Sistemi Operativi[1]

Mattia Monga

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

a.a. 2012/13

Lezione XIX: Gestione della memoria in JOS

Le strutture dati per la gestione della memoria

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

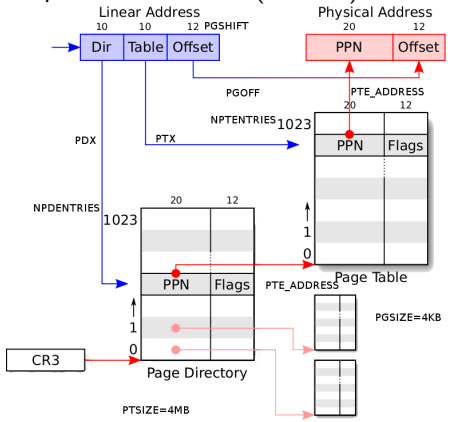
(Lo static garantisce che page_free_list sia "privata" del file `kern/pmap.c`. Analogamente la variabile nextfree è privata alla funzione boot_alloc, anche se la durata del suo valore è analoga a quella di una variabile globale: si mantiene fra una chiamata e l'altra)

1. L'array npages 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.

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.

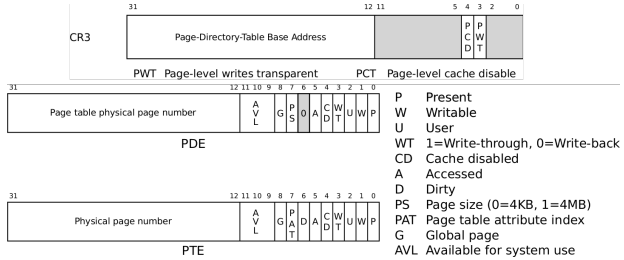


PDE Page Directory Entry

PPN Physical Page Number

Flags PTE_P (presente) PTE_W (scrivibile) PTE_U (utilizzabile in modalità user)

PDE, PTE e CR3



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

```
1  // A linear address 'la' has a three-part structure as follows:
2  //
3  // +--------10------+-------10-------+---------12----------+
4  // | Page Directory |   Page Table    | Offset within Page  |
5  // |      Index     |      Index      |                     |
6  // +----------------+-----------------+---------------------+
7  //  \--- PDX(la) --/ \--- PTX(la) --/ \---- PGOFF(la) ----/
8  //  \---------- PGNUM(la) ----------/
9  //
10 // The PDX, PTX, PGOFF, and PGNUM macros decompose linear addresses as shown.
11 // To construct a linear address la from PDX(la), PTX(la), and PGOFF(la),
12 // use PGADDR(PDX(la), PTX(la), PGOFF(la)).
```


Altre macro utili

```
1  // Page directory and page table constants.
2  #define NPDENTRIES 1024 // page directory entries per page directory
3  #define NPTENTRIES 1024 // page table entries per page table
4
5  #define PGSIZE 4096 // bytes mapped by a page
6  #define PGSHIFT 12 // log2(PGSIZE)
7
8  #define PTSIZE (PGSIZE*ES) // bytes mapped by a page directory entry
9  #define PTSHIFT 22 // log2(PTSIZE)
10
11 #define PTXSHIFT 12 // offset of PTX in a linear address
12 #define PDXSHIFT 22 // offset of PDX in a linear address
13
14 // Page table/directory entry flags.
15 #define PTE_P 0x001 // Present
16 #define PTE_W 0x002 // Writeable
17 #define PTE_U 0x004 // User
18
19 // Address in page table or page directory entry
20 #define PTE_ADDR(pte) ((physaddr_t) (pte) & ~0xFFF)
```


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)

```
1  // pseudo-codice
2  CR3 = (physaddr_t)0x00115000 // i 12 bit finali per i flag
3  // i 20 bit alti vanno cmq interpretati come multipli di 0x10000
4  // perche' la tabelle devono iniziare a indirizzi allineati
5  entry_pgdir = (uintptr_t)0xF0115000 = PGADDR(0x3c0, 0x115, 0)
6
7  // il primo livello di mapping
8  pde_t entry_pgdir[NPDENTRIES] = {
9  // Map VA's [0, 4MB) to PA's [0, 4MB)
10 [0] = ((uintptr_t)entry_pgtable - KERNBASE) + PTE_P,
11 // Map VA's [KERNBASE, KERNBASE+4MB) to PA's [0, 4MB)
12 [KERNBASE>>PDXSHIFT] = ((uintptr_t)entry_pgtable - KERNBASE) + PTE_P + PTE_W
13 };
14
15 // e finalmente
16 pte_t entry_pgtable[NPTENTRIES] = {
17 0x000000 | PTE_P | PTE_W,
18 0x001000 | PTE_P | PTE_W,
19 0x002000 | PTE_P | PTE_W,
20 // ...
21 0x3fe000 | PTE_P | PTE_W,
22 0x3ff000 | PTE_P | PTE_W,
23 };
```


La consultazione delle tabelle

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:

```
1  kernpgdir[PDX(UVPT)] = PADDR(kernpgdir) | PTE_U | PTE_P
2
3  UVPT== KSTACKTOP - 3*PTSIZE== 0xf0000000 - 3*4*1024*1024== 0xef400000
4
5  PDX(0xef400000)== 0x3bd & 0x03ff
6
7  kernpgdir== 0xf0119000
8  PADDR(kernpgdir)== kernpgdir - KSTACKTOP== 0xf0119000 - 0xf0000000
9
10
11 kernpgdir[0x03bd] = 0x00119005
```