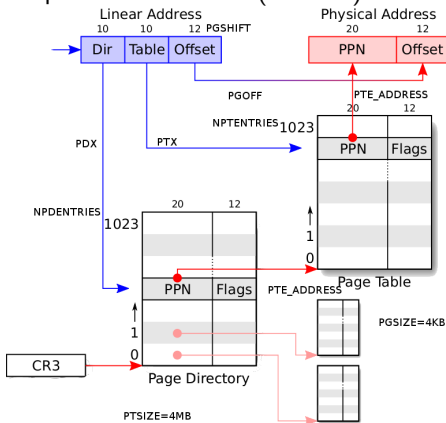
Lezione XXII: Gestione della memoria



- pages[0] non è disponibile
- pages[2] è disponibile
- Si riferisce alla pagina all'indirizzo virtuale
`page2kva(&pages[2]) == KERNBASE + 2*PGSIZE`
- `page_alloc(0)` ritornerebbe `&pages[2]` (togliendolo dalla `page_free_list`)
- Quattro `page_alloc(0)` di fila esauriscono le pagine a disposizione e la quinta fallirebbe

Paginazione

Una paginazione *diretta* con 20+12 bit, avrebbe 2^{20} Page Table Entry (PTE). Se ogni PTE è 32 bit (20 per il mapping e 12 per i flag) si hanno 4MB per la *page table*: con **2 livelli** (da 10 bit) si possono risparmiare le tabelle (da 4KB) di secondo livello non mappate.



- PDE Page Directory Entry
- PPN Physical Page Number
- Flags
 - PTE_P (presente)
 - PTE_W (scrivibile)
 - PTE_U (utilizzabile in modalità user)

PDE, PTE e CR3



Sistemi Operativi

Bruschi Monga

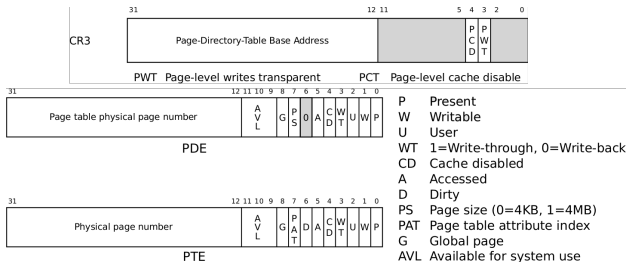
JOS

La gestione della memoria

Paginazione

JOS

La gestione della memoria



In `inc/mmu.h` vengono definite un po' di macro utili:

```
// A linear address 'la' has a three-part structure as follows:  
//  
// +-----10-----+-----10-----+-----12-----+  
// | Page Directory | Page Table | Offset within Page |  
// |   Index      |   Index    |  
// +-----+-----+-----+  
// \--- PDX(la) --/ \--- PTX(la) --/ \--- PGOFF(la) ----/  
// \----- PGNUM(la) -----/  
//  
// The PDX, PTX, PGOFF, and PGNUM macros decompose linear addresses as shown.  
// To construct a linear address la from PDX(la), PTX(la), and PGOFF(la),  
// use PGADDR(PDX(la), PTX(la), PGOFF(la)).  
// Address in page table or page directory entry  
#define PTE_ADDR(pte) ((physaddr_t) (pte) & ~0xFFF)
```

Altre macro utili



Sistemi
Operativi

Bruschi
Monga

```
// Page directory and page table constants.
#define NPENTRIES 1024      // page directory entries per page directory
#define NPENTRIES 1024      // page table entries per page table

#define PGSIZE 4096         // bytes mapped by a page
#define PGSHIFT 12          // log2(PGSIZE)

#define PTSIZE (PGSIZE*ES) // bytes mapped by a page directory entry
#define PTSHIFT 22          // log2(PTSIZE)

#define PTXSHIFT 12        // offset of PTX in a linear address
#define PDXSHIFT 22        // offset of PDX in a linear address

// Page table/directory entry flags.
#define PTE_P 0x001        // Present
#define PTE_W 0x002        // Writeable
#define PTE_U 0x004        // User

// Address in page table or page directory entry
#define PTE_ADDR(pte) ((physaddr_t) (pte) & ~0xFFF)
```

JOS

La gestione della
memoria

Paginazione

JOS

La gestione della
memoria



Il funzionamento

In kern/entry.S CR3 viene settato all'indirizzo **fisico** della page directory. (Dato il mapping iniziale i fisici possono essere dedotti anche *aritmeticamente* togliendo KERNBASE dal virtuale)

```
// pseudo-codice
CR3 = (physaddr_t)0x00115000 // i 12 bit finali per i flag
// i 20 bit alti vanno cmq interpretati come multipli di 0x10000
// perche` la tabella devono iniziare a indirizzi allineati
entry_pgdirt = (uintptr_t)0xF0115000 = PGADDR(0x3c0, 0x115, 0)

// il primo livello di mapping
pde_t entry_pgdirt[NPDENTRIES] = {
    // Map VA's [0, 4MB) to PA's [0, 4MB)
    [0] = ((uintptr_t)entry_pgtable - KERNBASE) + PTE_P,
    // Map VA's [KERNBASE, KERNBASE+4MB) to PA's [0, 4MB)
    [KERNBASE>>PDXSHIFT] = ((uintptr_t)entry_pgtable - KERNBASE) + PTE_P + PTE_W
};

// e finalmente
pte_t entry_pgtable[NPTENTRIES] = {
    0x000000 | PTE_P | PTE_W,
    0x001000 | PTE_P | PTE_W,
    0x002000 | PTE_P | PTE_W,
    // ...
    0x3fe000 | PTE_P | PTE_W,
    0x3ff000 | PTE_P | PTE_W,
};
```

Sistemi
Operativi

Bruschi
Monga

JOS

La gestione della
memoria

Paginazione

JOS

La gestione della
memoria



La consultazione dei due livelli avviene tramite `page_walk` che tratta anche il caso in cui il secondo livello non sia in memoria.

Un esempio numerico:

```
kernpgdir[PDX(UVPT)] = PADDR(kernpgdir) | PTE_U | PTE_P
```

```
UVPT == KSTACKTOP - 3*PTSIZE == 0xf0000000 - 3*4*1024*1024 == 0xef400000
```

```
PDX(0xef400000) == 0x3bd & 0x03ff
```

```
kernpgdir == 0xf0119000
```

```
PADDR(kernpgdir) == kernpgdir - KSTACKTOP == 0xf0119000 - 0xf0000000
```

```
kernpgdir[0x03bd] = 0x00119005
```




- Il setup della memoria avviene in `mem_init`
- La funzione di servizio principale è `boot_map_region`
- Allo scopo serve:
 - Gestire la relazione con la MMU: `pgdir_walk`, `page_insert`, `page_remove`, `page_lookup`
 - Gestire le strutture dati `struct PageInfo pages[]` e `page_free_list`: `page_init`, `page_alloc`, `page_free`, `page_decref`

Il mapping finale



Sistemi
Operativi

Bruschi
Monga

JOS

La gestione della
memoria

Paginazione

JOS

La gestione della
memoria

PGSIZE = 4096 (0x1000)

PTSIZE = 4M (0x400000)

simbolo	va	PDX	fisico
(4G)	0xffff ffff	0x3ff	va - KERNBASE
KERNBASE, KSTACKTOP	0xf000 0000	0x3c0	va - KERNBASE
MMIOLIM	0xefc0 0000	0x3bf	...page_alloc...
MMIOBASE, ULIM	0xef80 0000	0x3be	...page_alloc...
UVPT	0xef40 0000	0x3be	...page_alloc...
UPAGES	0xef00 0000	0x3bc	...page_alloc...
UXSTACKTOP, UTOP, UENVS	0xee0 0000	0x3bb	
USTACKTOP	0xeebf e000	0x3ba	
UTEXT	0x0080 0000	0x2	
PFTEMP	0x007f f000	0x1	
UTEMP	0x0040 0000	0x1	
USTABDATA	0x0020 0000	0x0	
EXTPHYSMEM	0x0010 0000	0x0	
IOPHYSMEM	0x000a 0000	0x0	
(0)	0x0000 0000	0x0	