Paginación: MMU Básica

Marco Vanotti

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

23 de Octubre de 2014

La clase pasada

1. Armaron un directorio de páginas con *identity mapping* para el rango `0x00000000 - 0x003FFFFF` (kernel, libre).

2. Activaron paginación.

3. Imprimieron en pantalla el nombre de su grupo.

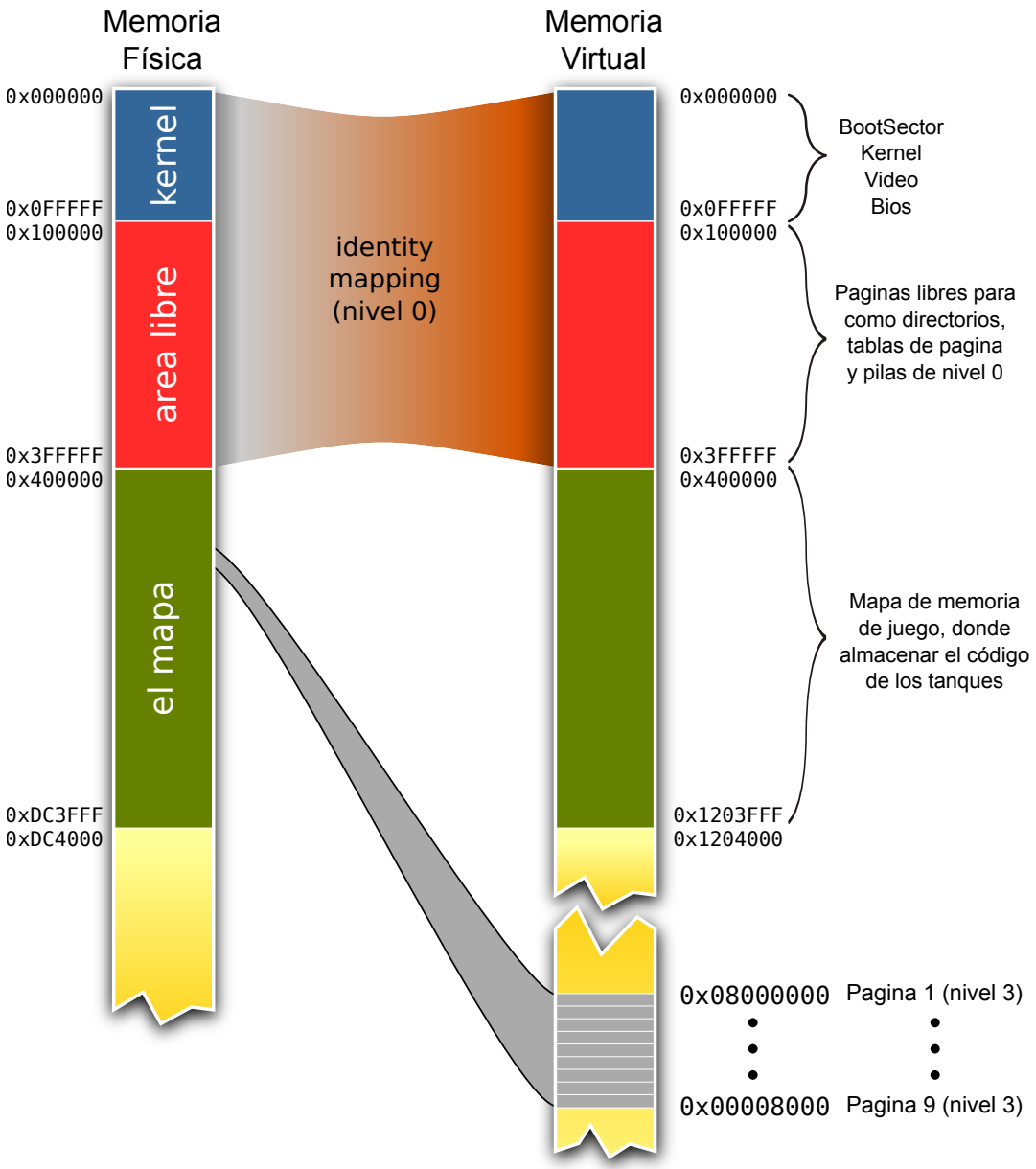
Hoy...

- Hoy van a implementar una pequeña

Memory **M**anagement **U**nit.

- ¿Qué tiene que poder hacer?
  - Inicializar la MMU
  - Inicializar el mapa de memoria de las *tareas zombies*
  - Mapear páginas
  - Desmapear páginas

Mapa de memoria de una tarea



Cómo construimos un mapa de memoria?

- Primero vamos a necesitar *obtener* una página libre para el *directorio* y otras 2 para las *tablas* de páginas.
- Estás páginas las podemos obtener de la memoria libre kernel. (de 0x100000 a 0x3FFFFF)
- *Vinculamos* las tablas con el directorio.
- Hacemos *identity mapping* sobre las direcciones **`0x00000000`** a **`0x003FFFFF`**.
- Luego, necesitamos mapear las páginas de **código** y las adyacentes.
- Sabemos cuáles son las direcciones virtuales que vamos a mapear:
  - **`0x08000000`** a **`0x08008000`**.
- ¿A qué dirección física las mapeamos?

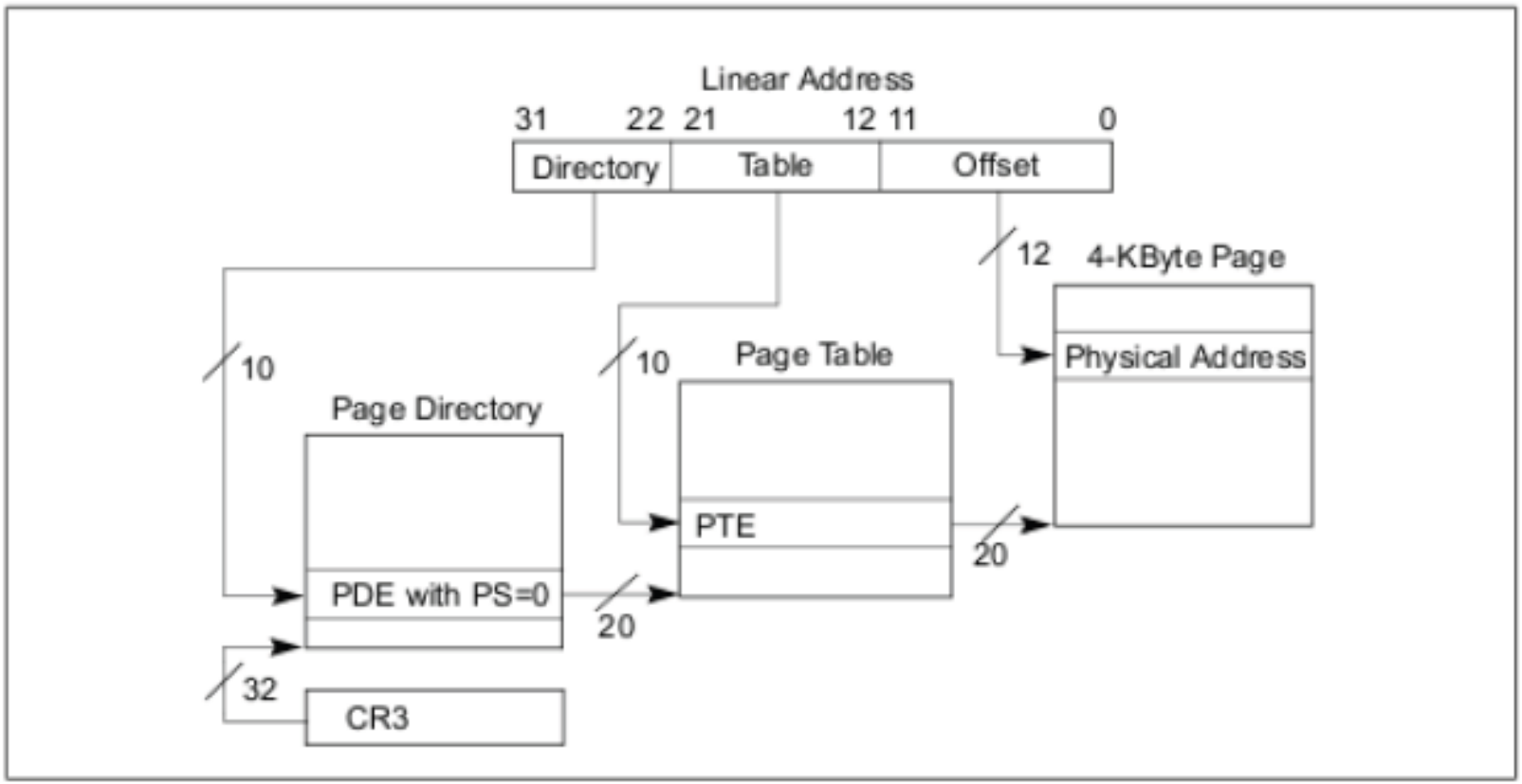
Mapear Páginas

Para llevar a cabo esto último, usaremos una función:

- ```
  void mmu_mapear_pagina(
          unsigned int virtual,
          unsigned int cr3,
          unsigned int fisica,
          unsigned int attrs)
  ```

Se encarga de *mapear* direcciones *virtuales* a direcciones *físicas* en un mapa de memoria dado (a través de un directorio de páginas).
  ```

Directorios y tablas...



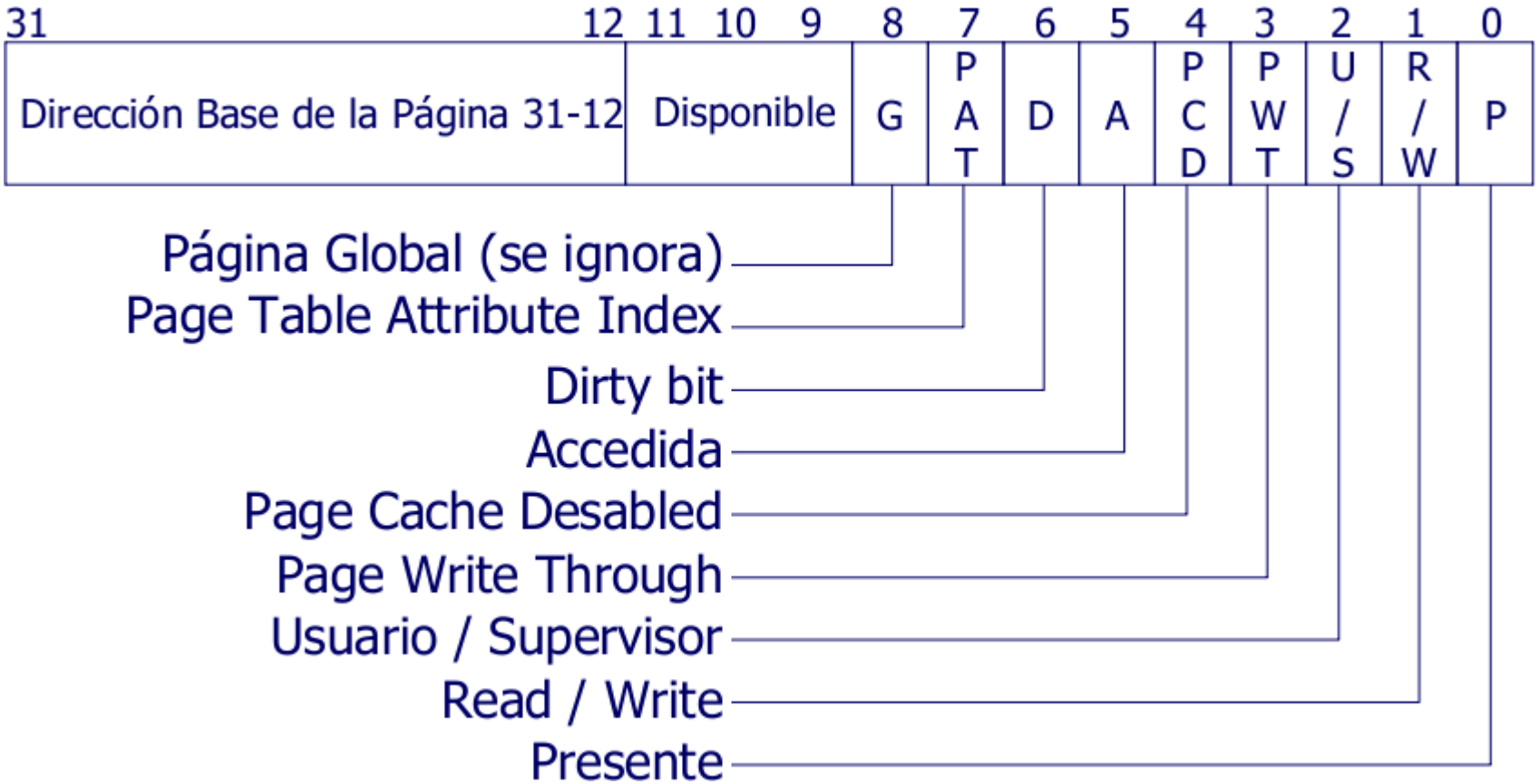
Mapear

1. Tomamos la dirección virtual y la descomponemos en sus partes:
   - *Índice en el directorio*,
   - *Índice en en la tabla* y
   - Desplazamiento dentro de la página (no la vamos a necesitar).
2. Obtenemos la *PDE* correspondiente.
3. Obtenemos la *PTE* correspondiente.
4. Completamos la *PTE* según corresponda.

PDE


31
12
11
10
9
8
7
6
5
4
3
2
1
0
Dirección Base de la Página 31-12
Disponible
G
PS
0
A
PCD
PWT
U/S
R/W
P
Página Global (se ignora)
Tamaño de Página (0=4Kbytes)
Reservado (poner en 0)
Accedida
Page Cache Desabled
Page Write Through
Usuario / Supervisor
Read / Write
Presente

PTE


31
12
11
10
9
8
7
6
5
4
3
2
1
0
Dirección Base de la Página 31-12
Disponible
G
PAT
D
A
PCD
PWT
U/S
R/W
P
Página Global (se ignora)
Page Table Attribute Index
Dirty bit
Accedida
Page Cache Desabled
Page Write Through
Usuario / Supervisor
Read / Write
Presente

Finalmente...

- Deben ejecutar la función *tlbflush()* para invalidar la cache de traducción de direcciones (en `mmu_mapear_pagina`).

Ejercicio 4

Implementar en *mmu.c* las funciones:

- *mmu_inicializar*
- *mmu_inicializar_dir_zombie*
  (no olvidarse de copiar el código de los zombies...)
- *mmu_mapear_pagina*
- *mmu_unmapear_pagina*

Ejercicio 4

Además, deben probar que las funciones que implementaron hacen lo que se supone que hacen. Para eso, en *kernel.asm*:

- Llaman a *mmu_inicializar* y luego
- a *mmu_inicializar_tarea*
- Cambian el *cr3* actual por el que retorna la función
- y cambian el color del fondo del primer caracter de la pantalla.
- Por último, vuelven a setear el *cr3* que tenían originalmente.

Algunas macros de C útiles...

```
#define   PDE_INDEX(virtual)   ???
#define   PTE_INDEX(virtual)   ???
#define   ALIGN(dir)           ???
#define   PG_PRESENT           ???
#define   PG_READ_WRITE        ???
#define   PG_USER              ???
```

¿Preguntas?