



Sistemi  
Operativi

Bruschi  
Monga Re

# Sistemi Operativi<sup>1</sup>

Mattia Monga

Dip. di Informatica  
Università degli Studi di Milano, Italia  
[mattia.monga@unimi.it](mailto:mattia.monga@unimi.it)

a.a. 2014/15

JOS

La gestione della  
memoria  
Paginazione

JOS

La gestione della  
memoria  
Strutture dati  
per i programmi  
utente  
Gestione  
eccezioni  
Esecuzione di un  
programma  
utente



Sistemi  
Operativi

Bruschi  
Monga Re

# Lezione XXII: Esecuzione di un programma utente in JOS

JOS

La gestione della memoria  
Paginazione

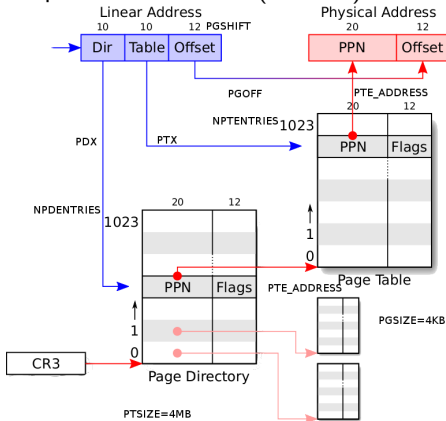
JOS

La gestione della memoria  
Strutture dati per i programmi utente  
Gestione eccezioni  
Esecuzione di un programma utente

# 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 (present)  
PTE\_W (writable)  
PTE\_U (user accessible)

Sistemi Operativi

Bruschi Monga Re

JOS

La gestione della memoria

Paginazione

JOS

La gestione della memoria

Strutture dati per i programmi utente

Gestione eccezioni

Esecuzione di un programma utente

# PDE, PTE e CR3



Sistemi Operativi

Bruschi Monga Re

JOS

La gestione della memoria

Paginazione

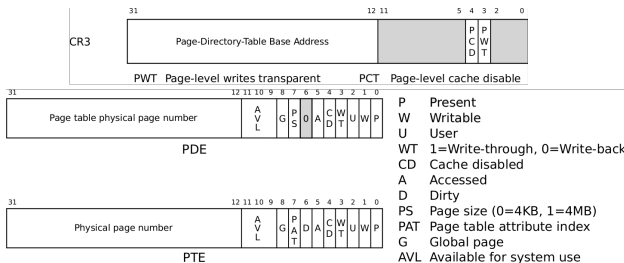
JOS

La gestione della memoria

Strutture dati per i programmi utente

Gestione eccezioni

Esecuzione di un programma utente



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



## JOS

La gestione della  
memoria

Paginazione

## JOS

La gestione della  
memoria

Strutture dati  
per i programmi  
utente

Gestione  
eccezioni

Esecuzione di un  
programma  
utente

```
1 // Page directory and page table constants.
2 #define NPENTRIES 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 };
```

Sistemi  
Operativi

Bruschi  
Monga Re

JOS

La gestione della  
memoria

Paginazione

JOS

La gestione della  
memoria

Strutture dati  
per i programmi  
utente

Gestione  
eccezioni

Esecuzione di un  
programma  
utente



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
```

JOS

La gestione della  
memoria

Paginazione

JOS

memoria

Strutture dati  
per i programmi  
utente

Gestione  
eccezioni

Esecuzione di un  
programma  
utente



- 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`

## JOS

La gestione della  
memoria  
Paginazione

## JOS

La gestione della  
memoria  
Strutture dati  
per i programmi  
utente  
Gestione  
eccezioni  
Esecuzione di un  
programma  
utente



# Il mapping finale



Sistemi  
Operativi

Bruschi  
Monga Re

JOS

La gestione della  
memoria  
Paginazione

JOS

La gestione della  
memoria  
Strutture dati  
per i programmi  
utente  
Gestione  
eccezioni  
Esecuzione di un  
programma  
utente

- 1 PGSIZE = 4096 (0x1000)
- 2
- 3 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	



La gestione è simile a quella di pages

```
1 struct Env *envs = NULL; // All environments
2 struct Env *curenv = NULL; // The current env
3 static struct Env *env_free_list; // Free environment list
4
5 struct Env {
6     struct Trapframe env_tf; // Saved registers
7     struct Env *env_link; // Next free Env
8     envid_t env_id; // Unique environment identifier
9     envid_t env_parent_id; // env_id of this env's parent
10    enum EnvType env_type; // Indicates special system environments
11    unsigned env_status; // Status of the environment
12    uint32_t env_runs; // Number of times environment has run
13
14    // Address space
15    pde_t *env_pgdir; // Kernel virtual address of page dir
16 };
```

Per ogni programma è previsto un nuovo mapping (env\_pgdir)!  
I programmi sono nella memoria del kernel, non nel file system (che non c'è): li carica load\_icode (per scriverla conviene copiare la gestione ELF dal boot)

JOS

La gestione della  
memoria  
Paginazione

JOS

La gestione della  
memoria  
Strutture dati  
per i programmi  
utente  
Gestione  
eccezioni  
Esecuzione di un  
programma  
utente



Il nocciolo è nel fatto che gli indirizzi contenuti nel programma utente fanno riferimento allo spazio di indirizzamento user. Per rendere “facile” il ciclo cambio la paginazione.

```

1 struct Elf *eb = (struct Elf*) binary;
2 struct Proghdr *ph, *eph;
3
4 if (eb->e_magic != ELF_MAGIC) panic("Invalid binary!");
5
6 ph = (struct Proghdr *) (binary + eb->e_phoff);
7 eph = ph + eb->e_phnum;
8 lcr3(PADDR(e->env_pgdir));
9 while (ph < eph){
10 if (ph->p_type == ELF_PROG_LOAD){
11 region_alloc(e, (void*)ph->p_va, ph->p_memsz);
12 memset((void*)ph->p_va, 0, ph->p_memsz);
13 memcpy((void*)ph->p_va, (void*)(binary + ph->p_offset), ph->p_filesz);
14 }
15 ph += 1;
16 }
17
18 lcr3(PADDR(kern_pgdir));
19 e->env_tf.tf_eip = eb->e_entry;

```

## JOS

La gestione della  
memoria  
Paginazione

## JOS

La gestione della  
memoria  
**Strutture dati  
per i programmi  
utente**  
Gestione  
eccezioni

Esecuzione di un  
programma  
utente

# Gestione delle eccezioni (e interruzioni)



Sistemi  
Operativi

Bruschi  
Monga Re

Il meccanismo hardware è il medesimo, logicamente si tratta di un *protected control transfer*

**Interrupt** asincrono, generato dalle periferiche

**Exception** sincrono, generato dai programmi (per errori o esplicite istruzioni come `int`)

Il punto fondamentale è che deve essere il **kernel** a decidere l'indirizzo di esecuzione della "gestione" e non chi genera l'eccezione.

JOS

La gestione della  
memoria  
Paginazione

JOS

La gestione della  
memoria  
Strutture dati  
per i programmi  
utente

Gestione  
eccezioni

Esecuzione di un  
programma  
utente



Il meccanismo hardware per imporre il trasferimento di controllo è che la gestione passa per l'IDT.

```
1 mov eax, 3
2 int 0x80
```

Il vettore 0x80 seleziona una riga dell'IDT che contiene (ce li ha messi il kernel...)

- eip e cs (fondamentale per i privilegi della gestione)
- TSS: serve per tenere uno stack speciale (kernel) dove salvare lo stato dei programmi utente interrotti.

## JOS

La gestione della  
memoria  
Paginazione

## JOS

La gestione della  
memoria  
Strutture dati  
per i programmi  
utente

**Gestione  
eccezioni**

Esecuzione di un  
programma  
utente





## JOS

La gestione della  
memoria  
Paginazione

## JOS

La gestione della  
memoria  
Strutture dati  
per i programmi  
utente

## Gestione eccezioni

Esecuzione di un  
programma  
utente

```
1
2 name: /* function starts here */
3     pushl $(num) /* error code */
4     jmp _alltraps
5
6 _alltraps:
7     pushl %ds // see Trapframe in inc/trap.h
8     pushl %es // see Trapframe in inc/trap.h
9     pushal
10    movw $GD_KD, %ax
11    movw %ax, %ds
12    movw %ax, %es
13    pushl %esp
14    call trap
```

Per popolare la IDT usare SETGATE  
SETGATE(idt[...], 1, GD\_KT, ..., 0);



```
1 env_pop_tf(&curenv->env_tf);
```

“Ripristina” lo stato del programma utente. . .