



# SISTEMAS OPERATIVOS

*Grado en Desarrollo de Videojuegos  
Universidad Complutense de Madrid*

## TEMA 4. Memoria

# Tema 4. Memoria

---

## 4.1 Fundamentos de la Memoria Virtual

- Objetivos
- Espacio de direcciones
- Mecanismos de traducción: Memoria Virtual Paginada

## 4.2 Mapa de Memoria de un Proceso

- Segmentos de Memoria
- Interfaz del Sistema

## 4.3 Caché de Páginas

- Objetivos
- Relación con el VFS y el Sistema de Memoria Virtual

## 4.4 Gestión de la Memoria Principal

- Políticas de Asignación
- Políticas de Reemplazamiento
- Buffering the páginas



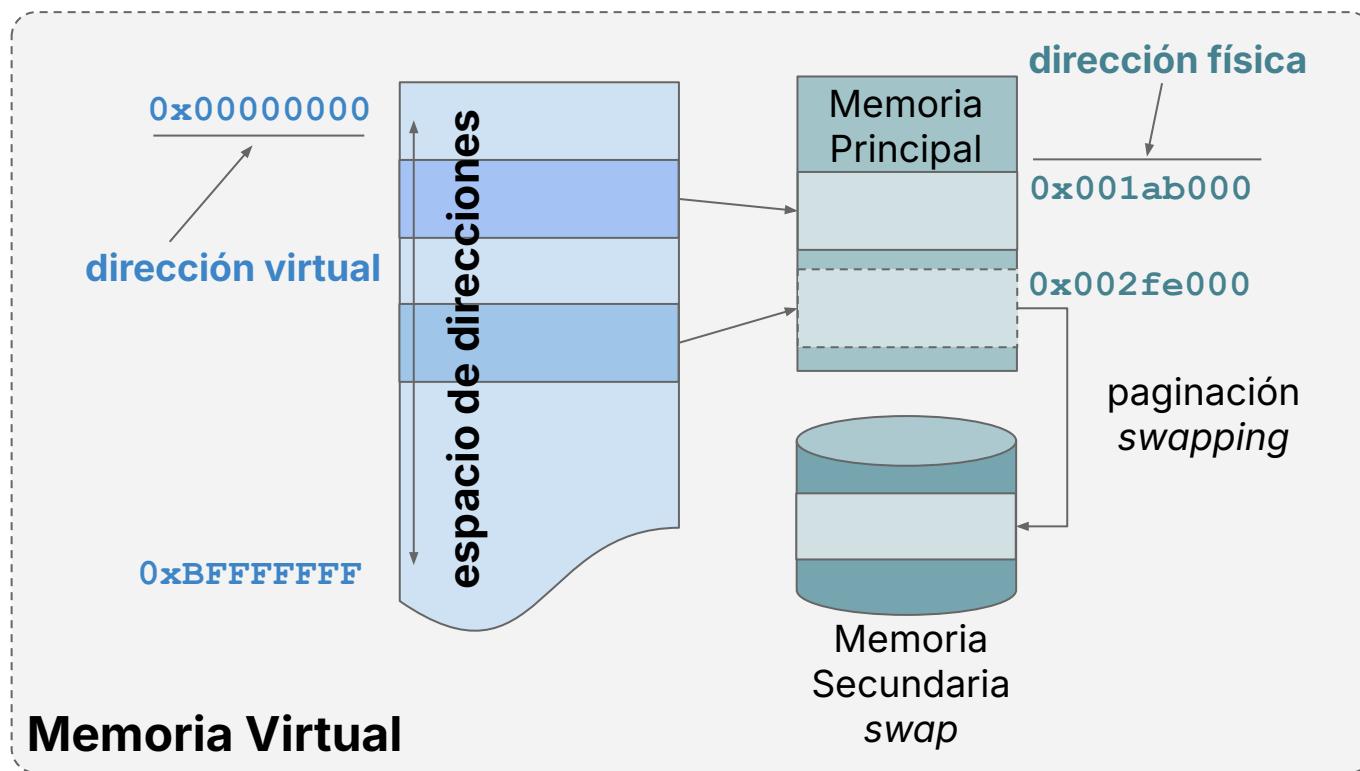
# SISTEMAS OPERATIVOS

*Grado en Desarrollo de Videojuegos  
Universidad Complutense de Madrid*

## TEMA 4.1 Fundamentos de la Memoria Virtual

# Memoria Virtual (I)

- La **memoria virtual** es una abstracción que proporciona a los procesos una memoria (**espacio de direcciones**) **lineal, grande y privado**.
- El sistema gestiona el intercambio de información entre la memoria principal (física) y la secundaria (swap, en disco)



# Memoria Virtual (II)

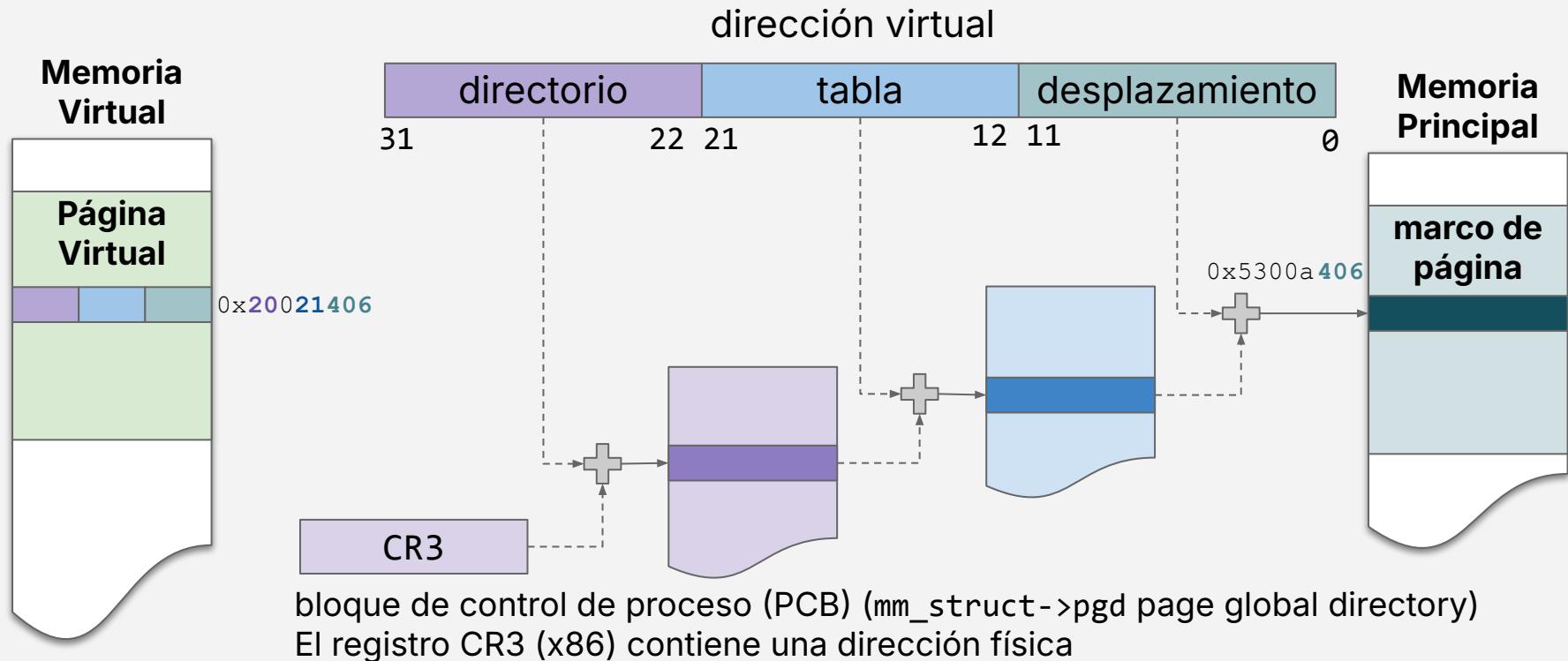
## Objetivos de la Memoria Virtual

- **Sobresuscripción.** Memoria principal *aparente* mayor que la disponible
  - Ejecutar procesos que usan (individualmente o de forma agregada) más memoria de la disponible
- **Multiprogramación.** Ejecución aislada de varios procesos (*y el kernel*)
  - El espacio de direcciones de cada proceso está separado
  - Los procesos solo pueden acceder a las regiones asignadas por el SO
  - Debe permitir que los procesos comparten regiones de memoria (ej. librerías compartidas)
- **Reubicación.** Simplifica el desarrollo de software permitiendo que los programas se pueden cargar en cualquier posición
  - No es necesario recompilar o modificar el código para diferentes perfiles de memoria
  - Varios procesos pueden coexistir sin colisionar

# Memoria Virtual Paginada (I)

- La **memoria virtual** se divide en secciones de tamaño fijo (**página**)
- El **rango contiguo** de direcciones virtuales de la página se traduce en un rango contiguo de direcciones físicas del mismo tamaño (**marco de página**).
- **Tabla de páginas** estructura que contiene la asignación de direcciones virtuales a físicas del proceso. Tiene **varios niveles de traducción** para ahorrar tamaño

Ejemplo: Arquitectura x86 de 32 bits. Dos niveles de traducción Directorio y Tabla de páginas



# Memoria Virtual Paginada (II)

## Entradas de la tabla de páginas

Contenidos para x86 ([asm/pgtable\\_types.h](#))<sup>\*</sup>

- **Dirección del marco de página** de memoria (o del siguiente nivel)
- **Presencia (P)**: si la página está en memoria principal
- **Modificada (D)**: si los contenidos de la página han cambiado
- **Protección**: indica si la página se puede escribir (RW) y ejecutar ([NX, No eXecute](#))
- **Acceso (A)**: la página ha sido usada (política de reemplazamiento)
- **Usuario (U/S)**: Nivel de privilegio para acceder a la página (usuario/supervisor)
- **Caché**: Si la página almacena en la caché de la CPU (CD, Cache Disable) y si se activa el modo *write-through* (WT)

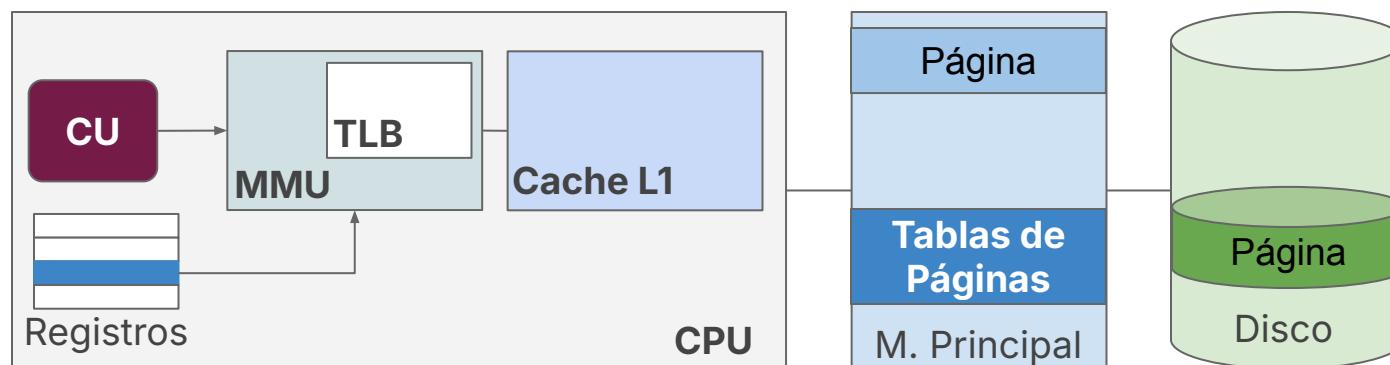
| Dirección Marco (20 bits) | - | 6 | 5 | 4  | 3  | 2   | 1  | 0 |
|---------------------------|---|---|---|----|----|-----|----|---|
|                           |   | D | A | CD | WT | U/S | RW | P |

<sup>\*</sup>**Nota:** El contenido de las tablas de páginas depende de la arquitectura

# Memoria Virtual Paginada (III)

La traducción de las direcciones virtuales se realiza con el soporte de la **unidad de gestión de memoria** (*Memory Management Unit, MMU*):

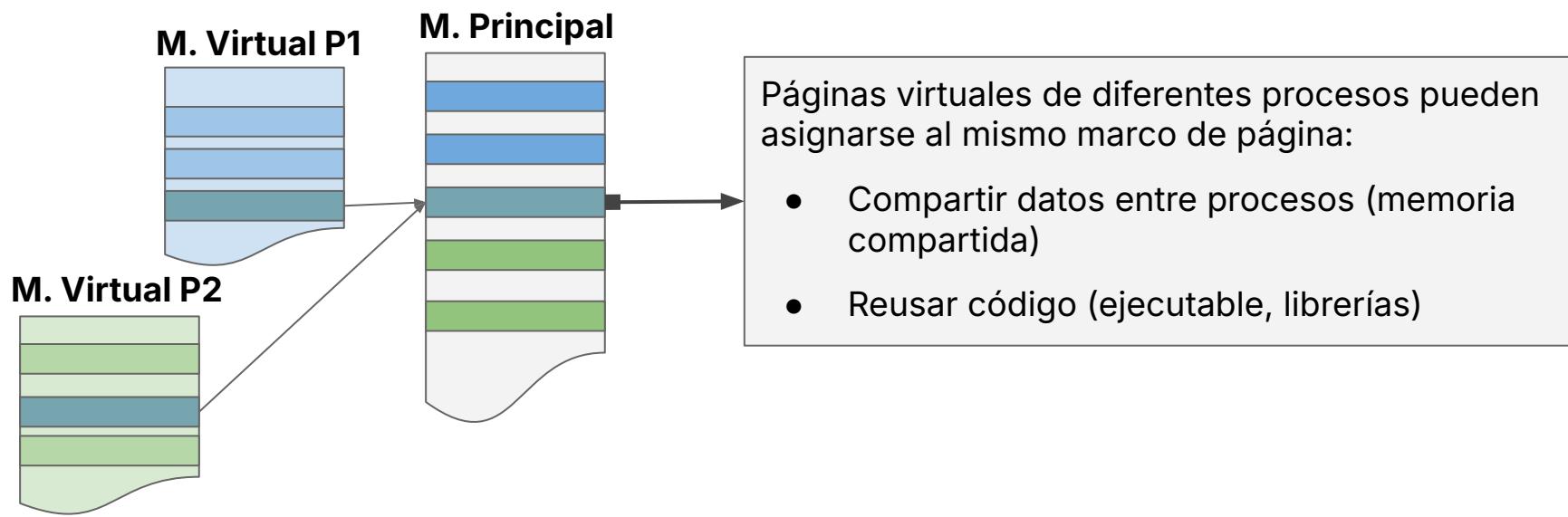
- Realiza las **indirecciones necesarias** en los niveles de paginación.
- Comprueba, en cada nivel, la **presencia** y **modos de acceso**. En caso de fallo de página o acceso no permitido genera una excepción.
- **Caché** para acelerar la traducción (*Translation Lookaside Buffer, TLB*)



# Memoria Virtual Paginada (IV)

El **Sistema Operativo** se encarga de:

- Gestionar las **excepciones** generadas por la MMU
  - Asignar **nuevas páginas** de memoria. Las páginas se reservan bajo demanda (*demand paging*) cuando se leen o escriben por primera vez.
  - **Traer** la página de la **memoria secundaria**
  - Terminar el proceso (**SIGSEV**). Acceso no válido (permiso) a memoria.
- Mantener las **tablas de páginas** actualizadas
- **Configurar la MMU** para cada proceso (ej. CR3 o limpiar la TLB)
- Configurar el **acceso compartido** a páginas de memoria



# Memoria Virtual Paginada (V)

## Tamaño de Página

- **Fragmentación Interna.** Páginas más pequeñas producen menor fragmentación (memoria asignada > memoria requerida)
- **Número de Páginas.** Páginas más pequeñas producen más páginas de memoria que gestionar:
  - Tablas de páginas mayores.
  - Menor efectividad de la TLB
- **Transferencias con Memoria Secundaria.** Las trasferencia de páginas pequeñas es menos eficiente (requiere potencialmente más operaciones e interrupciones)

## Tamaño de Página en Linux (x86)

- Tamaño de página por defecto es de 4KB (comando, `getconf PAGE_SIZE`)
- **Transparent Huge Pages.** Combina páginas de 4KB en páginas de 2MB (512 páginas contiguas).
  - Comportamiento por defecto (configurable con `madvise(2) - MADV_HUGEPAGE`)
  - `khugepaged` proceso del kernel que trata de combinar las páginas
- **Huge Pages.** Linux reserva un pool de páginas de 2MB ó 1GB (CPU pdpe1gb) que las aplicaciones pueden solicitar (p.ej. máquinas virtuales, bases de datos...)

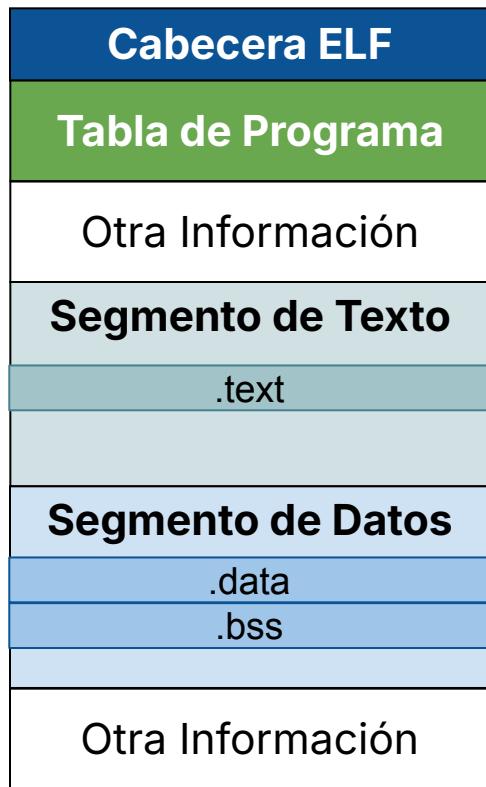


## **TEMA 4.2 Mapa de Memoria de un Proceso**

# Mapa de Memoria de un Proceso(I)

Un **programa** es un conjunto de instrucciones máquina y datos, almacenados en una **imagen ejecutable en disco** (entidad pasiva)

*ELF: Executable & Linking Format*



Información para cargar las diferentes partes (secciones) del programa en memoria.

| Tipo | Offset  | Tamaño | Dir. Virtual | Tamaño | Flags |
|------|---------|--------|--------------|--------|-------|
| LOAD | 0x02dd0 | 0x0258 | 0x03dd0      | 0x0268 | R W - |

Secciones: .data, .bss, ...

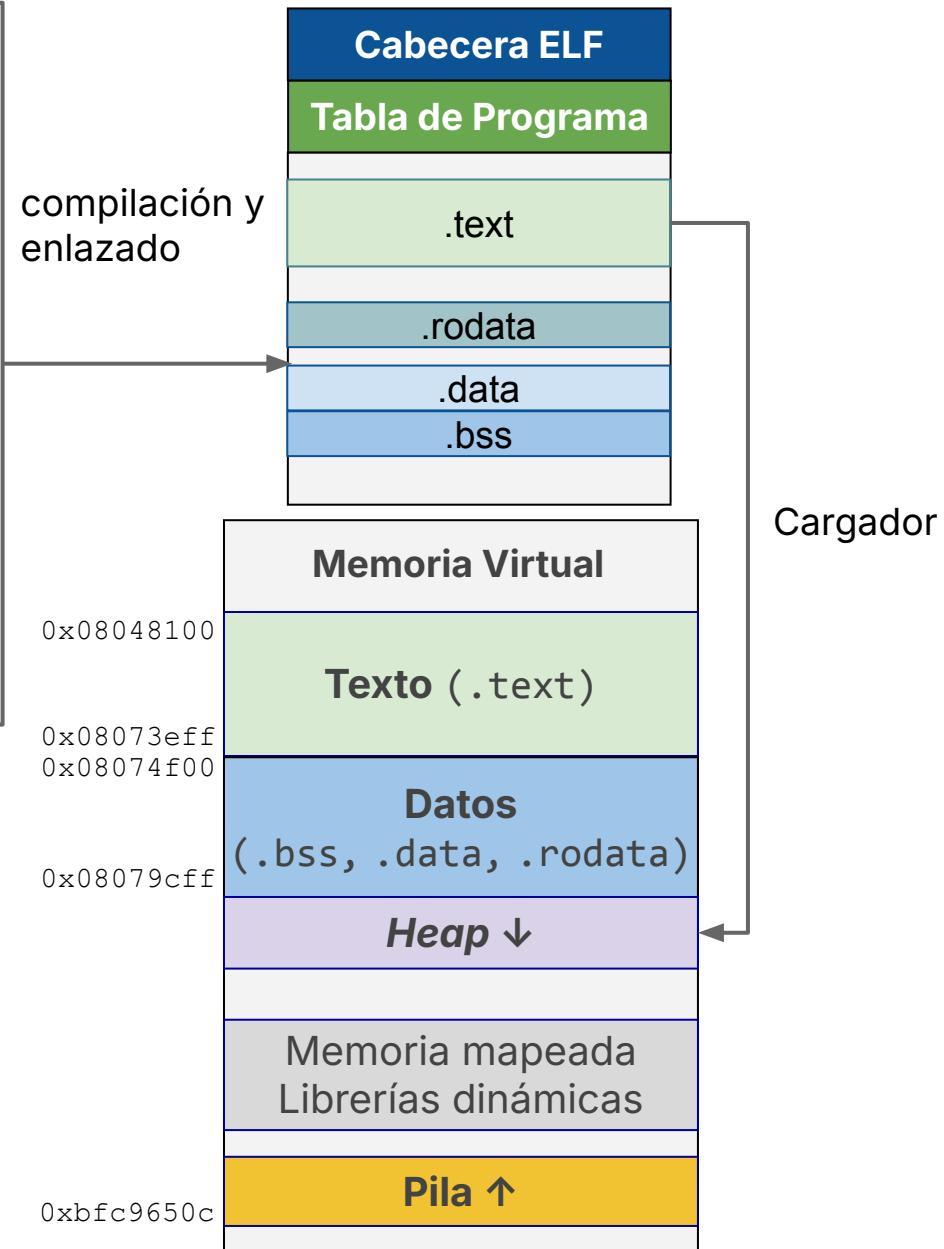
Algunas secciones importantes de un ejecutable:

**Sección Propósito**

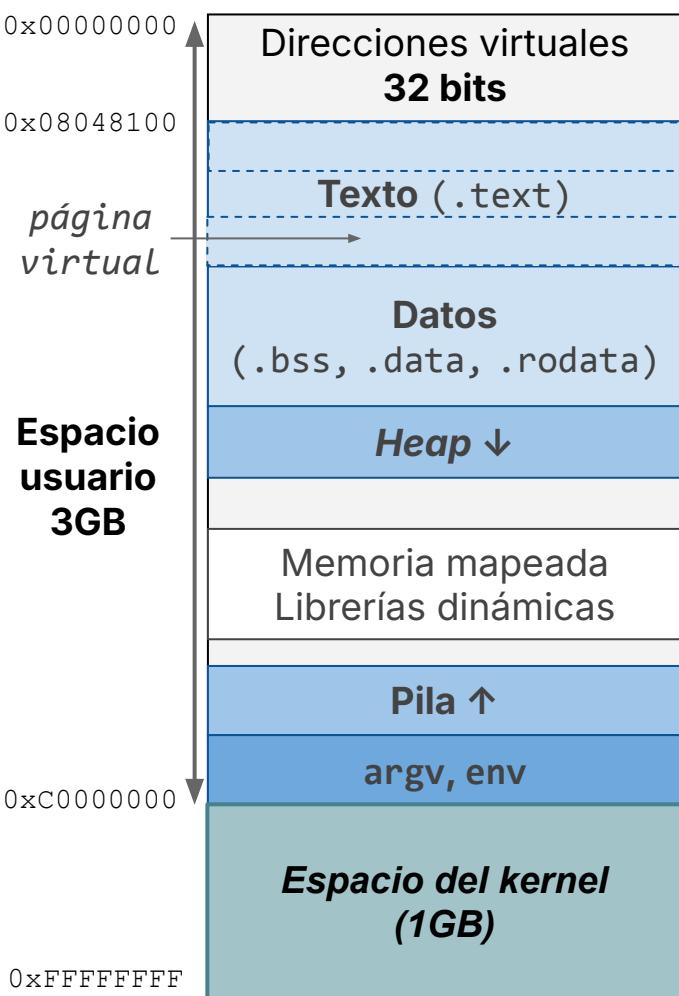
|         |  |
|---------|--|
| .text   | Instrucciones                                |
| .rodata | Constantes (p.ej. cadenas literales)         |
| .data   | Variables globales o static, inicializadas   |
| .bss    | Variables globales o static, sin inicializar |

# Mapa de Memoria de un Proceso(I)

```
int numero = 21; // .data  
  
int resultado; // .bss  
  
const char *msg = "Resultado:\n"; // .rodata  
  
int main(void) {  
    static int factor = 2; // .data  
  
    // .text  
    resultado = numero * factor;  
    printf("%s", msg);  
    return 0;  
}
```



# Mapa de Memoria de un Proceso(I)

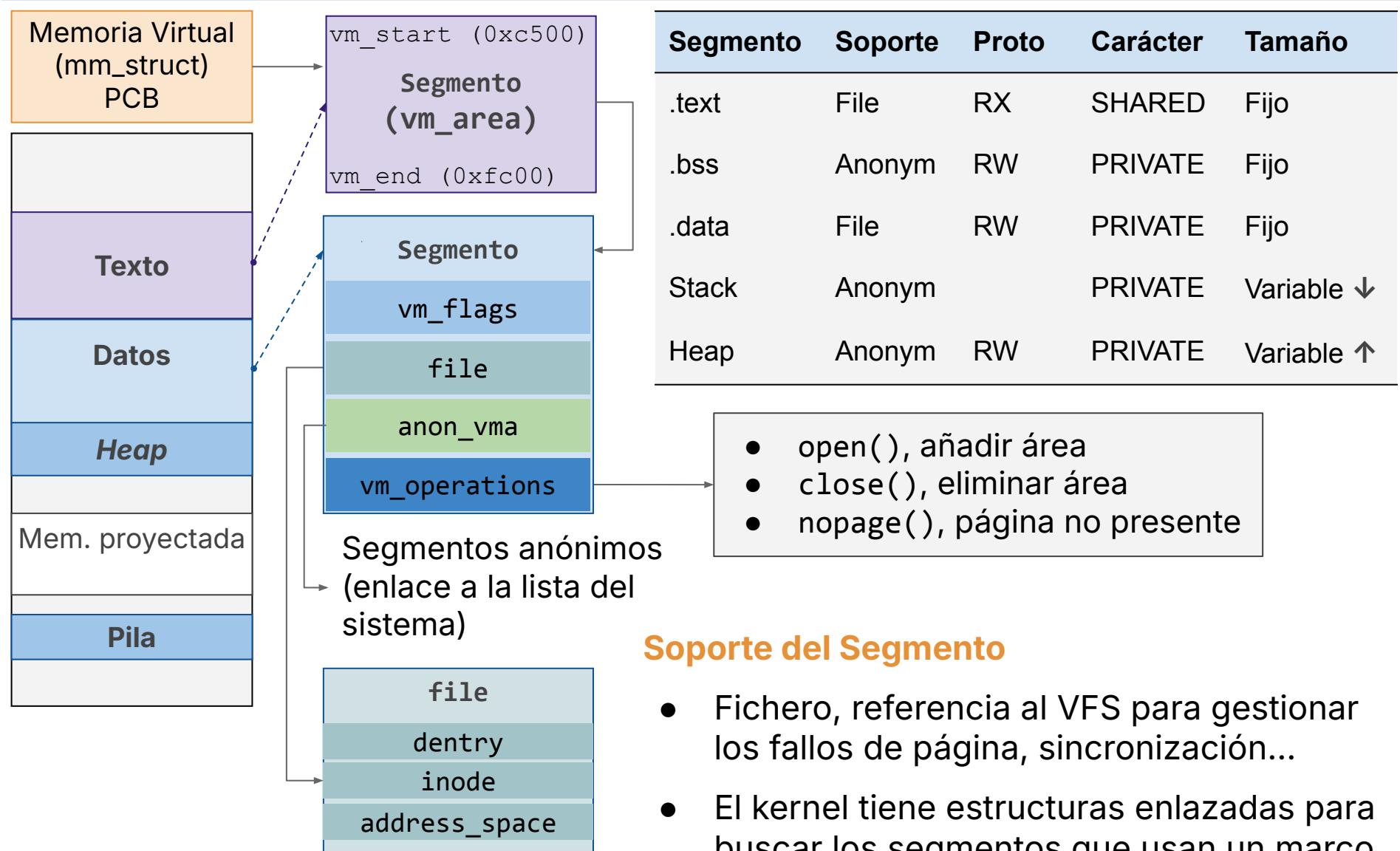


## Segmentos de Memoria (maps)

- Área de memoria contigua con varias páginas de memoria virtual (`vm_area_struct`)
- Las páginas tienen los mismos permisos (RW, NX) y son del mismo tipo
- Tipos:
  - Respaldadas por fichero
  - Anónimas (heap, stack, memoria dinámica)
  - Privadas accesibles únicamente por el proceso
  - Compartidas accesibles por otros procesos.

- Código y estructuras de datos del kernel. Compartido por todos los procesos
- Sólo accesible en modo privilegiado (bit U/S de la tabla de páginas)
- Tablas de páginas separadas (usuario,kernel) - Meltdown - [Kernel Page Table Isolation \(KPTI\)](#)

# Mapa de Memoria de un Proceso(II)



# Mapa de Memoria de un Proceso(III)

## Interfaz del Sistema (CLI)

- Los segmentos de un proceso se pueden consultar en `/proc/<pid>/maps`. Muestra el contenido de la lista de `vm_area_struct`.
- El comando `ps` puede mostrar el uso de memoria del proceso:
  - **RSS (Resident Set Size)** KB. Páginas en memoria principal. No incluye memoria secundaria o sin inicializar. Las páginas compartidas se cuentan para cada proceso que las referencia.
  - **VSZ (Virtual Memory Size)** KB. Memoria virtual del proceso incluye librerías dinámicas y ficheros proyectados en memoria.

## Ejemplo: Segmentos de memoria virtual de un proceso

| <code>vm_start</code> | - | <code>vm_end</code> | <code>flag</code> | <code>offset</code> | <code>ma:mi</code> | <code>inode</code> | <code>respaldo (path)</code> |
|-----------------------|---|---------------------|-------------------|---------------------|--------------------|--------------------|------------------------------|
| 557bfb1c5000          | - | 557bfb1fc000        | r--p              | 00000000            | fe:02              | 56408195           | /usr/bin/bash                |
| 557bfb1fc000          | - | 557bfb2ce000        | r-xp              | 00037000            | fe:02              | 56408195           | /usr/bin/bash                |
| 557bfb2ce000          | - | 557bfb312000        | r--p              | 00109000            | fe:02              | 56408195           | /usr/bin/bash                |
| 557bfb312000          | - | 557bfb318000        | r--p              | 0014d000            | fe:02              | 56408195           | /usr/bin/bash                |
| 557bfb318000          | - | 557bfb321000        | rw-p              | 00153000            | fe:02              | 56408195           | /usr/bin/bash                |
| 557bfb321000          | - | 557bfb32c000        | rw-p              | 00000000            | 00:00              | 0                  |                              |
| 557c11c2b000          | - | 557c11e08000        | rw-p              | 00000000            | 00:00              | 0                  | [heap]                       |

→ Segmento .text con código de bash

→ Segmentos anónimos

# Mapa de Memoria de un Proceso(IV)

## Interfaz del Sistema (API)

- Llamadas que modifican la memoria virtual de un proceso
  - **fork(2)**. Crea un nuevo proceso copiando las páginas de memoria virtual del padre (Copy-On-Write)
  - **execve(2)**. Carga un nuevo ejecutable reemplazando el espacio de direcciones
  - **brk(2)**. Mueve la localización del *program break* (justo al final del segmento de datos), aumentando/disminuyendo el heap del proceso.
  - **mmap(2)/munmap(2)/mremap(2)**. Crea/destruye/redimensiona un segmento de diferentes tipos.

**Nota:** `malloc(3)` internamente usa `brk(2)` y `mmap(2)` según la cantidad de memoria solicitada (`MMAP_THRESHOLD`).

# Mapa de Memoria de un Proceso(V)

## Interfaz del Sistema (API)

```
void *mmap(void *addr,  
           size_t length,  
           int prot,          // PROT_EXEC, PROT_READ, PROT_WRITE  
           int flags,         // MAP_SHARED, MAP_PRIVATE, MAP_ANON  
           int fd,            // Proyección de fichero  
           off_t offset);  
  
int munmap (void * addr , size_t len);
```

- `addr, len`: indicación de la dirección de memoria virtual de comienzo del segmento de `len` bytes. Normalmente `NULL`. El kernel elige la dirección alineada al tamaño de página.
  - `munmap`, elimina las regiones mapeadas entre `addr` y `addr + len`
- `MAP_SHARED` el segmento es visible a otros procesos y se actualiza el fichero de respaldo (si lo hay). `MAP_PRIVATE` tiene el comportamiento opuesto
- `MAP_ANON`. Región de memoria sin fichero de respaldo
- `fd, offset`. La región contiene los datos del fichero a proyectar. `prot` tiene que ser consistente con el modo de apertura del fichero.
- `MAP_HUGETLB, MAP_HUGE_2MB, MAP_HUGE_1GB` (Linux). Selecciona hugepages para el segmento

# Mapa de Memoria de un Proceso(VI)

## Interfaz del Sistema (API)

```
int msync (void * addr , size_t len , int flags)
```

- Planifica la actualización de los ficheros de respaldo de los segmentos MAP\_SHARED entre addr y addr + len. addr debe estar alineado al tamaño de página.
- Los flags permiten modificar el modo de actualización:
  - MS\_SYNC: bloquea el proceso hasta que se complete la operación
  - MS\_ASYNC: actualiza la caché de páginas y planifica la escritura a disco. Retorna inmediatamente.
  - MS\_INVALIDATE: se invalidan otros segmentos respaldados por el mismo fichero para que se carguen de nuevo en el siguiente acceso.
- **Nota:** Debe sincronizarse las páginas respaldadas por fichero y modificadas antes de llamar a la función munmap(2).



## TEMA 4.3 Caché de Páginas

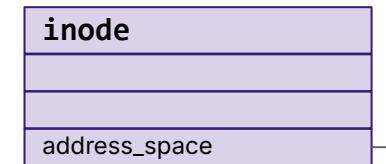
# Caché de Páginas (I)

- Mecanismo para **acelerar las lecturas y escrituras** de disco:
  - **E/S de ficheros** (`read(2), write(2)`)
  - **E/S** dispositivos en **modo bloque** (originalmente *buffer cache*)
- Integrado con todo el sistema de **memoria virtual**:
  - Proyección de ficheros (`mmap(2)`)
  - **Páginas anónimas** en el área de intercambio (swap)
  - Sistemas de ficheros en memoria, pipes...
- Permite usar **políticas comunes** para la **gestión de memoria principal**

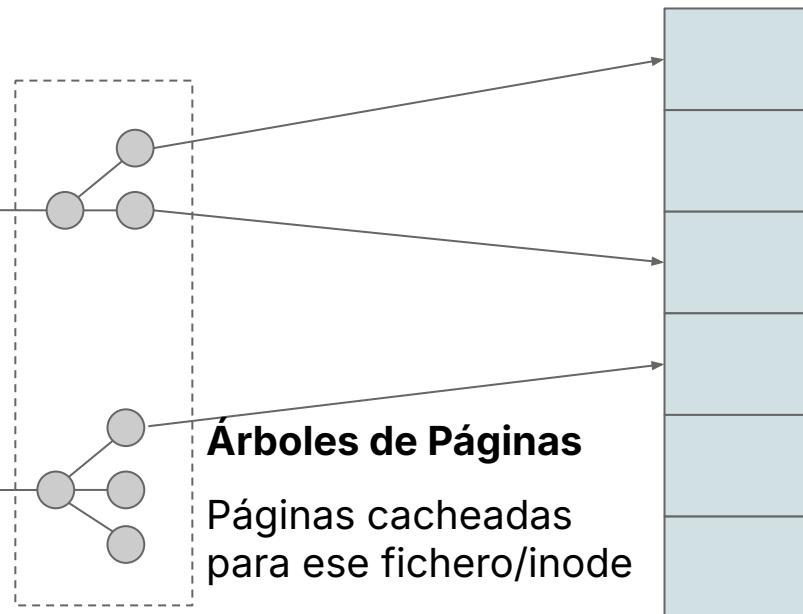
Ficheros (E/S, mmap)



Dispositivos Bloque



Marcos de Página  
(struct page)



Memoria Principal

# Caché de Páginas (II)

- El núcleo incluye **mecanismos de búsqueda** para:
  - Localizar todos los segmentos de memoria virtual a los que corresponde un marco de página (p.ej. invalidar el segmento en todos los procesos que lo mapean)
  - Localizar los bloques de memoria a partir del fichero y desplazamiento
- El núcleo es el responsable de mantener las páginas de la caché y los datos en cada dispositivo. Eventos de **sincronización**:
  - Periódica (e.g. cada 30s)
  - Llamadas al sistema p.ej. msync(2)
  - Número de páginas modificadas (*dirty*)
  - Falta de páginas de memoria (reemplazamiento)
- Los procesos pueden evitar la caché de páginas usando la opción **O\_DIRECT** de la llamada open(2), p.ej. caché en el espacio de usuario.



# SISTEMAS OPERATIVOS

*Grado en Desarrollo de Videojuegos  
Universidad Complutense de Madrid*

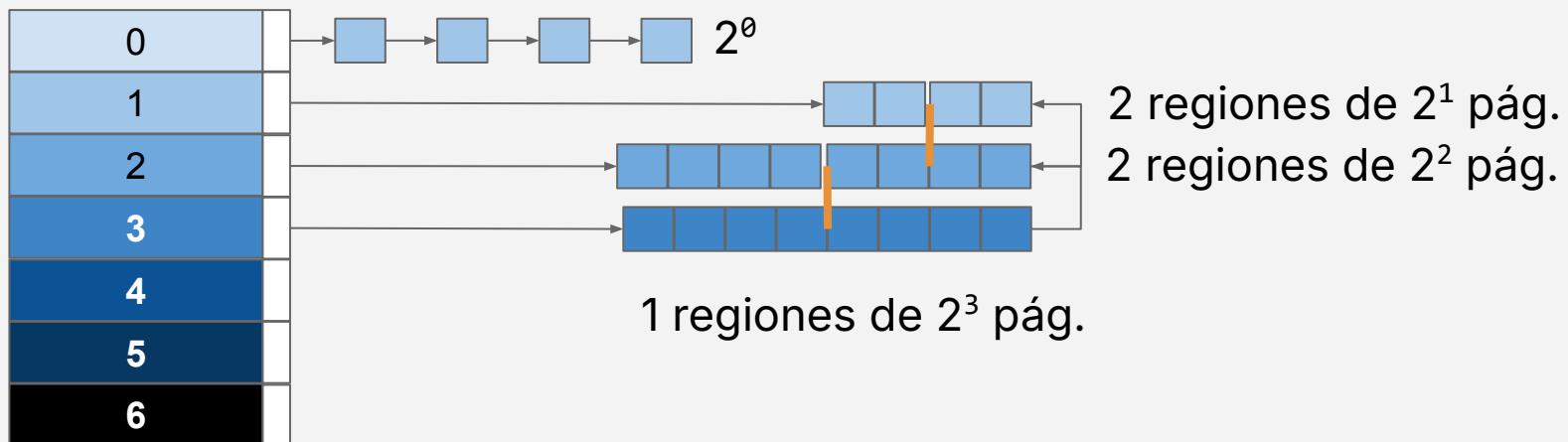
## TEMA 4.4 Gestión de la Memoria Principal

# Gestión de la Memoria Principal (I)

## Asignación de Marcos de Página

- En un sistema de memoria virtual paginada el sistema de asignación es responsable de buscar **marcos de páginas libres para un nuevo segmento**.
- El algoritmo más común en este caso es *Binary Buddy*
  - Los marcos se organizan en listas de  $2^{\text{ORDEN}}$  marcos contiguos
  - Cuando no hay una región de ORDEN adecuado
    - Se busca una región de  $\text{ORDEN}' > \text{ORDEN}$
    - Se divide en dos compañeros (*buddies*,  $\text{ORDEN}' - 1$ ) asignando uno y colocando el otro en la lista de regiones libres.

**Ejemplo:** Solicitud de una sección de 2 páginas contiguas ( $\text{ORDEN} = 1$ )



# Gestión de la Memoria Principal (II)

## Política de Reemplazamiento

- Cuando no hay marcos libres en memoria principal, determina qué marco de página debe liberarse para alojar una nueva página.
- **Objetivo:** minimizar los fallos de página, eliminando las páginas que no se volverán a usar.
- **Retención:** Determinadas páginas de memoria no se pueden reemplazar (p.ej. kernel del SO).
  - Las aplicaciones tiene acceso con la llamada mlock(2)
  - Casos de uso: tiempo real y seguridad
- **Ámbito:**
  - **Local:** Sólo puede reemplazarse un marco asignado al proceso que causa fallo.
  - **Global:** Puede reemplazarse cualquier marco
- **Algoritmos:** FIFO, Reloj, LRU

# Gestión de la Memoria Principal (III)

## Política de Reemplazamiento. FIFO

- **Criterio:** Página que lleva más tiempo residente en memoria principal
- Implementación sencilla
- Las páginas más longevas pueden seguir en uso activo

### Ejemplo:

- Memoria física con 4 marcos de página
- Referencias: *a b g a d e a b a d e g d e*

| Ref | a                    | b                    | g                    | a        | d                    | e                    | a                    | b                    | a | d        | e        | g                     | d                     | e                     |
|-----|----------------------|----------------------|----------------------|----------|----------------------|----------------------|----------------------|----------------------|---|----------|----------|-----------------------|-----------------------|-----------------------|
| 0   | <i>a<sub>1</sub></i> | a                    | a                    | <i>a</i> | <i>a</i>             | <i>e<sub>6</sub></i> | e                    | e                    | e | e        | <i>e</i> | <i>e</i>              | <i>d<sub>13</sub></i> | d                     |
| 1   |                      | <i>b<sub>2</sub></i> | b                    | b        | <i>b</i>             | <i>b<sub>7</sub></i> | a                    | <i>a</i>             | a | a        | a        | a                     | <i>a</i>              | <i>e<sub>14</sub></i> |
| 2   |                      |                      | <i>g<sub>3</sub></i> | g        | g                    | g                    | <i>g<sub>8</sub></i> | <i>b<sub>8</sub></i> | b | b        | b        | b                     | b                     | b                     |
| 3   |                      |                      |                      |          | <i>d<sub>5</sub></i> | d                    | d                    | d                    | d | <i>d</i> | <i>d</i> | <i>g<sub>12</sub></i> | g                     | g                     |

10 fallos

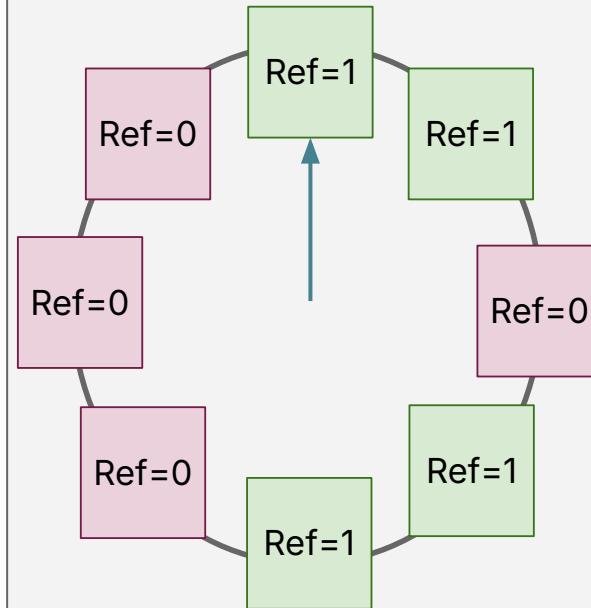
# Gestión de la Memoria Principal (IV)

## Política de Reemplazamiento. Reloj (o 2<sup>a</sup> oportunidad)

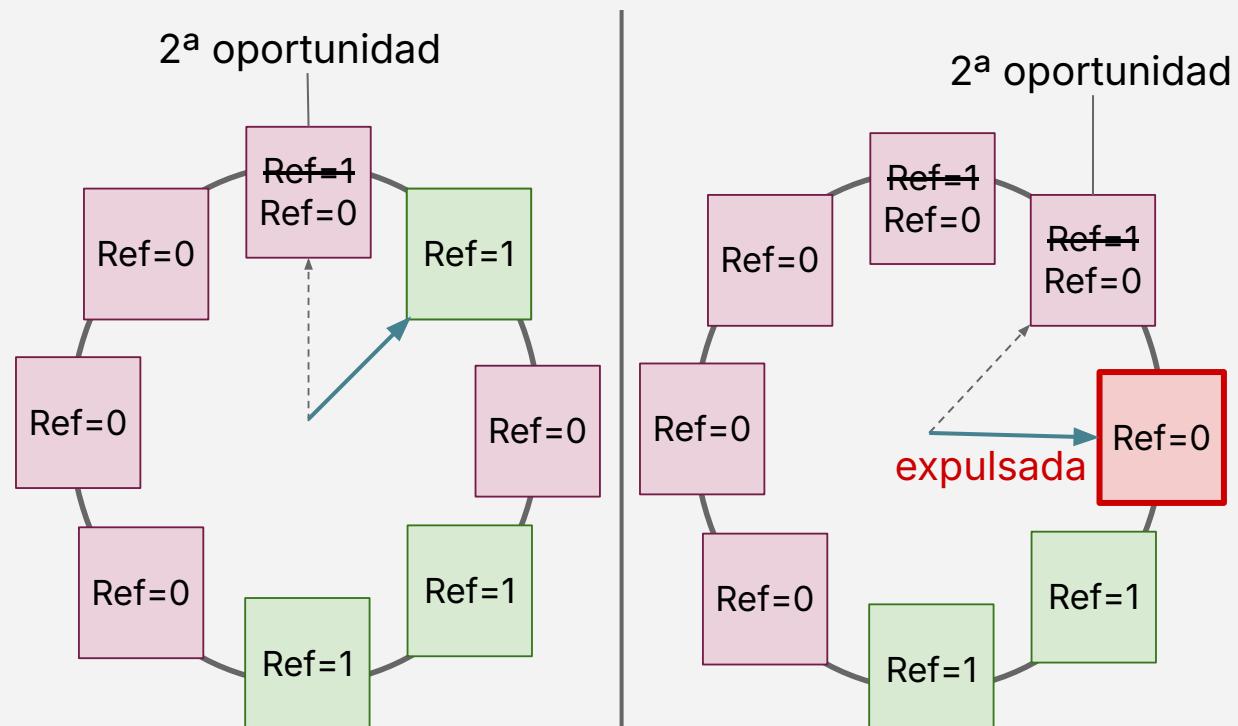
- **Criterio:** FIFO y uso del flag de acceso a la página (Ref)
  - Si la página a reemplazar (*first-in*) no se ha accedido (Ref=0) se reemplaza
  - Si el bit está activo (Ref=1) se borra el flag y se pasa al final de la lista.
- Implementación basada en una cola circular.

### Ejemplo

Inicio



2<sup>a</sup> oportunidad



# Gestión de la Memoria Principal (V)

## Ejemplo:

- Memoria física con 4 marcos de página
- Referencias: *a b g a d e a b a d e g d e*

| Ref | a                      | b                      | g                      | a                      | d                      | e                      | a                      | b                      | a                      | d                      | e                      | g                      | d                      | e                      |
|-----|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 0   | ↓ <i>a<sub>0</sub></i> | ↓ <i>a<sub>0</sub></i> | ↓ <i>a<sub>0</sub></i> | ↓ <i>a<sub>1</sub></i> | ↓ <i>a<sub>1</sub></i> | <i>a<sub>0</sub></i>   | <i>a<sub>0</sub></i>   | ↓ <i>a<sub>1</sub></i> | <i>a<sub>1</sub></i>   | ↓ <i>a<sub>1</sub></i> | <i>a<sub>1</sub></i>   | ↓ <i>a<sub>0</sub></i> | <i>a<sub>0</sub></i>   | ↓ <i>a<sub>0</sub></i> |
| 1   |                        | ↓ <i>b<sub>0</sub></i> | <i>b<sub>0</sub></i>   | <i>b<sub>0</sub></i>   | <i>b<sub>0</sub></i>   | ↓ <i>b<sub>0</sub></i> | ↓ <i>e<sub>0</sub></i> | <i>e<sub>0</sub></i>   | <i>e<sub>0</sub></i>   | <i>e<sub>0</sub></i>   | ↓ <i>e<sub>1</sub></i> | <i>e<sub>0</sub></i>   | <i>e<sub>0</sub></i>   | ↓ <i>e<sub>1</sub></i> |
| 2   |                        |                        | ↓ <i>g<sub>0</sub></i> | <i>g<sub>0</sub></i>   | <i>g<sub>0</sub></i>   | <i>g<sub>0</sub></i>   | ↓ <i>g<sub>0</sub></i> | ↓ <i>g<sub>0</sub></i> | ↓ <i>b<sub>0</sub></i> | <i>b<sub>0</sub></i>   | <i>b<sub>0</sub></i>   | ↓ <i>g<sub>0</sub></i> | <i>g<sub>0</sub></i>   | <i>g<sub>0</sub></i>   |
| 3   |                        |                        |                        | ↓ <i>d<sub>0</sub></i> | <i>d<sub>0</sub></i>   | <i>d<sub>0</sub></i>   | <i>d<sub>0</sub></i>   | ↓ <i>d<sub>0</sub></i> | ↓ <i>d<sub>0</sub></i> | ↓ <i>d<sub>1</sub></i> | ↓ <i>d<sub>1</sub></i> | ↓ <i>d<sub>0</sub></i> | ↓ <i>d<sub>1</sub></i> | ↓ <i>d<sub>1</sub></i> |

7 fallos

# Gestión de la Memoria Principal (VI)

## Política de Reemplazamiento. LRU

- **Criterio:** Página residente que fue accedida hace más tiempo
- **Principio de localidad.** Una referencia de memoria probablemente volverá a repetirse en el futuro próximo.
- Mecanismo para registrar las referencias a la página muy difícil de implementar
- **Linux LRU** se implementa con dos colas de páginas (`vmstat(1) active/inactive`)
  - Páginas activas. Usa el mecanismo del reloj con el flag de acceso de la TP.
  - Páginas inactivas. Usa un algoritmo FIFO

**Ejemplo:** 4 marcos de página. Referencias: *a b g a d e a b a d e g d e*

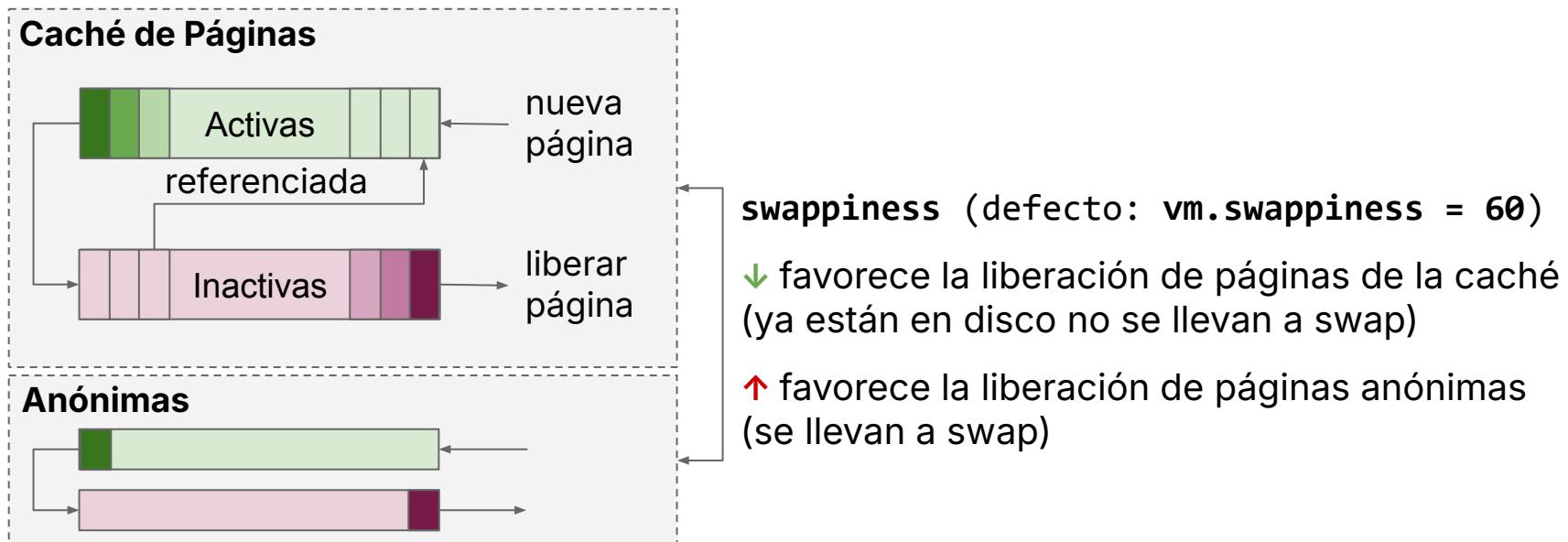
| Ref | a                    | b                    | g                    | a                    | d        | e                    | a                    | b | a                     | d                     | e                     | g                     | d                     | e |
|-----|----------------------|----------------------|----------------------|----------------------|----------|----------------------|----------------------|---|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|---|
| 0   | <i>a<sub>1</sub></i> | a                    | a                    | <i>a<sub>4</sub></i> | a        | a                    | <i>a<sub>7</sub></i> | a | <i>a<sub>9</sub></i>  | a                     | a                     | a                     | a                     | a |
| 1   |                      | <i>b<sub>2</sub></i> | b                    | b                    | <i>b</i> | <i>e<sub>6</sub></i> | e                    | e | e                     | <i>e<sub>11</sub></i> | e                     | e                     | <i>e<sub>14</sub></i> |   |
| 2   |                      |                      | <i>g<sub>3</sub></i> | g                    | g        | <i>g</i>             | <i>g<sub>8</sub></i> | b | b                     | <i>b</i>              | <i>g<sub>12</sub></i> | g                     | g                     |   |
| 3   |                      |                      |                      | <i>d<sub>5</sub></i> | d        | d                    | d                    | d | <i>d<sub>10</sub></i> | d                     | d                     | <i>d<sub>13</sub></i> | d                     |   |

7 fallos

# Gestión de la Memoria Principal (VI)

## Page Buffering

- Los páginas de memoria no referenciadas no se eliminan directamente. Se mantienen como páginas *reclamables* (páginas inactiva)
- Si la página vuelve a referenciarse se mueve a la cola de páginas activa evitando que se elimine.
- Demonio de paginación (kswapd en Linux) actúa en función de umbrales de uso
  - Ejecuta la política de reemplazo para liberar marcos de página de memoria
  - Libera páginas anónimas (swap-out) o de la caché de páginas



# Gestión de la Memoria Principal (VII)

```
$ free
```

|       | total    | used    | free     | shared | buff/cache | available |
|-------|----------|---------|----------|--------|------------|-----------|
| Mem:  | 32430644 | 4520676 | 26305888 | 999572 | 3028180    | 27909968  |
| Swap: | 16777212 | 0       | 16777212 |        |            |           |

```
$ cat /proc/meminfo
```

|                  |                   |
|------------------|-------------------|
| MemTotal:        | 32430644 kB       |
| MemFree:         | 26361972 kB       |
| MemAvailable:    | 27965488 kB       |
| <b>Buffers:</b>  | <b>237912 kB</b>  |
| <b>Cached:</b>   | <b>2672568 kB</b> |
| SwapCached:      | 0 kB              |
| <b>Active:</b>   | <b>3469572 kB</b> |
| <b>Inactive:</b> | <b>1253544 kB</b> |
| Active(anon):    | 2805132 kB        |
| Inactive(anon):  | 0 kB              |
| Active(file):    | 664440 kB         |
| Inactive(file):  | 1253544 kB        |
| Unevictable:     | 813624 kB         |

Memoria para E/S de **dispositivos en bloque**.  
(en la caché de páginas)

Memoria en la **caché de páginas (ficheros)**

Estadísticas de la **política LRU**:

- file de la caché de páginas
- anon páginas anónimas

Otros comandos para obtener **información de la memoria virtual** : vmstat, swapon, pmap, ps y top