

Práctica de Organización del Computador II

Paginación

Segundo Cuatrimestre 2023

Organización del Computador II
DC - UBA

Introducción

En la clase de hoy vamos a ver:

En la clase de hoy vamos a ver:

- **Repaso de paginación como mecanismo**

En la clase de hoy vamos a ver:

- **Repaso de paginación como mecanismo**
- **Proceso de traducción de direcciones virtuales a direcciones físicas**

En la clase de hoy vamos a ver:

- Repaso de paginación como mecanismo
- Proceso de traducción de direcciones virtuales a direcciones físicas
- Estructuras involucradas en paginación (CR3, `page directory`, `page table`)

En la clase de hoy vamos a ver:

- **Repaso de paginación como mecanismo**
- **Proceso de traducción de direcciones virtuales a direcciones físicas**
- **Estructuras involucradas en paginación (CR3, `page directory`, `page table`)**
- **Atributos relevantes**

En la clase de hoy vamos a ver:

- **Repaso de paginación como mecanismo**
- **Proceso de traducción de direcciones virtuales a direcciones físicas**
- **Estructuras involucradas en paginación (CR3, `page directory`, `page table`)**
- **Atributos relevantes**
- **Casos de uso para el mecanismo de paginación**

La idea es presentar la información necesaria para ayudarles a completar el taller.

Repaso de paginación

¿Qué significa paginar y qué función cumple la unidad de paginación?

¿Qué significa paginar y qué función cumple la unidad de paginación?

- La tarea más relevante desde el punto de vista funcional que cumple la unidad de paginación es la de **traducir una dirección virtual en una dirección física**.

¿Qué significa paginar y qué función cumple la unidad de paginación?

- La tarea más relevante desde el punto de vista funcional que cumple la unidad de paginación es la de **traducir una dirección virtual en una dirección física**.
- Esta traducción permite separar el espacio de direcciones expuestas al proceso (dir. virtuales) de su ubicación en la memoria física (dir. físicas en la memoria principal).

¿Por qué queremos dividir el espacio de direcciones virtuales del espacio físico?

¿Por qué queremos dividir el espacio de direcciones virtuales del espacio físico?

El motivo más relevante tiene que ver con poder separar el espacio de distintos procesos que se ejecutan sobre el mismo procesador y por ende comparten una misma memoria principal.

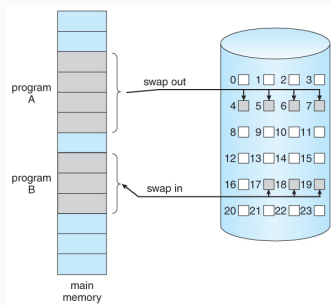
¿Por qué queremos dividir el espacio de direcciones virtuales del espacio físico?

El motivo más relevante tiene que ver con poder separar el espacio de distintos procesos que se ejecutan sobre el mismo procesador y por ende comparten una misma memoria principal.

Cada proceso puede manejarse sobre su espacio virtual como si fuera el físico, luego el sistema operativo puede administrar la memoria física como sea necesario, de forma dinámica y sin afectar la ejecución de cada proceso.

Lo vamos a ver mejor al cierre de la intro, pero este mecanismo permite por ejemplo que dos procesos utilicen el mismo rango de memoria virtual pero se ubiquen en dos rangos distintos de memoria física, incluso moviéndolos desde el disco a la memoria principal y viceversa.

Lo vamos a ver mejor al cierre de la intro, pero este mecanismo permite por ejemplo que dos procesos utilicen el mismo rango de memoria virtual pero se ubiquen en dos rangos distintos de memoria física, incluso moviéndolos desde el disco a la memoria principal y viceversa.



La unidad va utilizar dos elementos para realizar la traducción:

La unidad va utilizar dos elementos para realizar la traducción:

- **La dirección virtual** que es la que conoce el proceso.

La unidad va utilizar dos elementos para realizar la traducción:

- **La dirección virtual** que es la que conoce el proceso.
- **La estructura de paginación (directorio y tablas de páginas) del proceso** qué indican qué direcciones virtuales se corresponden a qué direcciones físicas.

La ubicación del directorio de páginas se encuentra en el registro de control CR3, con lo cual si la dirección virtual a resolver es `virt` y su dirección física asociada es `phys`, podemos definir el proceso de traducción implementado por la unidad de paginación como:

La ubicación del directorio de páginas se encuentra en el registro de control CR3, con lo cual si la dirección virtual a resolver es `virt` y su dirección física asociada es `phys`, podemos definir el proceso de traducción implementado por la unidad de paginación como:

- $map : uint32 \times CR3 \rightarrow uint32$

La ubicación del directorio de páginas se encuentra en el registro de control CR3, con lo cual si la dirección virtual a resolver es *virt* y su dirección física asociada es *phys*, podemos definir el proceso de traducción implementado por la unidad de paginación como:

- $map : uint32 \times CR3 \rightarrow uint32$
- $map(virt, CR3) = phys.$

Traduciendo direcciones

Veamos ahora cómo se traducen las direcciones virtuales a partir de un valor de dirección virtual y un directorio de páginas.

Veamos ahora cómo se traducen las direcciones virtuales a partir de un valor de dirección virtual y un directorio de páginas. Es importante notar que vamos a observar la dirección virtual y **dividirla en tres partes**.

Veamos ahora cómo se traducen las direcciones virtuales a partir de un valor de dirección virtual y un directorio de páginas.

Es importante notar que vamos a observar la dirección virtual y **dividirla en tres partes**.

- **Los 10 bits más altos** indicarán en qué entrada del directorio vamos a buscar la ubicación de la tabla de páginas necesaria (*pd_index*).

Veamos ahora cómo se traducen las direcciones virtuales a partir de un valor de dirección virtual y un directorio de páginas.

Es importante notar que vamos a observar la dirección virtual y **dividirla en tres partes**.

- **Los 10 bits más altos** indicarán en qué entrada del directorio vamos a buscar la ubicación de la tabla de páginas necesaria (*pd_index*).
- **Los 10 bits siguientes** indicarán en qué entrada de la tabla vamos a buscar la ubicación de la dirección base de la página necesaria (*pt_index*).

Veamos ahora cómo se traducen las direcciones virtuales a partir de un valor de dirección virtual y un directorio de páginas.

Es importante notar que vamos a observar la dirección virtual y **dividirla en tres partes**.

- **Los 10 bits más altos** indicarán en qué entrada del directorio vamos a buscar la ubicación de la tabla de páginas necesaria (*pd_index*).
- **Los 10 bits siguientes** indicarán en qué entrada de la tabla vamos a buscar la ubicación de la dirección base de la página necesaria (*pt_index*).
- **Los 12 bits más bajos** indicarán el desplazamiento desde la base de la página donde se encuentra el dato esperado (*page_offset*).

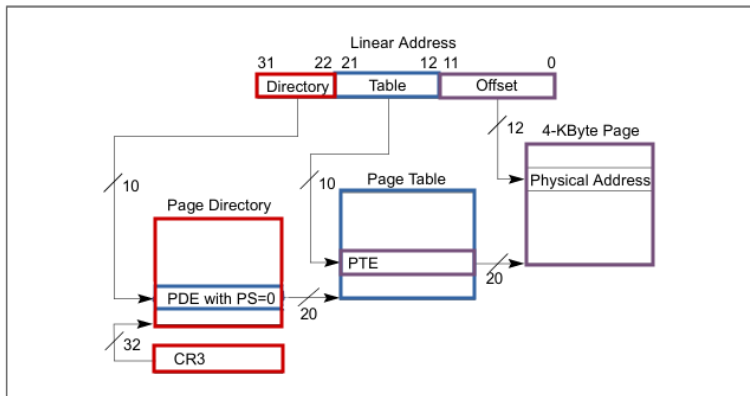


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

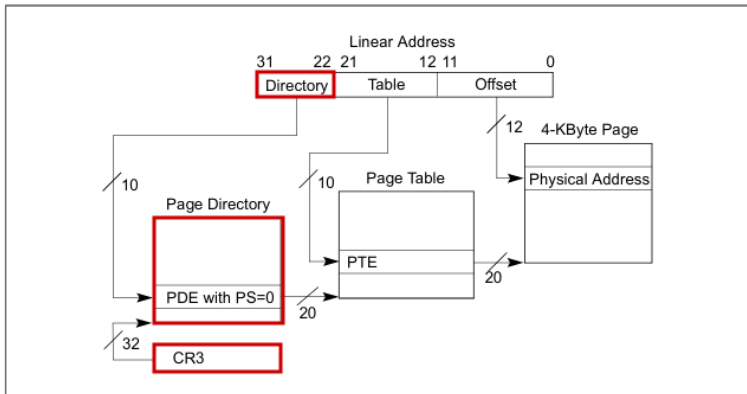


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Veamos primero el CR3.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Address of page directory ¹																				Ignored				P C D	PW T	Ignored				CR3			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page directory ¹																				Ignored					P C D	PW T	Ignored			CR3		

La parte más importante del registro de control CR3 son los 20 bits más altos, ya que contienen la dirección de la página donde se encuentra el directorio a utilizar para traducir las direcciones virtuales.

Dirección del directorio: `CR3 & 0xFFFF000`

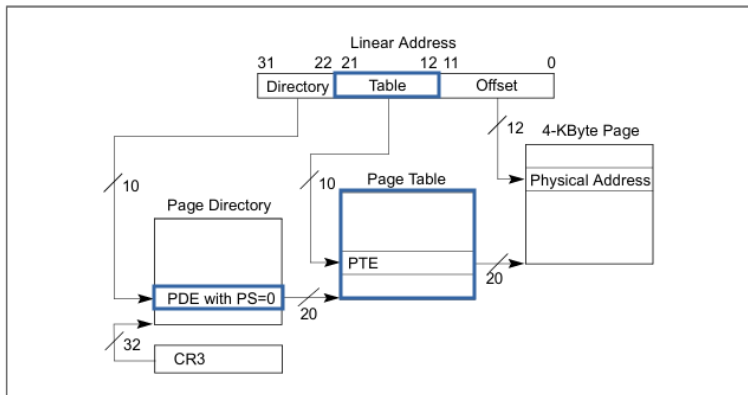


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Ahora veamos las entradas (**PDE** por page directory entry) del directorio de páginas (**PD** por page directory).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page table																				Ignored			<u>0</u>	I g n	A	P C D	P W T	U / S	R / W	<u>1</u>	PDE: page table	

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page table																				Ignored			<u>0</u>	I g n	A	P C D	PW T	U / S	R / W	<u>1</u>	PDE: page table	

Tanto el directorio como la tabla de páginas tendrán 1024 entradas, como cada entrada es de 32 bits (4 bytes) el directorio y las tablas ocupan 1 página (4KByte). Arriba vemos la estructura de una entrada del directorio. Los 20 bits más altos de la i -ésima entrada corresponden a la dirección de la i -ésima tabla de páginas.

Dirección de la i -ésima tabla: $pd[i] \& 0xFFFF000$

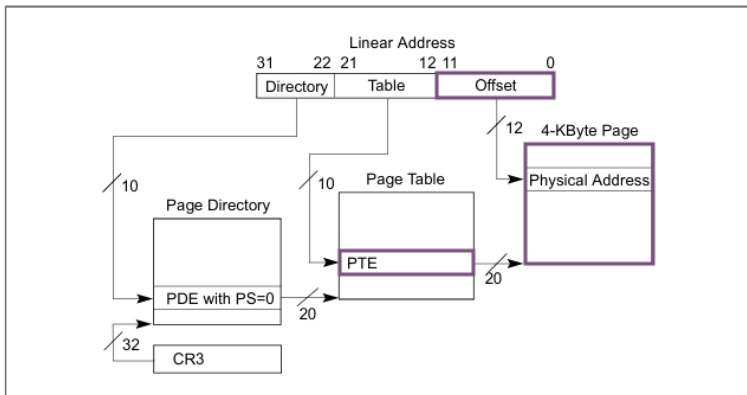


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Ahora veamos la tabla de páginas (**PT** por page table) y sus entradas (**PTE** por page table entry).

Tabla de páginas

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of 4KB page frame																					Ignored	G	P A T	D	A	P C D	P ^W T	U / S	R / W	1	PTE: 4KB page	

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of 4KB page frame																				Ignored	G	P A T	D	A	P C D	P ^W T	U / S	R / W	1		PTE: 4KB page	

Arriba vemos la estructura de una entrada de la tabla. Los 20 bits más altos de la i -ésima entrada corresponden a la dirección de la i -ésima página.

Dirección de la i -ésima página: $pt[i] \& 0xFFFF000$

Queremos traducir la dirección

$$virt = dir(10bits) | table(10bits) | offset(12bits) .$$

Queremos traducir la dirección

$$virt = dir(10bits) | table(10bits) | offset(12bits) .$$

- $pd := CR3 \& 0xFFFF000$ Dirección de PD (limpiamos CR3).

Queremos traducir la dirección

$$virt = dir(10bits) | table(10bits) | offset(12bits) .$$

- $pd := CR3 \& 0xFFFF000$ Dirección de PD (limpiamos CR3).
- $pd_index := (virt \gg 22) \& 0x3FF$ Índice de PD (los 10 bits más altos de $virt$).

Queremos traducir la dirección

$$virt = dir(10bits) | table(10bits) | offset(12bits) .$$

- $pd := CR3 \& 0xFFFF000$ Dirección de PD (limpiamos CR3).
- $pd_index := (virt \gg 22) \& 0x3FF$ Índice de PD (los 10 bits más altos de $virt$).
- $pt := pd[pd_index] \& 0xFFFF000$ Dirección de PT (limpiamos la PDE).

Queremos traducir la dirección

$$virt = dir(10bits) | table(10bits) | offset(12bits) .$$

- $pd := CR3 \& 0xFFFF000$ Dirección de PD (limpiamos CR3).
- $pd_index := (virt \gg 22) \& 0x3FF$ Índice de PD (los 10 bits más altos de $virt$).
- $pt := pd[pd_index] \& 0xFFFF000$ Dirección de PT (limpiamos la PDE).
- $pt_index := (virt \gg 12) \& 0x3FF$ Índice de PT (los 10 bits del medio de $virt$).

Queremos traducir la dirección

$$virt = dir(10bits) | table(10bits) | offset(12bits) .$$

- $pd := CR3 \& 0xFFFF000$ Dirección de PD (limpiamos CR3).
- $pd_index := (virt \gg 22) \& 0x3FF$ Índice de PD (los 10 bits más altos de $virt$).
- $pt := pd[pd_index] \& 0xFFFF000$ Dirección de PT (limpiamos la PDE).
- $pt_index := (virt \gg 12) \& 0x3FF$ Índice de PT (los 10 bits del medio de $virt$).
- $page_addr := pt[pt_index] \& 0xFFFF000$ Dirección de la página (limpiamos la PTE).

Queremos traducir la dirección

$$virt = dir(10bits) | table(10bits) | offset(12bits) .$$

- $pd := CR3 \& 0xFFFF000$ Dirección de PD (limpiamos CR3).
- $pd_index := (virt \gg 22) \& 0x3FF$ Índice de PD (los 10 bits más altos de $virt$).
- $pt := pd[pd_index] \& 0xFFFF000$ Dirección de PT (limpiamos la PDE).
- $pt_index := (virt \gg 12) \& 0x3FF$ Índice de PT (los 10 bits del medio de $virt$).
- $page_addr := pt[pt_index] \& 0xFFFF000$ Dirección de la página (limpiamos la PTE).
- $offset := virt \& 0xFFF$ Offset desde el inicio de la página (los 12 bits más bajos de $virt$).

Queremos traducir la dirección

$$virt = dir(10bits) | table(10bits) | offset(12bits) .$$

- $pd := CR3 \& 0xFFFF000$ Dirección de PD (limpiamos CR3).
- $pd_index := (virt \gg 22) \& 0x3FF$ Índice de PD (los 10 bits más altos de $virt$).
- $pt := pd[pd_index] \& 0xFFFF000$ Dirección de PT (limpiamos la PDE).
- $pt_index := (virt \gg 12) \& 0x3FF$ Índice de PT (los 10 bits del medio de $virt$).
- $page_addr := pt[pt_index] \& 0xFFFF000$ Dirección de la página (limpiamos la PTE).
- $offset := virt \& 0xFFF$ Offset desde el inicio de la página (los 12 bits más bajos de $virt$).
- $phys := page_addr | offset$ Dirección física (sumamos la base de la página y el offset de $virt$).

Taller

En el taller van a tener que implementar algunas funciones relacionadas con la paginación y la unidad de manejo de memoria.

```
paddr_t mmu_next_free_kernel_page(void);  
paddr_t mmu_next_free_user_page(void);
```

- Devuelve la dirección física de la próxima página de kernel/user disponible.
- Las páginas de kernel y de usuario se encuentran en rangos de direcciones distintos, por lo que es necesario llevar un contador para cada uno.

```
paddr_t mmu_init_kernel_dir(void);
```

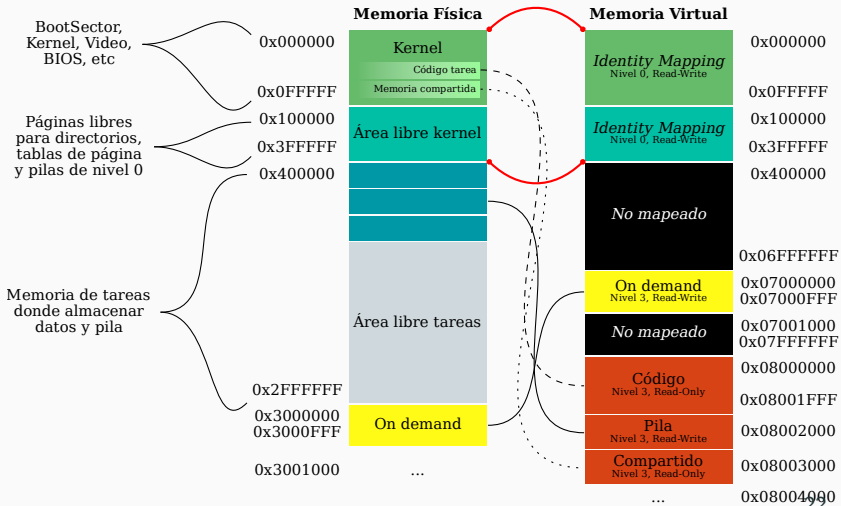
- Genera el identity mapping de las primeras 4MB de memoria.
- Recuerden setear correctamente los atributos de las páginas (W, S, P).
- ¿Cuántas páginas necesito para armar las tablas?

```
void mmu_map_page(uint32_t cr3, vaddr_t virt, paddr_t phy,  
↪ uint32_t attrs);
```

- Genera un mapeo de una página virtual a una página física.
- ¿en qué directorio? ¿Y si ya existe la page table?
- ¿Los atributos van para la PDE o la PTE?
- Recuerden llamar a `tlbflush`

```
paddr_t mmu_unmap_page(uint32_t cr3, vaddr_t virt);
```

- Desmapea una página virtual y devuelve la dirección de la página física desmapeada
- Pueden preguntar por Present y actuar en consecuencia
- Recuerden llamar a `tlbflush`

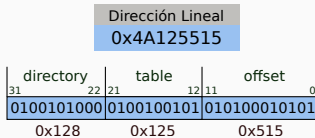


```
paddr_t mmu_init_task_dir(paddr_t phy_start)
```

- Genera el mapeo estático de una tarea del sistema
- Mapea 2 páginas de código, una de pila y una shared
- Recuerden mapear al kernel en los primeros 4MB

Dirección Lineal

0x4A125515



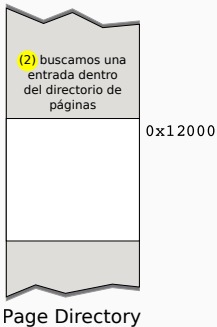
Dirección Lineal		
0x4A125515		
directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515



Dirección Lineal		
0x4A125515		
directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515



Dirección Lineal 0x4A125515		
directory 31 22	table 21 12	offset 11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515



Dirección Lineal

0x4A125515

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(1) buscar el CR3

CR3 00012000

(2) buscamos una entrada dentro del directorio de páginas

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(3) decodificamos la PDE

0x12000

0x12000 + 0x128 * 4

Page Directory

Dirección Lineal

0x4A125515

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(1) buscar el CR3

CR3 00012000

(2) buscamos una entrada dentro del directorio de páginas

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(3) decodificamos la PDE

0x12000

0x12000 + 0x128 * 4

Page Directory

Dirección Lineal

0x4A125515

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(1) buscar el CR3

CR3 00012000

(2) buscamos una entrada dentro del directorio de páginas

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(3) decodificamos la PDE

0x12000

0x12000 + 0x128 * 4

(4) buscamos una entrada dentro del page table

0x23000

Page Directory

Page Table

Dirección Lineal

0x4A125515

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(1) buscar el CR3

CR3 00012000

(2) buscamos una entrada dentro del directorio de páginas

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(3) decodificamos la PDE

0x12000

00023003 $0x12000 + 0x128 * 4$

(4) buscamos una entrada dentro del page table

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(5) decodificamos la PTE

0x23000

00002003 $0x23000 + 0x125 * 4$

Page Directory

Page Table

Dirección Lineal

0x4A125515

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(1) buscar el CR3

CR3 00012000

(2) buscamos una entrada dentro del directorio de páginas

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(3) decodificamos la PDE

0x12000

00023003 0x12000+0x128*4

(4) buscamos una entrada dentro del page table

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(5) decodificamos la PTE

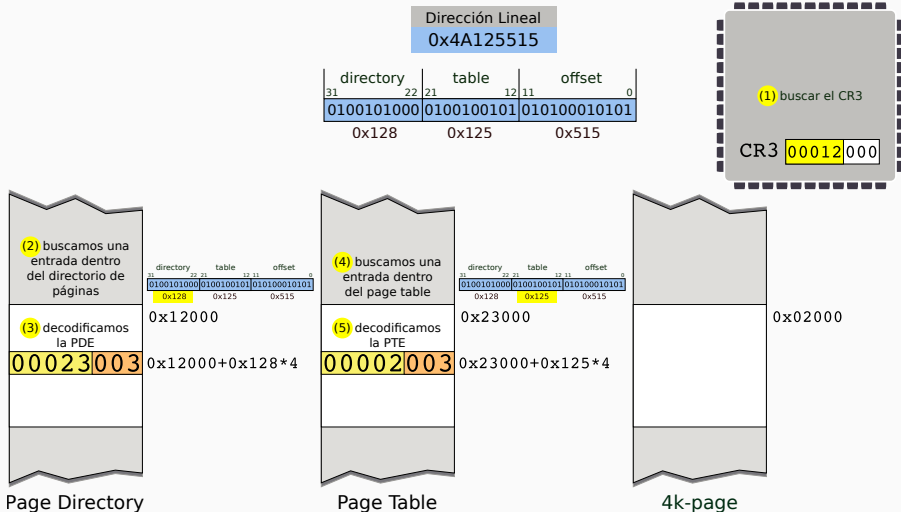
0x23000

00002003 0x23000+0x125*4

Page Directory

Page Table

Ejemplo Numérico



Dirección Lineal

0x4A125515

directory	table	offset
31 22 21	12 11	0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(1) buscar el CR3

CR3 00012000

(2) buscamos una entrada dentro del directorio de páginas

directory	table	offset
31 22 21	12 11	0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(3) decodificamos la PDE

0x12000

$0x12000 + 0x128 * 4$

(4) buscamos una entrada dentro del page table

directory	table	offset
31 22 21	12 11	0
0100101000	0100100101	010100010101
0x128	0x125	0x515

(5) decodificamos la PTE

0x23000

$0x23000 + 0x125 * 4$

directory	table	offset
31 22 21	12 11	0
0100101000	0100100101	010100010101
0x128	0x125	0x515

0x02000

$0x02000 + 0x515$

(6) ¡al fin!
ahora buscamos la entrada de memoria que queremos

4k-page

Page Directory

Page Table

Recuerden que el procesador cuenta con una tabla de traducciones pre-computadas a la que llamaremos buffer auxiliar de traducción o **translation lookaside buffer**, que básicamente almacena las últimas traducciones realizadas para no tener que volver a computarlas.

Recuerden que el procesador cuenta con una tabla de traducciones pre-computadas a la que llamaremos buffer auxiliar de traducción o **translation lookaside buffer**, que básicamente almacena las últimas traducciones realizadas para no tener que volver a computarlas.

Cuando realicemos un cambio en nuestras estructuras de paginación es necesario forzar una limpieza del mismo para evitar que las direcciones pre-computadas que ya no son válidas se sigan empleando, para esto realizamos un intercambio del registro CR3 con un valor temporal y luego lo restauramos.

Veamos qué atributos se encuentran disponibles en las entradas de la tabla de páginas para comprender cómo se implementan algunos mecanismos que presentaremos en breve.

Veamos qué atributos se encuentran disponibles en las entradas de la tabla de páginas para comprender cómo se implementan algunos mecanismos que presentaremos en breve.

Recuerden que a diferencia de lo que sucede con segmentación en paginación sólo tenemos **dos niveles de privilegio, supervisor o kernel (0) y usuario (1)**.

- El bit **D**(dirty) es seteado por unidad de memoria del procesador cuando se escribe en la página, se limpia por soft.

- El bit **D**(dirty) es seteado por unidad de memoria del procesador cuando se escribe en la página, se limpia por soft.
- El bit **A**(accessed) es seteado por unidad de memoria del procesador cuando se escribe o lee en la página, se limpia por soft.

- El bit **D**(dirty) es seteado por unidad de memoria del procesador cuando se escribe en la página, se limpia por soft.
- El bit **A**(accessed) es seteado por unidad de memoria del procesador cuando se escribe o lee en la página, se limpia por soft.
- El **PCD** (cache disabled) hace que la página no se almacene en memoria rápida.

- El bit **D**(dirty) es seteado por unidad de memoria del procesador cuando se escribe en la página, se limpia por soft.
- El bit **A**(accessed) es seteado por unidad de memoria del procesador cuando se escribe o lee en la página, se limpia por soft.
- El **PCD** (cache disabled) hace que la página no se almacene en memoria rápida.
- **PWT** (write through) hace que al escribir la escritura se refleje en cache y memoria a la vez, sino sólo se actualiza la memoria cuando se desaloja la línea de caché.

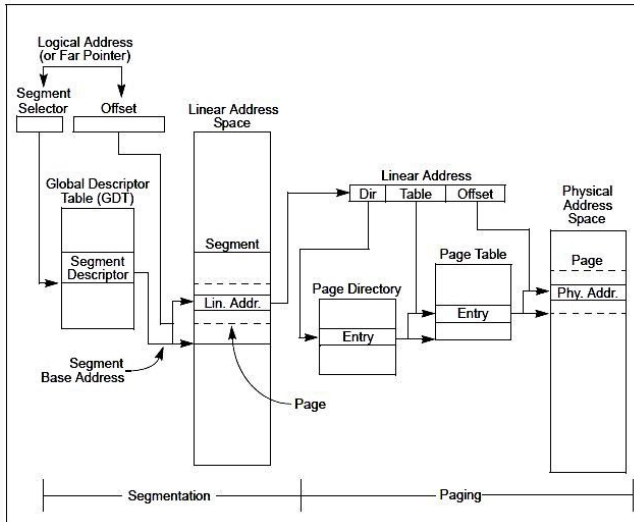
- **U/S** indica si la página puede ser accedida por el espacio de usuario (bit en 1) o sólo supervisor/kernel (bit en 0).

- **U/S** indica si la página puede ser accedida por el espacio de usuario (bit en 1) o sólo supervisor/kernel (bit en 0).
- **R/W** indica si la página puede leerse y escribirse (bit en 1) o sólo leerse (bit en 0).

- **U/S** indica si la página puede ser accedida por el espacio de usuario (bit en 1) o sólo supervisor/kernel (bit en 0).
- **R/W** indica si la página puede leerse y escribirse (bit en 1) o sólo leerse (bit en 0).
- **P** indica si la página se encuentra cargada en memoria o no.

Sólo para entender la interacción de las dos unidades, veamos como funciona la traducción al involucrar la unidad de segmentación.

Sólo para entender la interacción de las dos unidades, veamos como funciona la traducción al involucrar la unidad de segmentación. Recuerden que al usar segmentación flat podemos olvidarnos de la unidad de segmentación al pensar la traducción de las direcciones.



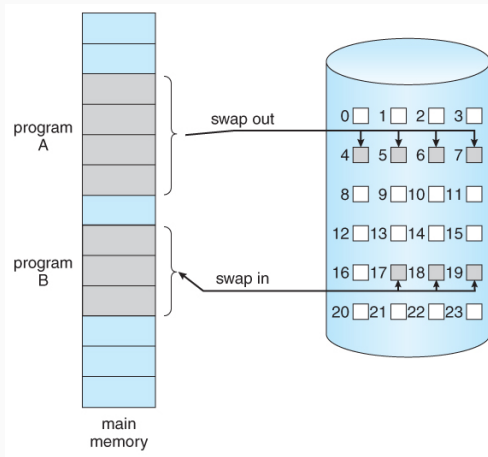
Casos de uso

Veamos ahora algunos ejemplos de cómo el mecanismo de paginación nos permite implementar soluciones flexibles para diversos problemas.

Veamos ahora algunos ejemplos de cómo el mecanismo de paginación nos permite implementar soluciones flexibles para diversos problemas.

Estos son también buenos ejemplos de cómo las excepciones pueden utilizarse no sólo para detectar comportamiento patológico sino para disparar respuestas del sistema operativo frente a ciertas situaciones esperadas en el curso de su funcionamiento.

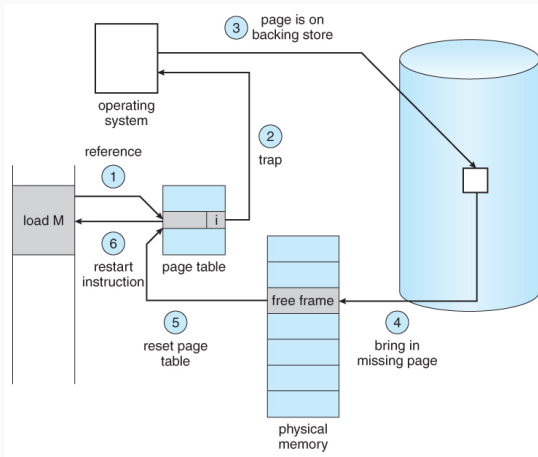
Cuando varios procesos comparten el uso de la memoria principal y la misma se encuentra al límite de la sobre-utilización o si un proceso se encuentra suspendido mientras otro precisa ejecutarse, **el sistema operativo puede decidir mover las páginas de estos procesos desde y hacia el disco sin afectar la visión del proceso de su espacio de memoria**, incluso si al volver a volcarlo en memoria principal cambian sus direcciones físicas.



Intercambio de procesos en memoria principal.

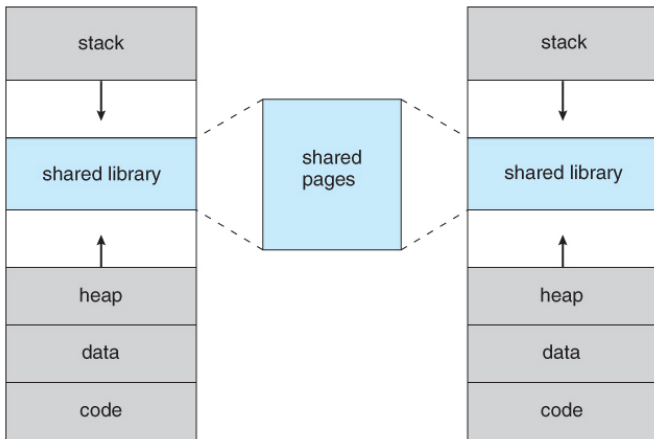
Para implementar efectivamente este mecanismo el sistema operativo puede registrar internamente la frecuencia y prioridad de los procesos para decidir cuáles van o vuelven del disco.

El sistema operativo puede decidir no cargar una parte de las páginas asociadas a un proceso y marcar sus entradas como no presentes (bit **P** en 0), y luego en la rutina de atención de la excepción del **page fault** buscar la página en disco, mapearla en la tabla correspondiente, marcar el bit de presente en 1 y volver a ejecutar.



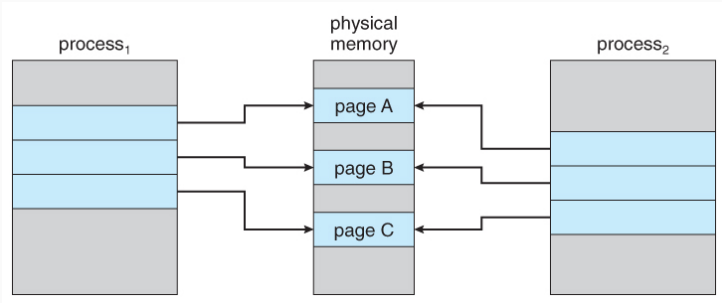
Un **page fault** no siempre es malo.

Hablamos anteriormente de las **bibliotecas compartidas**, que son utilidades que el sistema operativo expone a uno o varios procesos como forma de código binario que puede accederse dinámicamente. Si varios procesos acceden al mismo código no hace falta copiarlo una vez por cada instancia, podemos mapear las mismas páginas físicas (de la biblioteca compartida) en distintas estructuras por proceso.

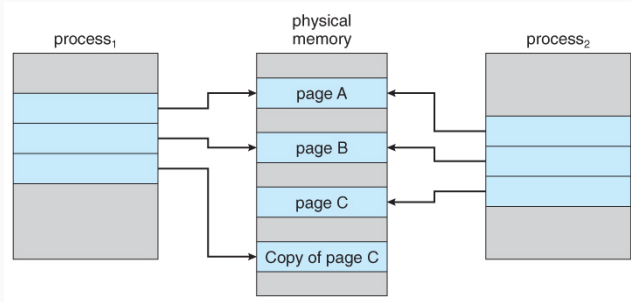


Un mismo rango físico mapeado en distintos rangos virtuales.

En los sistemas operativos modernos es habitual que se implemente la creación de nuevos procesos a partir de la copia del proceso en ejecución (**fork**), frente a esta situación se pueden mapear varios procesos contra el mismo conjunto de páginas físicas que contienen su código y sólo duplicarlas si se escribe en ellas y más de un proceso las está accediendo. Para esto se pueden marcar las páginas como de sólo escritura, y al detectar la excepción copia la página, la mapea como de escritura para el proceso que generó la interrupción y vuelve a ejecutar.



Varios procesos resultantes de un **fork** pueden compartir páginas físicas siempre y cuando no escriban en ellas.



Cuando un proceso escribe en una página compartida es necesario copiarla y modificar las estructuras de paginación acorde a esto.

Estos son algunos casos de uso interesantes que involucran el mecanismo de **interrupciones** y el de **paginación**.

Cierre

En la introducción de hoy vimos:

En la introducción de hoy vimos:

- **Repaso de paginación como mecanismo**

En la introducción de hoy vimos:

- **Repaso de paginación como mecanismo**
- **Proceso de traducción de direcciones virtuales a direcciones físicas**

En la introducción de hoy vimos:

- Repaso de paginación como mecanismo
- Proceso de traducción de direcciones virtuales a direcciones físicas
- Estructuras involucradas en paginación (CR3, `page directory`, `page table`)

En la introducción de hoy vimos:

- Repaso de paginación como mecanismo
- Proceso de traducción de direcciones virtuales a direcciones físicas
- Estructuras involucradas en paginación (CR3, page directory, page table)
- Atributos relevantes

En la introducción de hoy vimos:

- Repaso de paginación como mecanismo
- Proceso de traducción de direcciones virtuales a direcciones físicas
- Estructuras involucradas en paginación (CR3, `page directory`, `page table`)
- Atributos relevantes
- Casos de uso para el mecanismo de paginación

Consultas y ejercitación
