Tema 3. Gestión de Memoria

3.1. Gestión de memoria principal
3.2. Memoria virtual: Organización
3.3. Memoria virtual: Gestión
3.4. Gestión de memoria en Linux

Objetivos

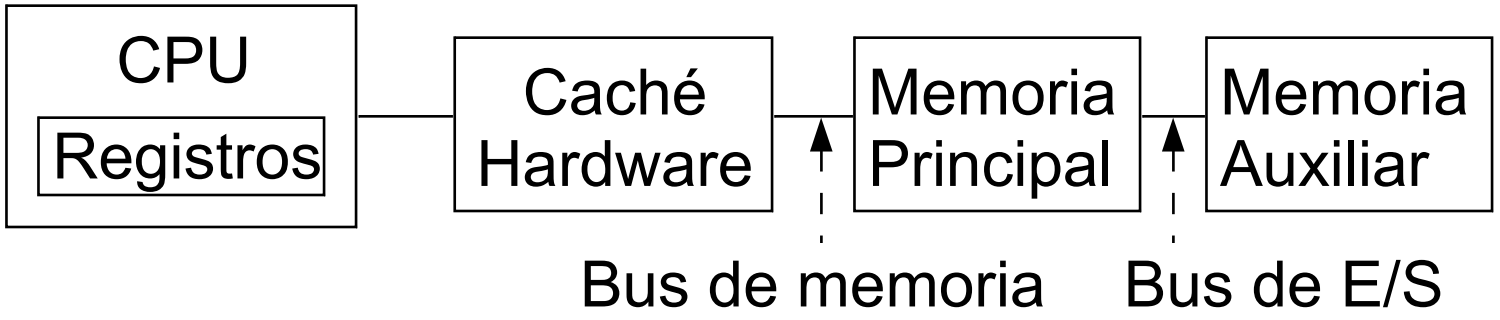
- Distinguir entre dirección relativa o lógica y dirección física o real y entre espacio de direcciones lógico y físico
- Conocer las distintas formas en las que el sistema operativo puede organizar y gestionar la memoria física
- Saber en qué consiste y para qué se utiliza el intercambio (*swapping*)
- Entender qué es la Memoria Virtual y por qué se utiliza
- Conocer los mecanismos de paginación, segmentación y segmentación paginada en un sistema con memoria virtual
- Comprender qué es la propiedad de localidad y su relación con el comportamiento de un programa en ejecución
- Conocer la teoría del conjunto de trabajo y el problema de la hiperpaginación
- Saber cómo gestiona Linux la memoria de un proceso

Jerarquía de Memoria

Dos principios sobre memoria:

- Menor cantidad, acceso más rápido
- Mayor cantidad, menor coste por byte

Así, los elementos frecuentemente accedidos se ponen en memoria rápida, cara y pequeña; el resto, en memoria lenta, grande y barata.



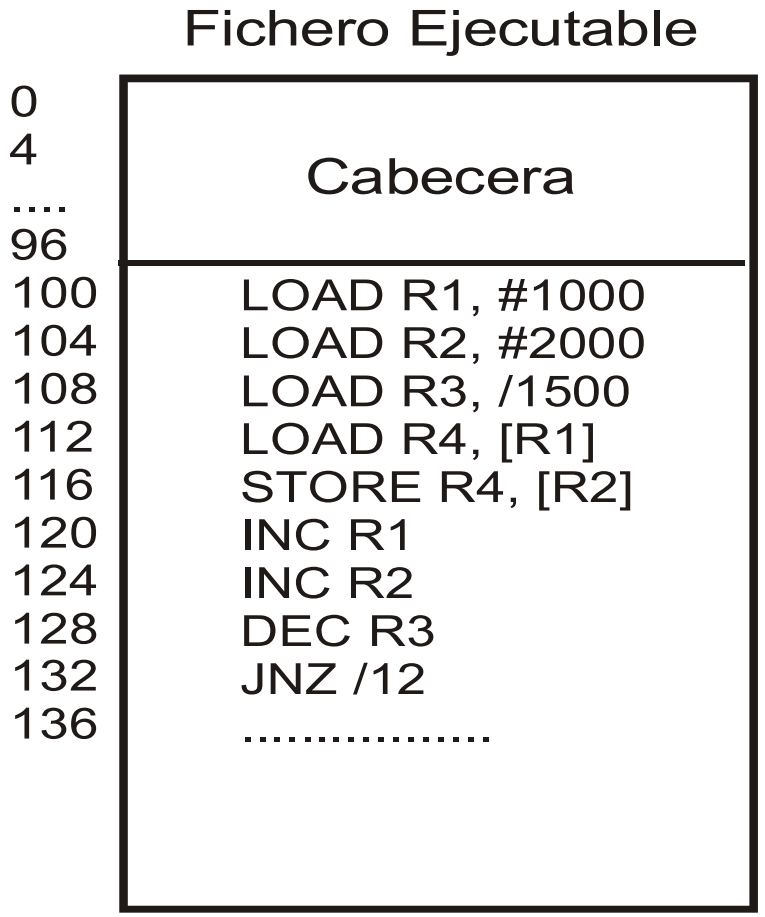
Conceptos sobre Cachés

- **Caché** - copia que puede ser accedida más rápidamente que el original
- **Idea**: hacer los casos frecuentes eficientes, los caminos infrecuentes no importan tanto
- *Acierto de caché*: item en la caché
- *Fallo de caché*: item no en caché; hay que realizar la operación completa
- **Tiempo de Acceso Efectivo** (TAE) = Probabilidad_acierto * coste_acierto + Probabilidad_fallo * coste_fallo
- Funciona porque los programas no son aleatorios, explotan la **localidad -> principio de localidad**

Espacio de direcciones lógico y espacio de direcciones físico

Espacio de direcciones lógico: conjunto de direcciones lógicas o virtuales generadas por un programa

Espacio de direcciones físico: conjunto de direcciones físicas correspondientes a las direcciones lógicas en un instante dado



Carretero, p.165

Mapa de memoria de un proceso

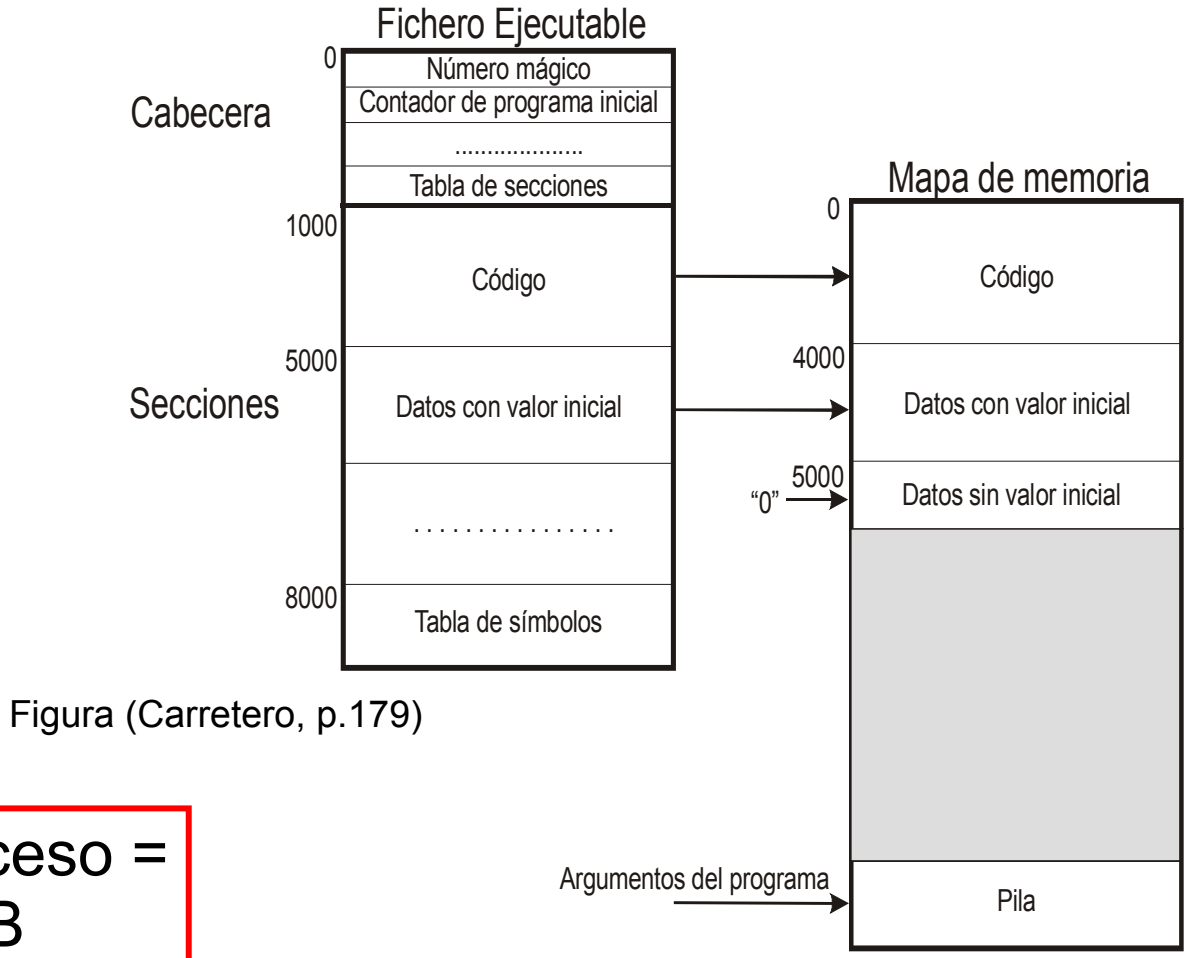


Figura (Carretero, p.179)

Imagen de proceso = mapa + PCB

Objetivos generales de la gestión de memoria

Organización: ¿cómo está dividida la memoria? ¿un proceso o varios procesos?

Gestión: Dado un esquema de organización, ¿qué estrategias se deben seguir para obtener un rendimiento óptimo?

- Estrategias de **asignación:** Contigua, No contigua
- Estrategias de **sustitución** o **reemplazo**
- Estrategias de **búsqueda** o **recuperación**

Protección

- El SO de los procesos de usuario
- Los procesos de usuario entre ellos

Asignación contigua y no contigua

Podemos clasificar las organizaciones de memoria como:

- *Contiguas* - La asignación de almacenamiento para un programa se hace en un único bloque de posiciones continuas de memoria -> Particiones fijas y Particiones variables
- *No contiguas* - Permiten dividir el programa en bloques o segmentos que se pueden colocar en zonas no necesariamente continuas de memoria principal -> Paginación, Segmentación y Segmentación paginada

Intercambio (*Swapping*)

- Intercambiar procesos entre memoria y un almacenamiento auxiliar
- El almacenamiento auxiliar debe ser un disco rápido con espacio para albergar las imágenes de memoria de los procesos de usuario
- El factor principal en el tiempo de intercambio es el tiempo de transferencia
- El ***intercambiador*** tiene las siguientes responsabilidades:
  - » Seleccionar procesos para retirarlos de MP
  - » Seleccionar procesos para incorporarlos a MP
  - » Gestionar y asignar el espacio de intercambio

3.2. Memoria virtual: Organización

- Concepto de Memoria Virtual
  - » El tamaño del programa, los datos y la pila puede exceder la cantidad de memoria física disponible para él.
  - » Se usa un almacenamiento a dos niveles:
    - – *Memoria Principal* **->** partes del proceso necesarias en un momento dado
    - – *Memoria Secundaria* **->** espacio de direcciones completo del proceso

Concepto de memoria virtual (y II)

» Es necesario:

- saber qué se encuentra en memoria principal
- una política de movimiento entre MP y MS

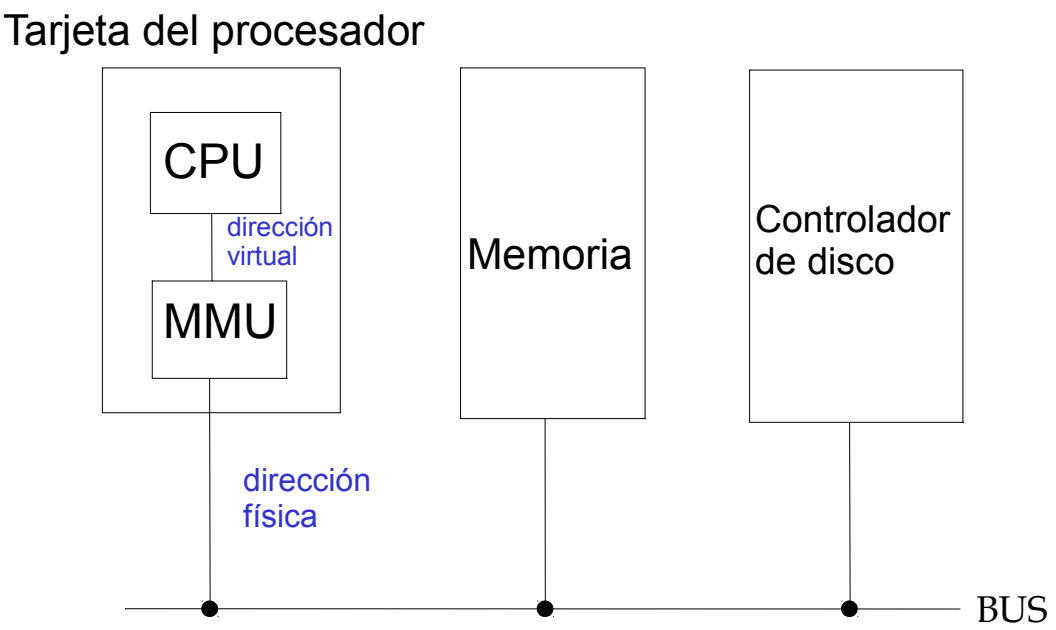
- Además, la memoria virtual
  - » resuelve el problema del crecimiento dinámico de los procesos
  - » permite aumentar el grado de multiprogramación

Unidad de Gestión de Memoria

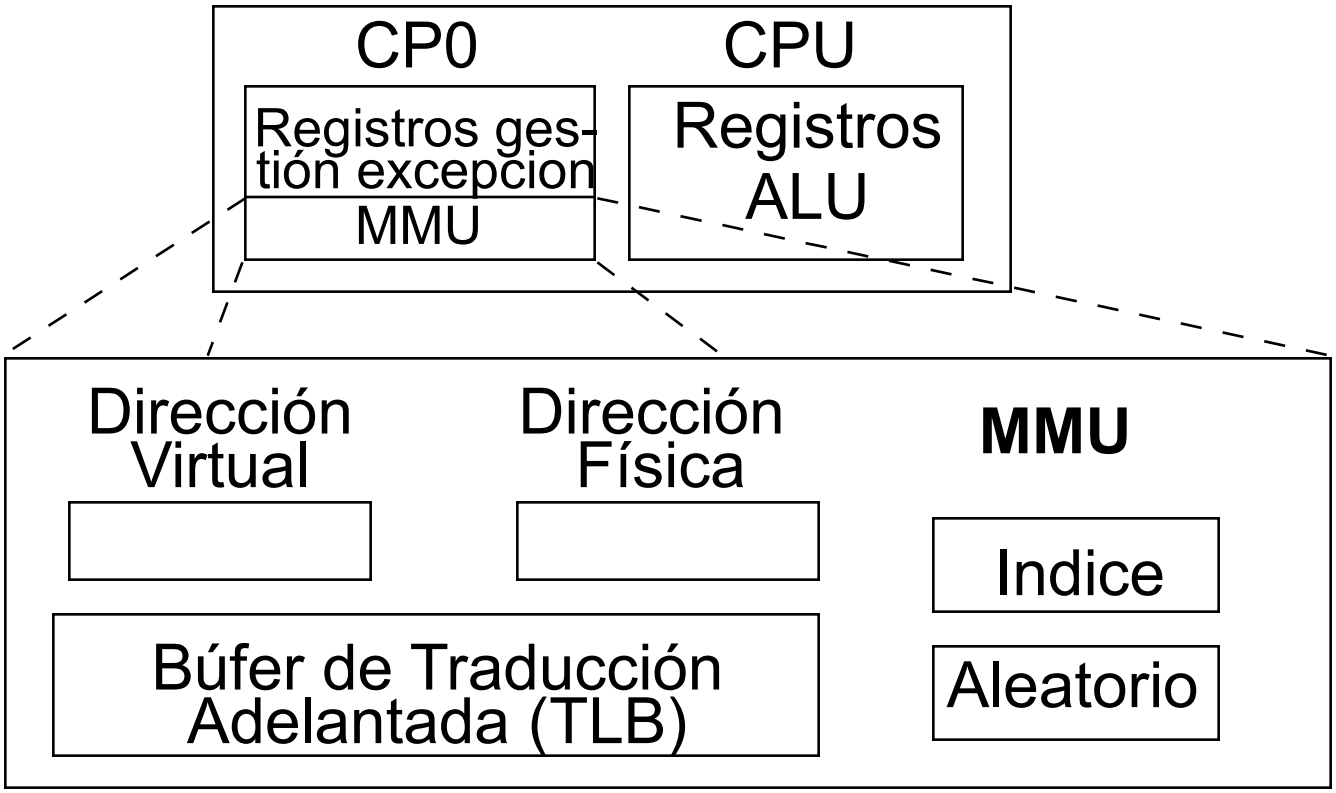
- La **MMU** (*Memory Management Unit*) es un dispositivo hardware que traduce direcciones virtuales a direcciones físicas ¡Este dispositivo está gestionado por el SO!
- En el esquema MMU más simple, el valor del registro base se añade a cada dirección generada por el proceso de usuario al mismo tiempo que es enviado a memoria
- El programa de usuario trata sólo con direcciones lógicas, nunca con direcciones reales

Unidad de Gestión de Memoria (y II)

• Además de la traducción, el MMU deberá:
- detectar si la dirección aludida se encuentra o no en MP
- generar una excepción si no se encuentra en MP



Ejemplo de MMU: El MIPS R2000/3000



Paginación

- El espacio de direcciones físicas de un proceso puede ser no contiguo
- La memoria física se divide en bloques de tamaño fijo, denominados *marcos de página*. El tamaño es potencia de dos, de 0.5 a 8 Kb
- El espacio lógico de un proceso se divide en bloques del mismo tamaño, denominados *páginas*
- Los marcos de páginas contendrán páginas de los procesos

Paginación (y II)

- Las **direcciones lógicas**, que son las que genera la CPU se dividen en ***número de página*** (p) y ***desplazamiento*** dentro de la página (d)

- Las **direcciones físicas** se dividen en ***número de marco*** (m, *dirección base del* marco donde está almacenada la página) y ***desplazamiento*** (d)

Paginación (y III)

- Cuando la CPU genere una dirección lógica será necesario traducirla a la dirección física correspondiente, la *tabla de páginas* mantiene información necesaria para realizar dicha traducción. *Existe una tabla de páginas por proceso*
- *Tabla de ubicación en disco* (una por proceso) ubicación de cada página en el almacenamiento secundario
- *Tabla de marcos de página*, usada por el S.O. y contiene información sobre cada marco de página

Contenido de la tabla de páginas

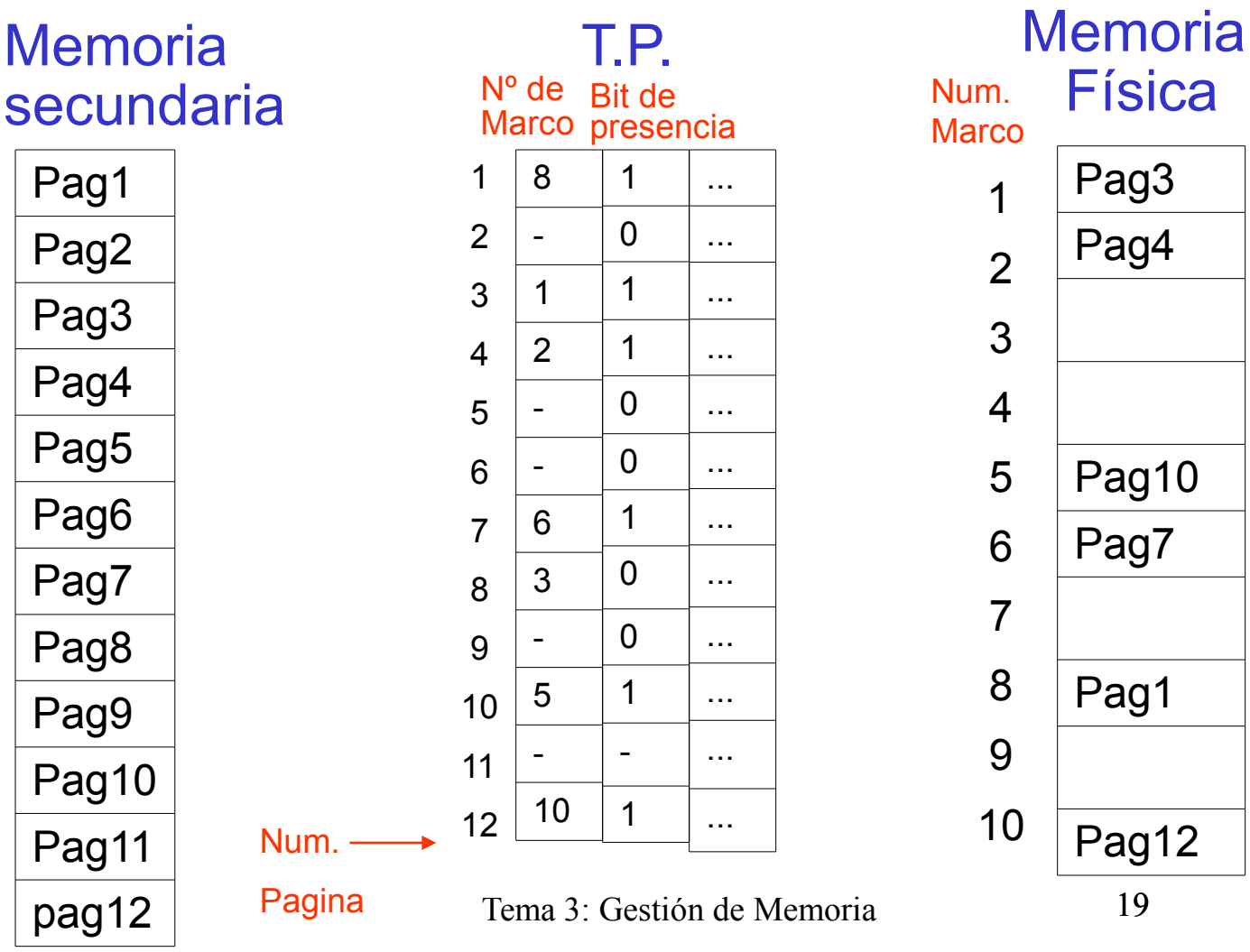
Una entrada por cada página del proceso:

- Número de marco (dirección base del marco) en el que está almacenada la página si está en MP
- Bit de presencia o bit válido
- Bit de modificación
- Modo de acceso autorizado a la página (bits de protección)

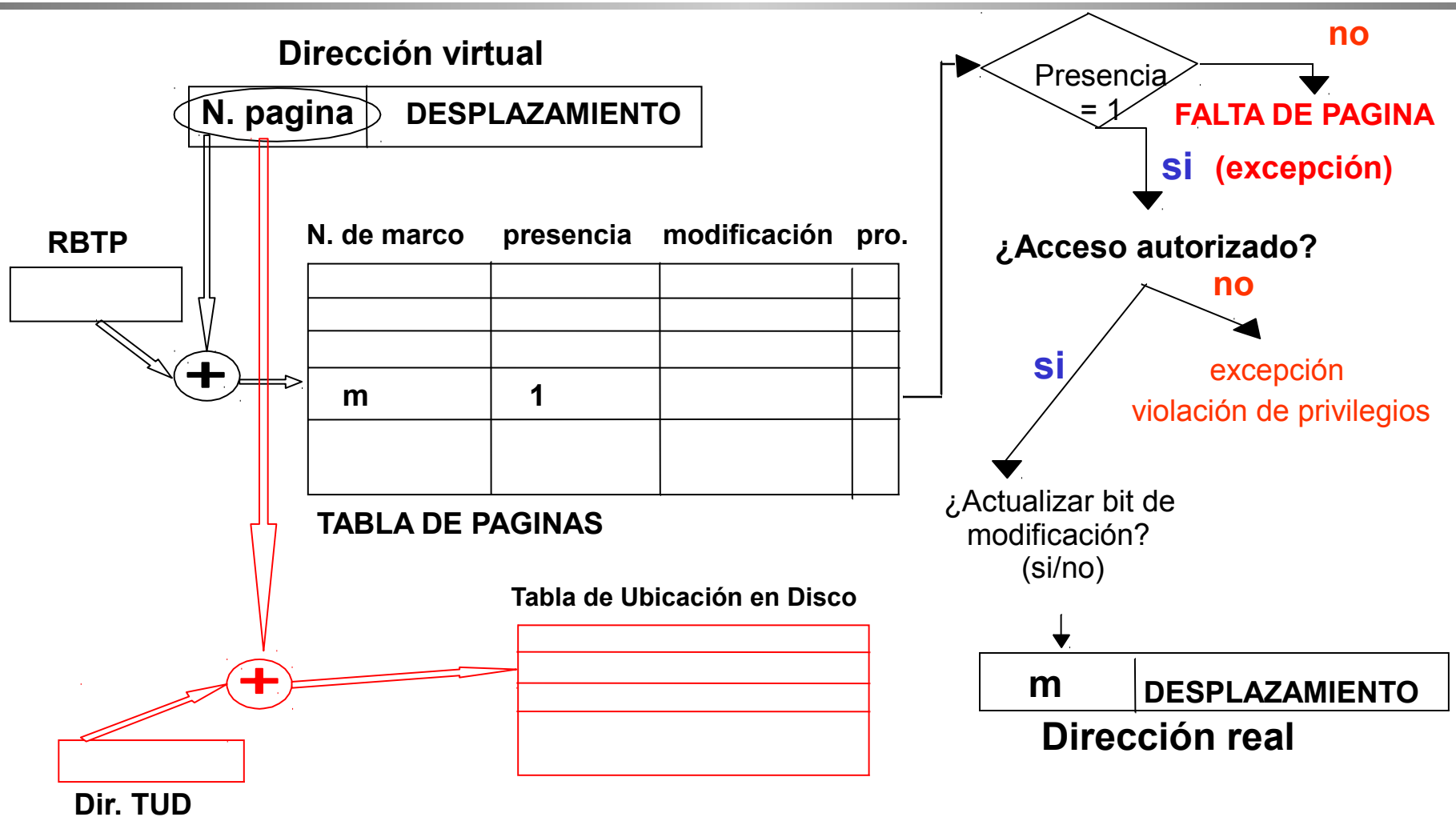
Nº de página

nº marco	presencia	modificación	protección
46	**1**	**0**	**01**

Ejemplo: contenido de la tabla de páginas



Esquema de traducción



Falta de página

1. Bloquear proceso
2. Encontrar la ubicación en disco de la página solicitada (*tabla de ubicación en disco*)
3. Encontrar un marco libre. Si no hubiera, se puede optar por desplazar una página de MP
4. Cargar la página desde disco al marco de MP
5. Actualizar tablas (bit presencia=1, nº marco, ...)
6. Desbloquear proceso
7. Reiniciar la instrucción que originó la falta de página

Implementación de la Tabla de Páginas

- La tabla de páginas se mantiene en memoria principal
- El *registro base de la tabla de páginas* (*RBTP*) apunta a la tabla de páginas (suele almacenarse en el PCB del proceso)
- En este este esquema:
  - » cada acceso a una instrucción o dato requiere dos accesos a memoria: uno a la tabla de páginas y otro a memoria → resuelto con TLB (búfer de traducción anticipada)
  - » un problema adicional viene determinado por el tamaño de la tabla de páginas

Tamaño de la Tabla de Páginas

- Ejemplo:
  - » Dirección virtual: 32 bits.
  - » Tamaño de página = 4 Kbytes (2^{12} bytes).
- tamaño del campo desplazamiento = **12 bits**
- tamaño número de página virtual = **20 bits**
- Nº de páginas virtuales = 2^{20} = **¡1,048,576!**
- Solución para reducir el tamaño de la TP:
  - » Paginación multinivel

Paginación multinivel

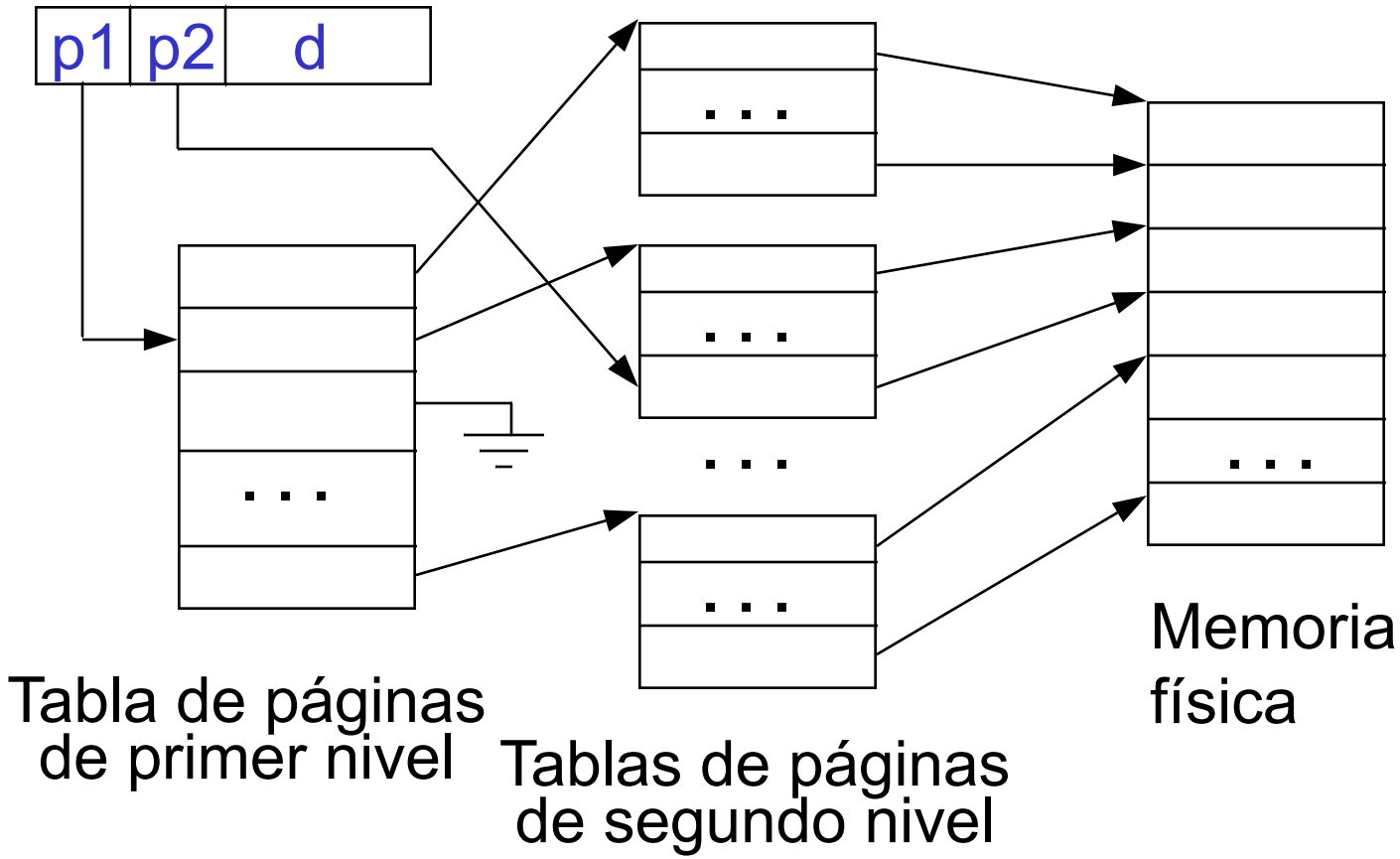
- Esta solución opta por “paginar las tablas de páginas”

- La partición de la tabla de páginas permite al SO dejar particiones no usadas sin cargar hasta que el proceso las necesita. Aquellas porciones del espacio de direcciones que no se usan no necesitan tener tabla de página

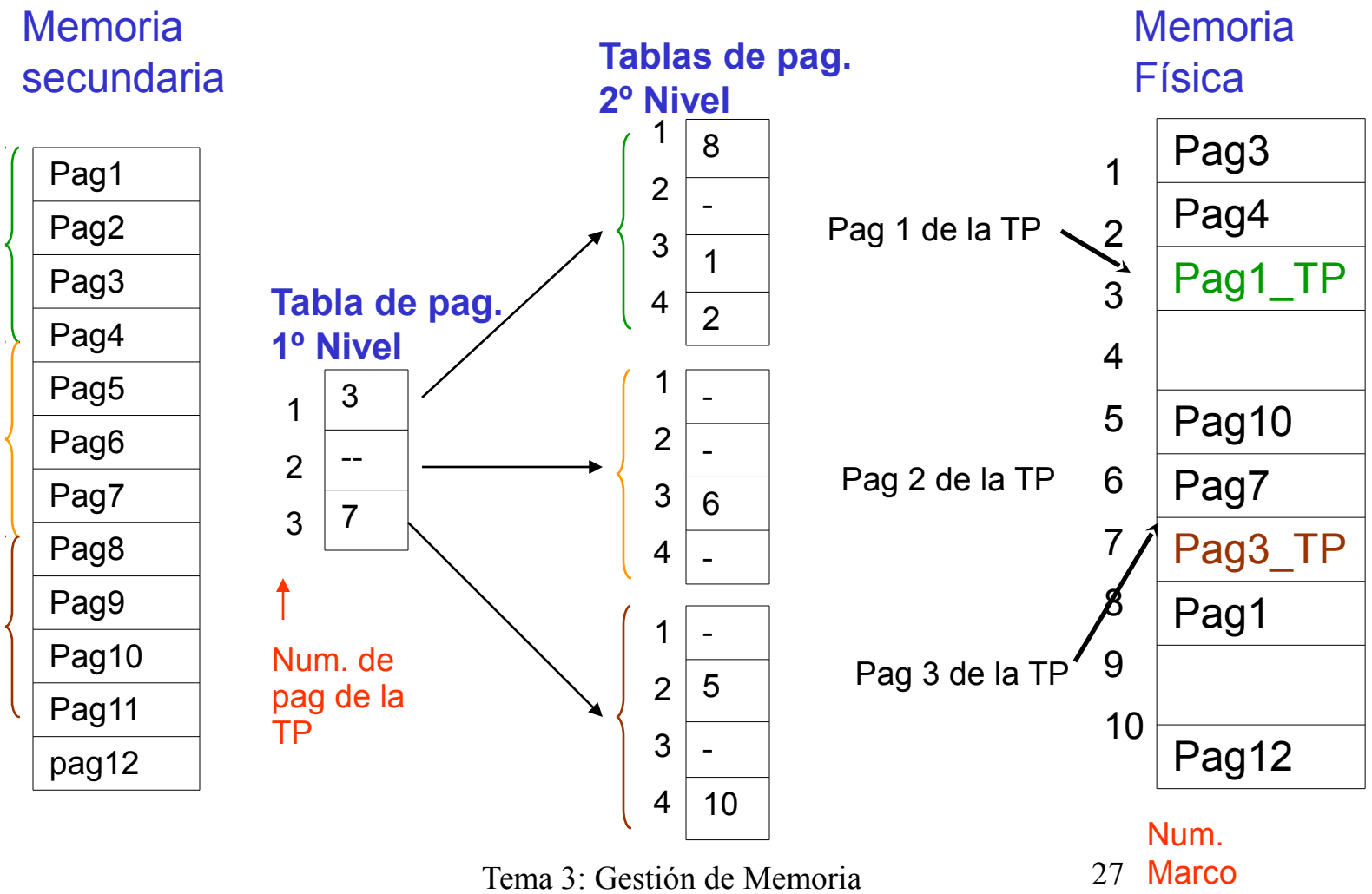
Paginación a dos niveles

- Lo que hacemos es dividir la tabla de páginas en partes del tamaño de una página.
- La dirección lógica se divide en:
  - » número de página (n bits):
    - – un número de página p1 (=k)
    - – desplazamiento de página p2 (=n-k)
  - » desplazamiento de página d (m bits)
- Así una dirección lógica es de la forma:

Esquema de paginación a dos niveles

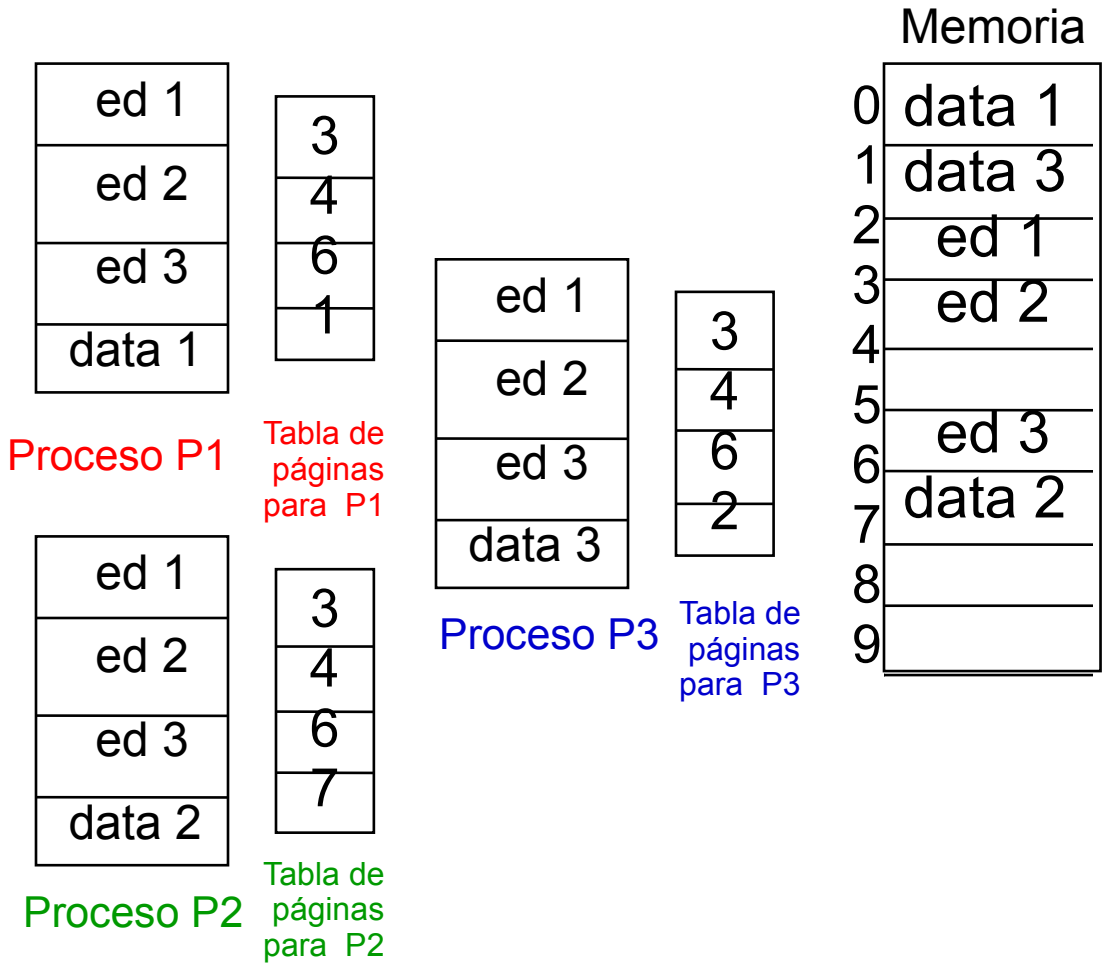


Ejemplo: Esquema de paginación a dos niveles



Páginas compartidas

- Una copia de código de solo lectura (*reentrante*) compartido entre varios procesos. Ej. editores, compiladores, sistemas de ventanas



Segmentación

- Esquema de organización de memoria que soporta mejor la visión de memoria del usuario: un programa es una colección de unidades lógicas -segmentos-, p. ej. procedimientos, funciones, pila, tabla de símbolos, matrices, etc.

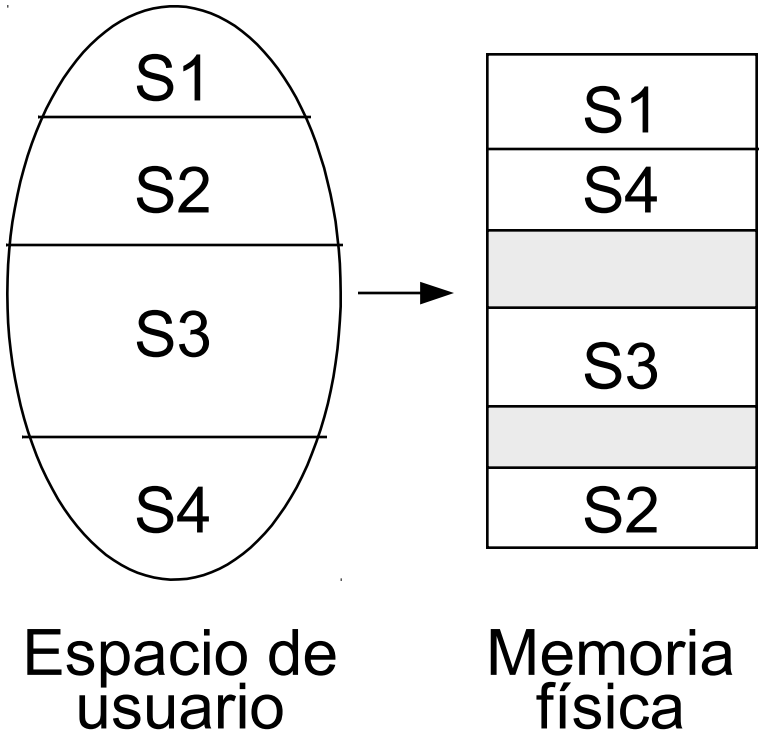


Tabla de Segmentos

- Una dirección lógica es una tupla:

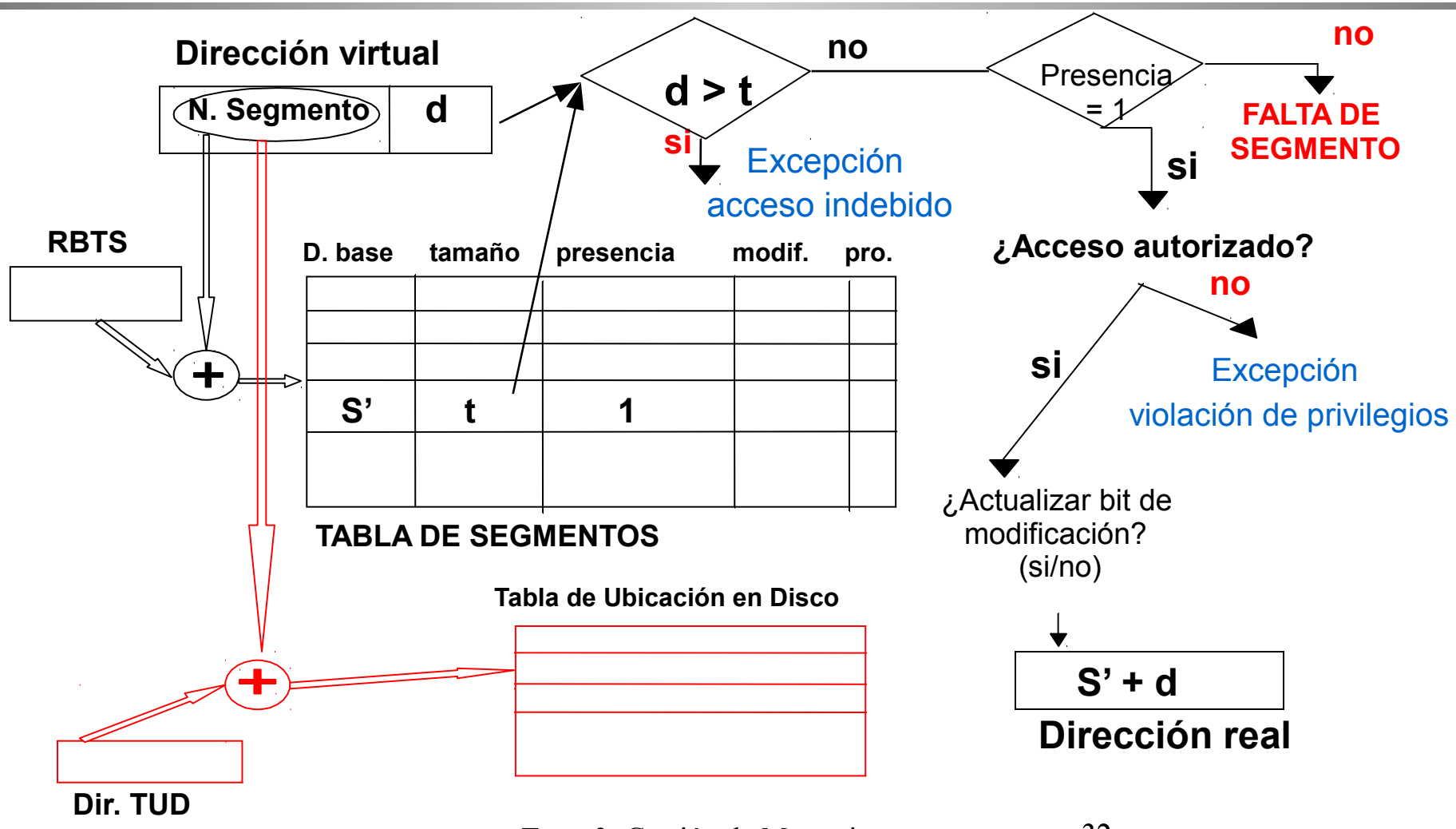
  <número_de_segmento, desplazamiento>

- La *Tabla de Segmentos* aplica direcciones bidimensionales definidas por el usuario en direcciones físicas de una dimensión. Cada entrada de la tabla tiene los siguientes elementos (aparte de presencia, modificación y protección):
  - » *base* - dirección física donde reside el inicio del segmento en memoria.
  - » *tamaño* - longitud del segmento.

Implementación de la Tabla de Segmentos

- La tabla de segmentos se mantiene en memoria principal
- El *Registro Base de la Tabla de Segmentos* (*RBTS*) apunta a la tabla de segmentos (suele almacenarse en el PCB del proceso)
- El *Registro Longitud de la Tabla de Segmentos* (*STLR*) indica el número de segmentos del proceso; el nº de segmento *s,* generado en una dirección lógica, es legal si $s < STLR$ (suele almacenarse en el PCB del proceso)

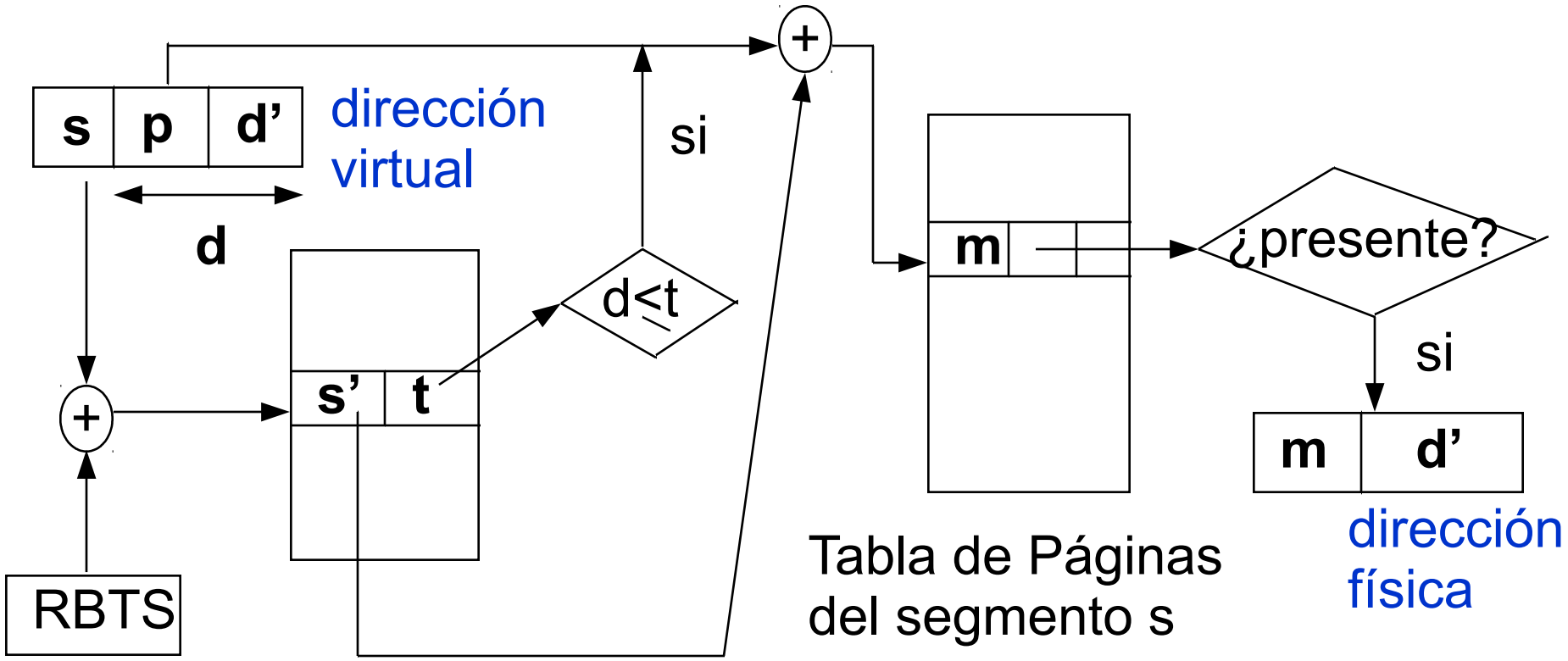
Esquema de traducción



Segmentación Paginada

- La variabilidad del tamaño de los segmentos y el requisito de memoria contigua dentro de un segmento, complica la gestión de MP y MS
- Por otro lado, la paginación simplifica la gestión pero complica más los temas de compartición y protección
- Algunos sistemas combinan ambos enfoques, obteniendo la mayoría de las ventajas de la segmentación y eliminando los problemas de una gestión de memoria compleja

Esquema de traducción



s' : dirección base de la Tabla de Páginas del segmento s

3.3. Memoria Virtual: Gestión

- Gestión de Memoria Virtual con paginación
- Criterios de clasificación respecto a:
  - » *Políticas de asignación*: Fija o Variable
  - » *Políticas de búsqueda (recuperación)*:
    - – Paginación por demanda
    - – Paginación anticipada (**!=** prepaginación)
  - » *Políticas de sustitución (reemplazo)*:
    - – Sustitución local
    - – Sustitución global

Gestión de la Memoria Virtual

- Independientemente de la **política de sustitución** utilizada, existen ciertos criterios que siempre deben cumplirse:
  - » Páginas “limpias” frente a “sucias”
    - – se pretende minimizar el coste de transferencia
  - » Páginas compartidas
    - – se pretende reducir el nº de faltas de página
  - » Páginas especiales
    - – algunos marcos pueden estar bloqueados (ejemplo: buferes de E/S mientras se realiza una transferencia)

Influencia del tamaño de página

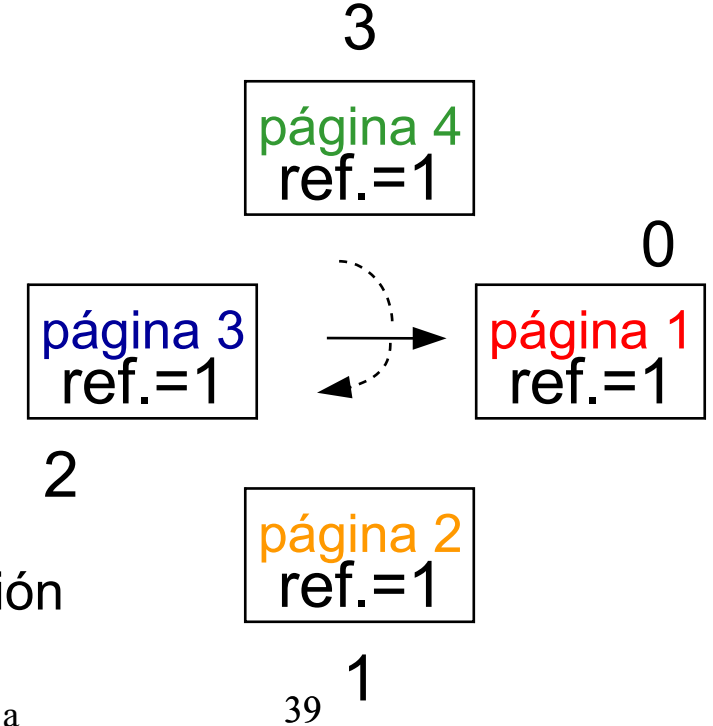
- Cuanto más pequeñas
  - » Aumento del tamaño de las tablas de páginas
  - » Aumento del nº de transferencia MP->Disco
  - » Reducen la fragmentación interna
- Cuanto más grandes
  - » Grandes cantidades de información que no serán usadas están ocupando MP
  - » Aumenta la fragmentación interna
- Búsqueda de un equilibrio

Algoritmos de sustitución

- Podemos tener las siguientes combinaciones
  - » asignación fija y sustitución local
  - » asignación variable y sustitución local
  - » asignación variable y sustitución global
- Veremos distintos algoritmos:
  - ➔ Optimo: sustituye la página que no se va a referenciar en un futuro o la que se referencie más tarde
  - ➔ FIFO: sustituye la página más antigua
  - ➔ LRU: sustituye la página que fue objeto de la referencia más antigua
  - ➔ Algoritmo del reloj

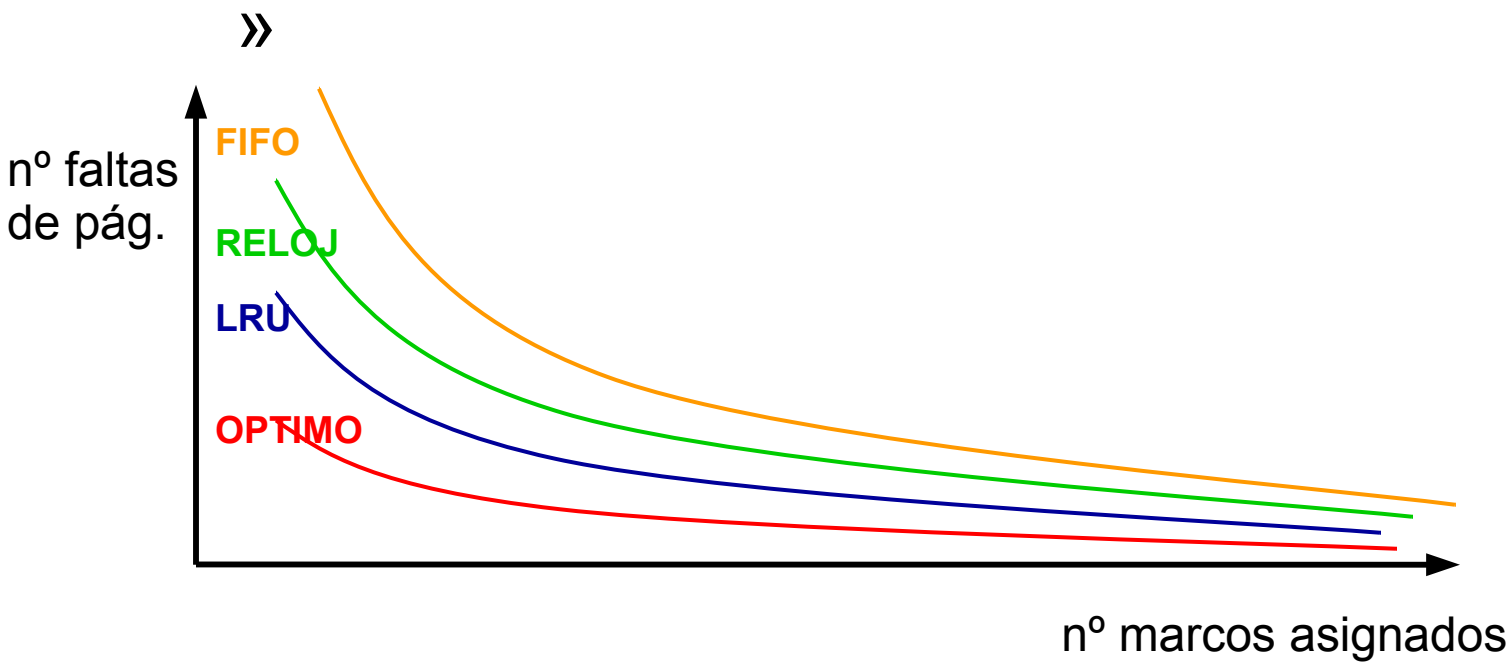
Algoritmo del reloj

- Cada página tiene asociado un bit de referencia R (lo pone a 1 el hardware)
- Los marcos de página se representan por una lista circular y un puntero a la página visitada hace más tiempo
- Selección de una página:
  1. Consultar marco actual
  2. ¿Es R=0?
     - No: R=0; ir al siguiente marco y volver al paso 1
     - Si: seleccionar para sustituir e incrementar posición



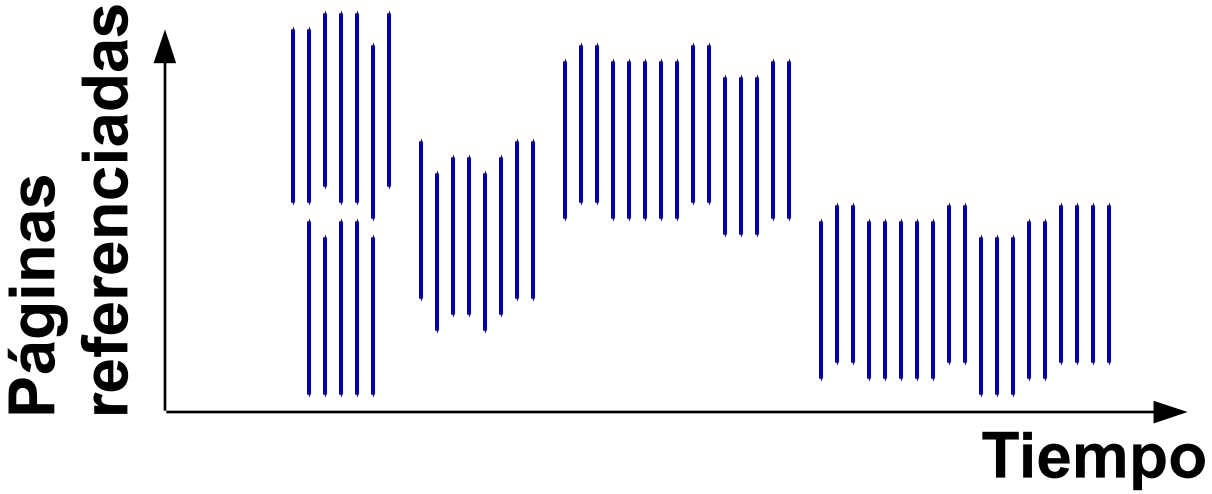
Comparación

Conclusión: Influye más la cantidad de MP disponible que el algoritmo de sustitución usado



Comportamiento de los programas

- Viene definido por la secuencia de referencias a página que realiza el proceso
- Importante para maximizar el rendimiento del sistema de memoria virtual (TLB, alg. sustitución, ...)



Propiedad de localidad

- Distintos tipos
  - » Temporal: Una posición de memoria referenciada recientemente tiene una probabilidad alta de ser referenciada en un futuro próximo (ciclos, rutinas, variables globales, ...)
  - » Espacial: Si cierta posición de memoria ha sido referenciada es altamente probable que las adyacentes también lo sean (array, ejecución secuencial, ...)

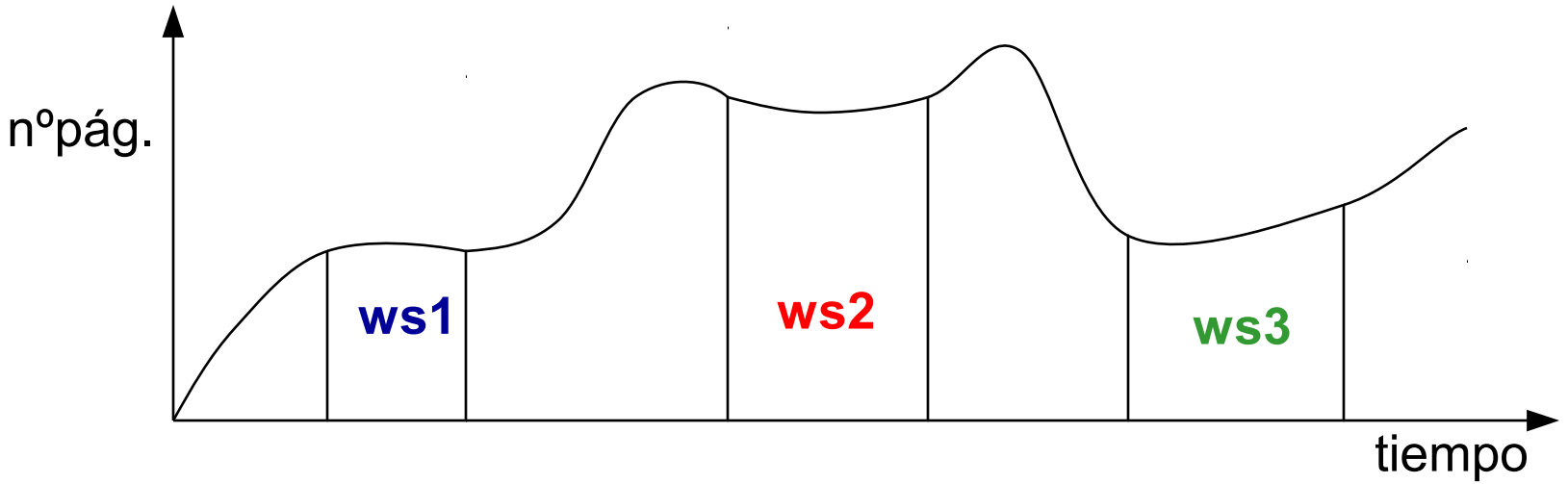
Conjunto de Trabajo

- Observaciones:
  - » Mientras el conjunto de páginas necesarias puedan residir en MP, el nº de faltas de página no crece mucho
  - » Si eliminamos de MP páginas de ese conjunto, la activación de paginación crece mucho
- Conjunto de trabajo *(Working Set)* de un proceso es el conjunto de páginas que son referenciadas frecuentemente en un determinado intervalo de tiempo

WS(t,z) = páginas referenciadas en el intervalo de tiempo t - z y t

Conjunto de Trabajo: propiedades

- Propiedades
  - » Los conjuntos de trabajo son transitorios
  - » No se puede predecir el tamaño futuro de un conjunto de trabajo
  - » Difieren unos de otros sustancialmente

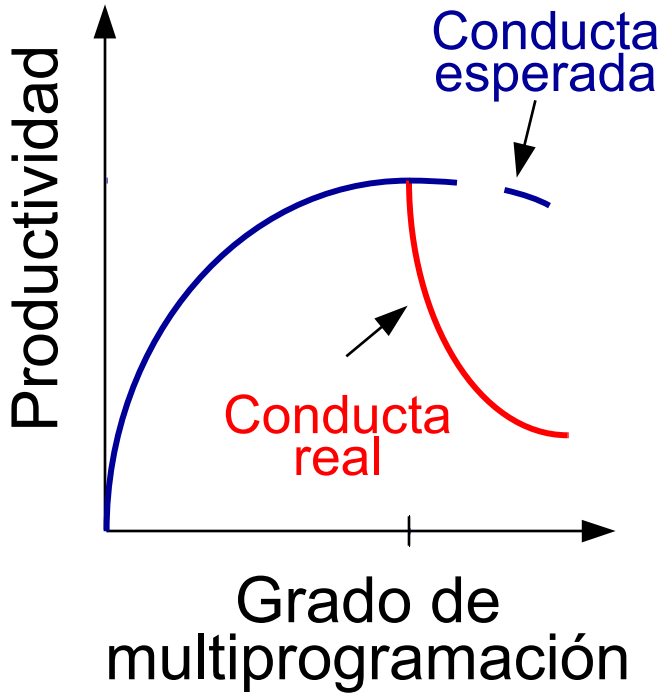


Teoría del Conjunto de Trabajo

- Un proceso sólo puede ejecutarse si su conjunto de trabajo está en memoria principal
- Una página no puede retirarse de memoria principal si está dentro del conjunto de trabajo del proceso en ejecución

Hiperpaginación

- Si un proceso no tiene "suficientes" páginas, la tasa de faltas es alta
  - » bajo uso de la CPU
  - » el SO incrementa el grado de multiprogramación
  - » más faltas de páginas
- *Hiperpaginación* = el sistema operativo está ocupado en resolver faltas de página



Hiperpaginación (y II)

- Formas de evitar la hiperpaginación:
  - » Asegurar que cada proceso existente tenga asignado un espacio en relación a su comportamiento **->** Algoritmos de asignación variables
  - » Actuar directamente sobre el grado de multiprogramación **->** Algoritmos de regulación de carga

Algoritmo basado en el modelo del WS

- En cada referencia, determina el conjunto de trabajo: páginas referenciadas en el intervalo **(t - x, t]** y **sólo** esas páginas son mantenidas en MP

  – En la figura se representan las páginas del proceso que están en MP
  – Proceso de 5 páginas
  – **x** = 4
  – En t=0 WS = {A, D, E}, A se referenció en t=0, D en t=-1 y E en t=-2

C, C, D, B, C,E, C,E,A,D

A	A	A	A	-	-	-	-	-	A	A
-	-	-	-	B	B	B	B	-	-	-
-	C	C	C	C	C	C	C	C	C	C
D	D	D	D	D	D	D	D	-	-	-
E	E	-	-	-	-	E	E	E	E	E
	*				*	*			*	*

Algoritmo FFP
(Frecuencia de Falta de Página)

- **Idea**: para ajustar el conjunto de páginas de un proceso, usa los intervalos de tiempo entre dos faltas de página consecutivas:
  - » Si intervalo de tiempo grande, mayor que Y, todas las páginas no referenciadas en dicho intervalo son retiradas de MP
  - » En otro caso, la nueva página es simplemente incluida en el conjunto de páginas residentes
- Formalmente:

t_c = instante de tº de la actual falta de página
t_{c-1}= instante de tº de la anterior falta de página
Z = conjunto de páginas referenciadas en un intervalo de tº
R = conjunto de páginas residentes en MP

$$R(t_c,Y)\begin{cases} Z(t_{c-1},t_c) & \text{si } t_c - t_{c-1} > Y \\ R(t_{c-1},Y) + Z(t_c) & \text{en otro caso} \end{cases}$$

Ejemplo del algoritmo FFP

- Se garantiza que el conjunto de páginas residentes crece cuando las faltas de página son frecuentes y decrece cuando no lo son

– En la figura se representan las páginas del proceso que están en MP
– Proceso de 5 páginas
– **Y** = 2
– La página A se referenció en t=-1 y E en t=-2
– D se referencia en t=0

D,C, C,D,B,C,E, C,E, A,D

A	A	A	A	-	-	-	-	-	A	A
-	-	-	-	B	B	B	B	B	-	-
-	C	C	C	C	C	C	C	C	C	C
D	D	D	D	D	D	D	D	D	-	D
E	E	E	E	-	-	E	E	E	E	E
*	*			*		*			*	*

3.4 Gestión de memoria en Linux

- Gestión de memoria a bajo nivel.
- El espacio de direcciones de proceso.
- La caché de páginas y la escritura de páginas a disco.

Gestión de memoria a bajo nivel

- La página física es la unidad básica de gestión de memoria: struct_page

```
struct page {
unsigned long flags;   // PG_dirty, PG_locked
atomic_t _count;
struct address_space *mapping;
void *virtual;
…
}
```

Gestión de memoria a bajo nivel

- Una página puede ser utilizada por:
  - La caché de páginas. El campo mapping apunta al objeto representado por addres_space.
  - Datos privados.
  - Una proyección de la tabla de páginas de un proceso.
  - El espacio de direcciones de un proceso.
  - Los datos del kernel alojados dinámicamente.
  - El código del kernel.

Gestión de memoria a bajo nivel

- Interfaces para la asignación de memoria en páginas.

struct page * alloc_pages(gfp_t gfp_mask, unsigned int order)

La función asigna 2^order páginas físicas contiguas y devuelve un puntero a la struct page de la primera página, y si falla devuelve NULL.

unsigned long __get_free_pages(gfp_t gfp_mask, unsigned int order)

Esta función asigna 2^order páginas físicas contiguas y devuelve la dirección lógica de la primera página.

Gestión de memoria a bajo nivel

- Interfaces para la liberación de memoria en páginas.

void __free_pages(struct page *page, unsigned int order)

void free_pages(unsigned long addr, unsigned int order)

Las funciones liberan 2^order páginas a partir de la estructura página o de la página que coincide con la dirección lógica.

Gestión de memoria a bajo nivel

- Interfaces para la asignación/liberación de memoria en bytes.

void * kmalloc(size_t size, gfp_t flags)

void kfree(const void *ptr)

- Las funciones son similares a las que proporciona C en espacio de usuario malloc() y free().

Ejemplo de código kernel

- Asignación/liberación de memoria en páginas.

```
unsigned long page;
page = __get_free_pages(GFP_KERNEL, 3);
/* 'page' is now the address of the first of eight contiguous
pages ... */
free_pages(page, 3);
/* our pages are now freed and we should no longer access the
address stored in 'page'
*/
```

Ejemplo de código kernel

- Asignación/liberación de memoria en bytes.

```
struct example *p;
p = kmalloc(sizeof(struct example), GFP_KERNEL);
if (!p)
/* handle error ... */
kfree(p);
```

Gestión de memoria a bajo nivel

- Zonas de memoria. El tipo **gfp_t** permite especificar el tipo de memoria que se solicita mediante tres categorías de flags:
  - Modificadores de acción (GFP_WAIT, GFP_IO)
  - Modificadores de zona. (GFP_DMA)
  - Tipos (especificación más abstracta).
- Ejemplos de solicitud de tipos de memoria:
  - GFP_KERNEL indica una solicitud de memoria para kernel.
  - GFP_USER permite solicitar memoria para el espacio de usuario de un proceso.

Caché de bloques. Organización.

- La asignación y liberación de estructuras de datos es una de las operaciones más comunes en un kernel de SO. Para agilizar esta solicitud/liberación de memoria Linux usa el **nivel de bloques** (slab layer).
- El nivel de bloques actua como un nivel de caché de estructuras genérico.
  - Existe una caché para cada tipo de estructura distinta: pej. task_struct caché o inode caché.
  - Cada caché contiene multiples bloques constituidos por una o más páginas físicas contiguas.
  - Cada bloque aloja estructuras del tipo correspondiente a la caché.

Caché de bloques. Funcionamiento.

- Cada bloque puede estar en uno de tres estados: lleno, parcial o vacío.
- Cuando el kernel solicita una nueva estructura:
  - La solicitud se satisface desde un bloque parcial si existe alguno.
  - Si no, se satisface a partir de un bloque vacío.
  - Si no existe un bloque vacío para ese tipo de estructura, se crea uno nuevo y la solicitud se satisface en este nuevo bloque.

Espacio de direcciones de proceso

- Espacio de direcciones de proceso es el espacio de direcciones de los procesos ejecutándose en modo usuario. Linux utiliza memoria virtual (VM).
- A cada proceso se le asigna un espacio de memoria plano de 32 o 64 bits único. No obstante se puede compartir el espacio de memoria (CLONE_VM para hebras).
- El proceso solo tiene permiso para acceder a determinados intervalos de direcciones de memoria, denominados **áreas de memoria**.

Espacio de direcciones de proceso

¿Qué puede contener un área de memoria?

- Un mapa de memoria de la sección de código (text section).
- Un mapa de memoria de la sección de variables globales inicializadas (data section).
- Un mapa de memoria con una proyección de la página cero para variables globales no inicializadas (bss section)
- Un mapa de memoria con una proyección de la página cero para la pila de espacio de usuario.

Espacio de direcciones de proceso

- El descriptor de memoria representa en Linux el espacio de direcciones de proceso.

```
struct mm_struct {

struct vm_area_struct *mmap;  /*Lista de áreas de memoria (VMAs)*/

struct rb_root mm_rb;     /* árbol red-black de VMAs, para buscar un elemento concreto */

struct list_head mmlist;  /* Lista con todas las mm_struct: espacios de direcciones */

atomic_t mm_users; /* Número de procesos utilizando este espacio de direcciones */

atomic_t mm_count; /* Contador que se activa con la primera referencia al espacio de direcciones y se desactiva cuando mm_users vale 0 */
```

Espacio de direcciones de proceso

```
/*(cont. struct mm_struct) Límites de las secciones principales */

unsigned long start_code; /* start address of code */
unsigned long end_code; /* final address of code */
unsigned long start_data; /* start address of data */
unsigned long end_data; /* final address of data */
unsigned long start_brk; /* start address of heap */
unsigned long brk; /* final address of heap */
unsigned long start_stack; /* start address of stack */
unsigned long arg_start; /* start of arguments */
unsigned long arg_end; /* end of arguments */
unsigned long env_start; /* start of environment */
unsigned long env_end; /* end of environment */

 /* Información relacionada con páginas */
pgd_t *pgd; /* page global directory */
unsigned long rss; /* pages allocated */
unsigned long total_vm; /* total number of pages */
}
```

Espacio de direcciones de proceso

¿Cómo se asigna un descriptor de memoria?

- Copia del descriptor de memoria al ejecutar fork().
- Compartición del descriptor de memoria mediante el flag CLONE_VM de la llamada clone().

¿Cómo se libera un descriptor de memoria?

- El núcleo decrementa el contador mm_users incluido en mm_struct. Si este contador llega a 0 se decrementa el contador de uso mm_count. Si este contador llega a valer 0 se libera la mm_struct en la caché.

Espacio de direcciones de proceso

Un área de memoria (**struct vm_area_struct**) describe un intervalo contiguo del espacio de direcciones.

```
struct vm_area_struct {

struct mm_struct *vm_mm; /* struct mm_struct asociada que representa el espacio de direcciones */

unsigned long vm_start; /* VMA start, inclusive */
unsigned long vm_end; /* VMA end , exclusive */
unsigned long vm_flags; /* flags */
struct vm_operations_struct *vm_ops; /* associated ops */
struct vm_area_struct *vm_next; /* list of VMA's */
struct rb_node vm_rb; /* VMA's node in the tree */
};
```

Ejemplo de espacio de direcciones

Utilizando el archivo /proc/<pid>/maps podemos ver las VMAs de un determinado proceso.

El formato del archivo es:

start-end permission offset major:minor inode file

start-end.Dirección de comienzo y final de la VMA en el espacio de direcciones del proceso.
permission. Describe los permisos de acceso al conjunto de páginas del VMA. {r,w,x,-}{p|s}
offset. Si la VMA proyecta un archivo indica el offset en el archivo, si no vale 0.
major:minor. Se corresponden con los números major,minor del dispositivo en donde reside el archivo.
inode: Almacena el número de inodo del archivo.
file: El nombre del archivo.

Ejemplo de espacio de direcciones

```
int main(int argc, char *argv[])
{
while (1);
return 0;
}
```

aleon@aleon-laptop:~$ cat /proc/2263/maps

```
00400000-00401000 r-xp 00000000 08:06 5771454                    /home/aleon/vmareas
00601000-00602000 rw-p 00001000 08:06 5771454                    /home/aleon/vmareas
7f06e683a000-7f06e69b7000 r-xp 00000000 08:05 1332123            /lib/libc-2.11.1.so
7f06e6bb6000-7f06e6bba000 r--p 0017c000 08:05 1332123            /lib/libc-2.11.1.so
7f06e6bba000-7f06e6bbb000 rw-p 00180000 08:05 1332123            /lib/libc-2.11.1.so
7f06e6bc0000-7f06e6be0000 r-xp 00000000 08:05 1332125            /lib/ld-2.11.1.so
7f06e6ddf000-7f06e6de0000 r--p 0001f000 08:05 1332125            /lib/ld-2.11.1.so
7f06e6de0000-7f06e6de1000 rw-p 00020000 08:05 1332125            /lib/ld-2.11.1.so
7fffd3dc3000-7fffd3dd8000 rw-p 00000000 00:00 0                  [stack]
```

Espacio de direcciones de proceso

¿Cómo se crea un intervalo de direcciones?

- do_mmap() permite:
  - Expandir un VMA ya existente (porque el intervalo que se añade es adyacente a uno ya existente y tiene los mismos permisos)
  - Crear una nueva VMA que represente el nuevo intervalo de direcciones

unsigned long do_mmap(struct file *file, unsigned long addr,
unsigned long len, unsigned long prot,
unsigned long flag, unsigned long offset)

Espacio de direcciones de proceso

¿Cómo se elimina un intervalo de direcciones?

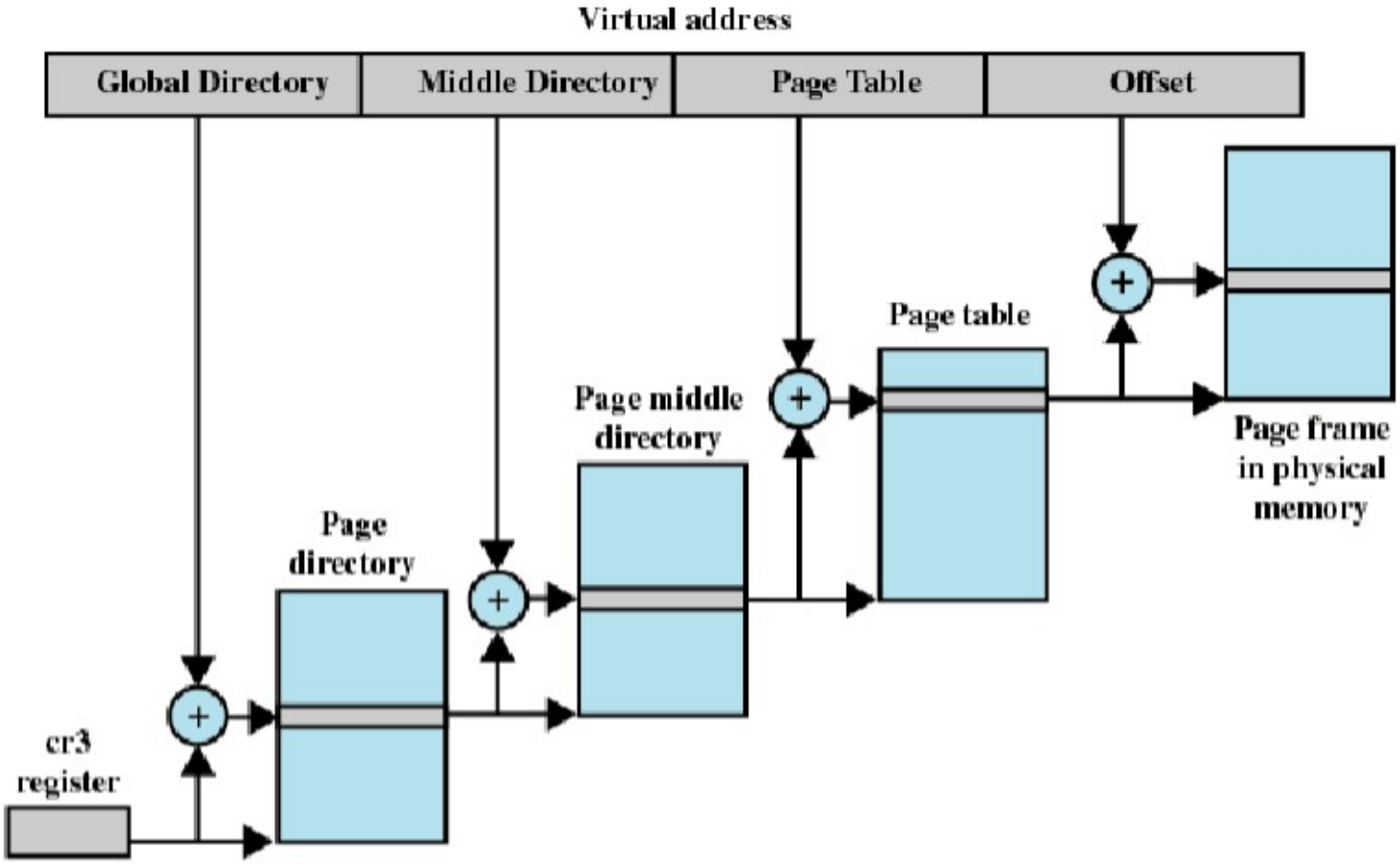
- do_munmap() permite eliminar un intervalo de direcciones. El parámetro mm especifica el espacio de direcciones del que se va a eliminar el intervalo de memoria que comienza en "start" y tiene una longitud de "len" bytes.

int do_munmap(struct mm_struct *mm, unsigned long start,
size_t len)

Espacio de direcciones de proceso

- Las direcciones virtuales deben convertirse a direcciones físicas mediante tablas de páginas. En linux tenemos 3 niveles de tablas de páginas.
  - La tabla de páginas de más alto nivel es el directorio global de páginas (del inglés page global directory, PGD), que consta de un array de tipo "pgd_t".
  - Las entradas del PGD apuntan a entradas de la tabla de páginas de segundo nivel (page middle directory, PMD), que es un array de tipo "pmd_t".
  - Las entradas del PMD apuntan a entradas en la PTE. El último nivel es la tabla de páginas y contiene entradas de tabla de páginas del tipo "pte_t" que apuntan a páginas físicas: struct_page.

Tablas de páginas multinevel de Linux



Caché de páginas. Conceptos.

- La caché de páginas está constituida por páginas físicas de RAM, y los contenidos de éstas se corresponden con bloques físicos de disco.
- El tamaño de la caché de páginas es dinámico.
- El dispositivo sobre el que se realiza la técnica de caché se denomina almacén de respaldo (backing store).
- Lectura/Escritura de datos de/a disco.
- Fuentes de datos para la caché: archivos regulares, de dispositivos y archivos proyectados en memoria.

Desalojo de la caché de páginas.

- Proceso por el cual se eliminan datos de la caché junto con la estrategia para decidir cuáles datos eliminar.
- Linux selecciona páginas limpias (no marcadas PG_dirty) y las reemplaza con otro contenido.
- Si no existen suficientes páginas limpias en la caché, el kernel fuerza un proceso de escritura a disco para hacer disponibles más páginas limpias.
- Ahora queda por decidir que páginas limpias seleccionar para eliminar (selección de víctima).

Selección de víctima.

- Least Recently Used (LRU). Requiere mantener información de cuando se accede a cada página y seleccionar las páginas con el tiempo más antiguo. El problema es el acceso a archivos una sola vez.
- Linux soluciona el problema usando dos listas pseudo-LRU: active list e inactive list.
  - Las páginas de la active list no pueden ser seleccionadas como víctimas y se añaden páginas accedidas solamente si residen en la inactive list.
  - Las páginas de la inactive list pueden ser seleccionadas como víctimas.

Caché de páginas. Operaciones.

- Una página puede contener varios bloques de disco posiblemente no contiguos (depende del método de asignación de bloques a archivos).
- La caché de páginas de Linux usa una estructura para gestionar entradas de la caché y operaciones de E/S de páginas: address_space.
- Lectura/escritura de páginas de/en caché.
- Hebras de escritura retardada.