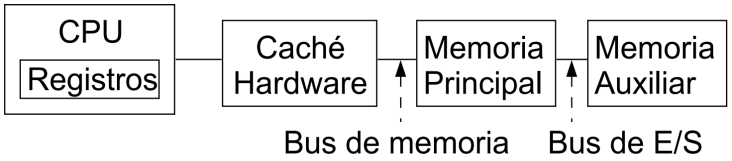
Enrique López García

SISTEMAS OPERATIVOS

TEMA 3. Gestión de Memoria.

Gestión de memoria principal

Jerarquía de Memoria



Dos **principios sobre memoria**:

- Menor cantidad ⇒ acceso más rápido.
- Mayor cantidada ⇒ menor coste por byte

Por tanto, los elementos frecuentemente accedidos se ponen en **memoria rápida** (cara y pequeña) y el resto en **memoria lenta** (grande y barata).

Memoria Caché

Definición. **Copia** de la memoria que puede ser accedida **más rápidamente** que la original.

Objetivo. Hacer los **casos frecuentes eficientes** (los caminos infrecuentes no importan tanto).

Conceptos.

- **Acierto de caché**: acceder a un item que está en la caché.
- **Fallo de caché**: acceder a un item que no está en la caché, por lo que hay que hay que acceder a memoria.
- **Tiempo de Acceso Efectivo** (**TAE**) = ProbabilidadAcierto * CosteAcierto + ProbabilidadFallo * CosteFallo
- Funciona porque los programas no son aleatorios, explotan la localidad → **principio de localidad**

Espacio de Direcciones Lógico y Físico

Imagen de proceso = mapa + PCB

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.

Objetivos Generales de la Gestión de Memoria

Organización. Cómo se divide la memoria de uno o varios procesos.

Gestión. Dado un esquema de organización, qué estrategias seguir para obtener un rendimiento óptimo.

- Estrategias de **asignación**. Contigua (huecos al liberar memoria) o No Contigua (acceso menos eficiente)
- Estrategias de **sustitución** o **reemplazo**.
- Estrategias de **búsqueda** o **recuperación**.

Protección...

... del SO respecto de los procesos de usuario.
... de los procesos de usuario entre ellos.

Asignación Contigua

La asignación de almacenamiento para un programa se hace en **un único bloque de posiciones contiguas** de memoria. Tipos de particiones:

- **Particiones Fijas**. Mismo tamaño para cada proceso ⇒ Si uno lo supera no puede ejecutarse.

- **Particiones variables**. Tamaño según la necesidad de cada proceso.

La gran desventaja es que se va fragmentando la memoria quedando pequeños huecos de memoria sin usar pues los nuevos programas pueden no caber en dichos huecos.

Asignación No Contigua

Permiten dividir el programa en **bloques o segmentos** que se pueden colocar en zonas **no necesariamente contiguas** de memoria principal.

Tipos: Paginación, Segmentación y Segmentación Paginada

Intercambio (Swapping)

Idea. Intercambiar procesos entre memoria (principal) y un almacenamiento auxiliar (secundario)

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**.

Intercambiador

- Seleccionar procesos para **retirarlos de MP**.
- Seleccionar procesos para **incorporarlos a MP**.
- **Gestionar y asignar** el **espacio** de intercambio.

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** → Parte del proceso necesarios en un momento dado. (RAM)
- **Memoria Secundaria** → Espacio de direcciones completo del proceso. (SSD)

Es necesario:

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

Además, la memoria virtual

- evita que el proceso se dé cuenta de dichas particiones (MP y MS) → evitar tiempos prolongados.
- resuelve el problema del crecimiento dinámico de los procesadores,
- permite aumentar el grado de multiprogramación.

Unidad de Gestión de Memoria (MMU - Memory Management Unit)

Es un dispositivo **hardware** gestionado por el SO que **traduce direcciones virtuales a direcciones físicas**.

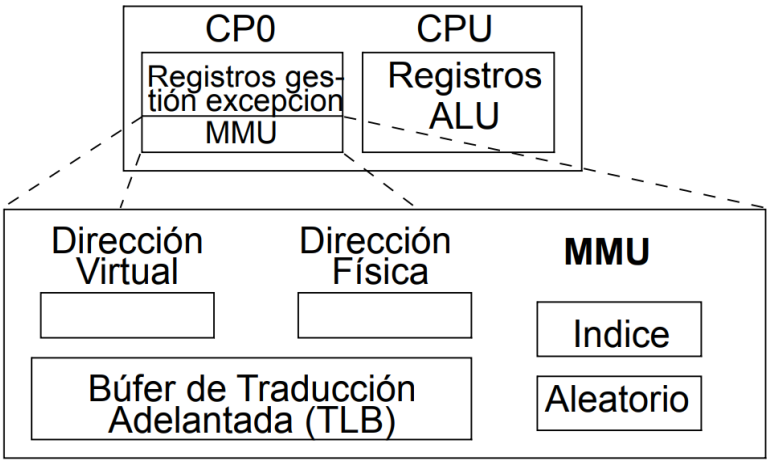
- Los programas de usuario tratan sólo con direcciones lógicas, nunca con direcciones reales.

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

Si la tabla de índices está en MP, habría que hacer dos accesos para acceder a la posición de memoria. Una solución a este problema es habilitar una **caché o buffer** donde se **guarda la conversión** de una dirección **lógica a física** (se puede apreciar en el dibujo de ejemplo).

Además en cada cambio de contexto se cambia de posición de memoria a la que acceder, por lo que estamos viendo que es una acción cada vez más costosa.



Paginación

Con este método

- 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.**
- En los marcos de páginas se situarán las páginas de los procesos cuando estos se carguen en memoria.

En relación al **tamaño de los marcos de página**...

- Cuantos **más pequeños** sean ➡ menos huecos de memoria sin usar pero más tiempo de acceso a un proceso completo.
- Cuantos **más grandes** sean ➡ más memoria contigua y mejor velocidad de acceso pero más huecos en memoria

Las **direcciones lógicas**, que son las que genera la CPU se dividen en número de página (**p**) y desplazamiento dentro de la página (**d**) ~ se intenta que sean múltiplo de dos.

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

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.

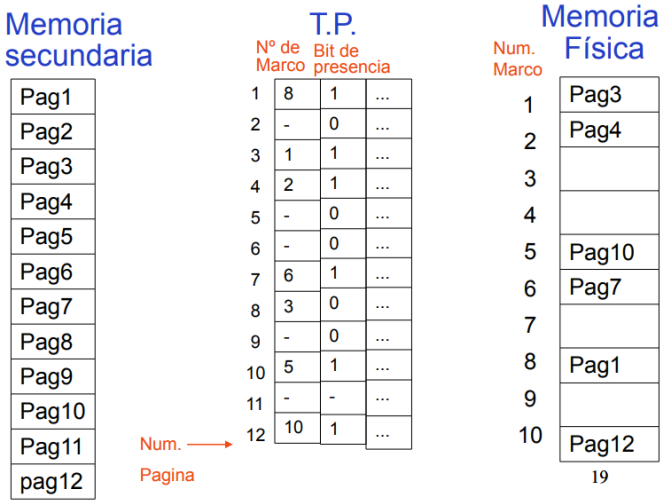
También existen otras tablas como:

- La **tabla de ubicación en disco** (una por proceso) que guarda la ubicación de cada página en el almacenamiento secundario.
- La **tabla de marcos de página**, usada por el SO 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 (1 si está en MP, 0 en caso contrario)
- **Bit de modificación**. Indica si se ha modificado la página para que al sacarla de MP se vuelva a escribir.
- **Modo de acceso** autorizado a la página (bits de protección). Permiso de lectura, escritura...



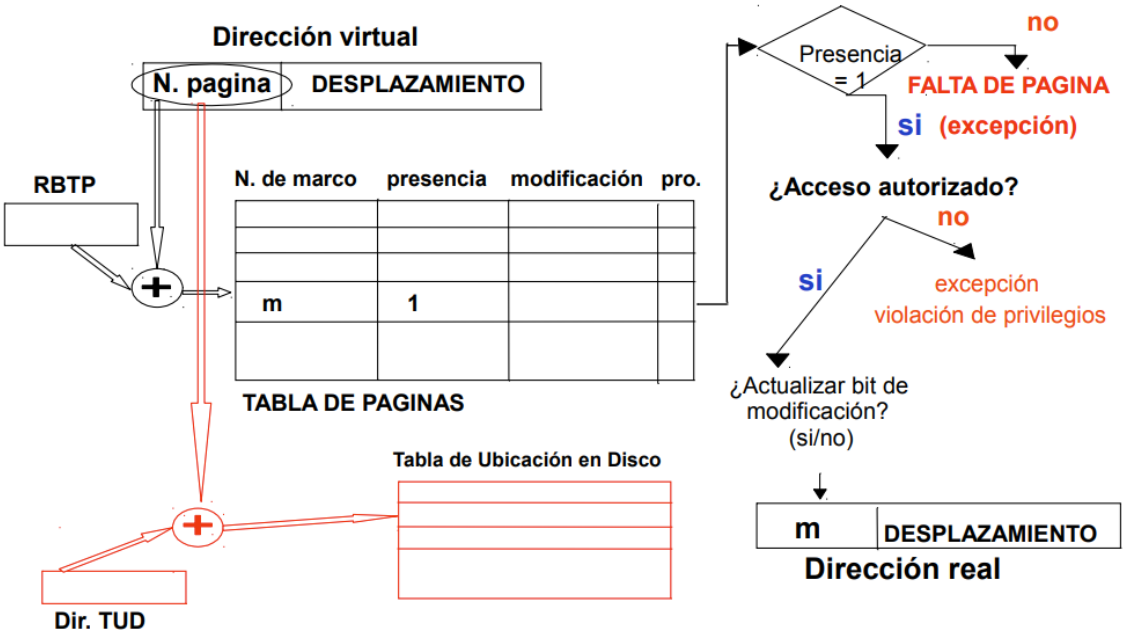
Ventaja

Pueden haber muchos procesos cargados independientemente de su tamaño total.

Desventaja

Menos eficiente al acceder a memoria (discontinua).

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 de presencia=1, nº de 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 MP. El **registro base de la tabla de páginas** (RTBP) apunta a la Tabla de Páginas (suele almacenarse en el PCB del proceso. En 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 el TLB (búfer de traducción anticipada)
- Un problema adicional viene determinado por el tamaño de la Tabla de Páginas ➡ si un proceso tiene muchas páginas, la Tabla de Páginas ocupa mucho espacio

EJEMPLO: Tamaño de la Tabla de Páginas

Si un sistem que usa Paginación tiene los siguientes datos:

- **Dirección virtual**: 32 bits
- **Tamaño de página** = 4 KB = 2^{12} B

Entonces el **tamaño** del campo **desplazamiento** es 12 bits, y el tamaño del **número de página virtual** es 20 bits.

Eso quiere decir que el número de páginas virtuales es 2^{20}, esto es, ¡¡1.048.576!!

Solución para reducir el tamaño de la TP: **PAGINACIÓN MULTINIVEL**

Paginación Multinivel

Concepto

Se basa en la idea de **paginar las tablas de páginas**. Con esto se consigue que **se cargue sólo la porción de la tabla de páginas que está en uso** y se deje sin cargar aquellos espacios de direcciones que no se usan.

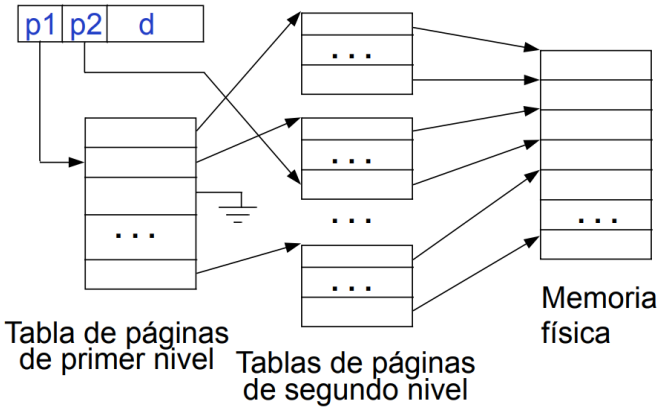
Paginación a dos niveles

Lo que hacemos es **dividir la tabla de páginas en partes del tamaño de una página**. Por lo que la dirección lógica se divide en:

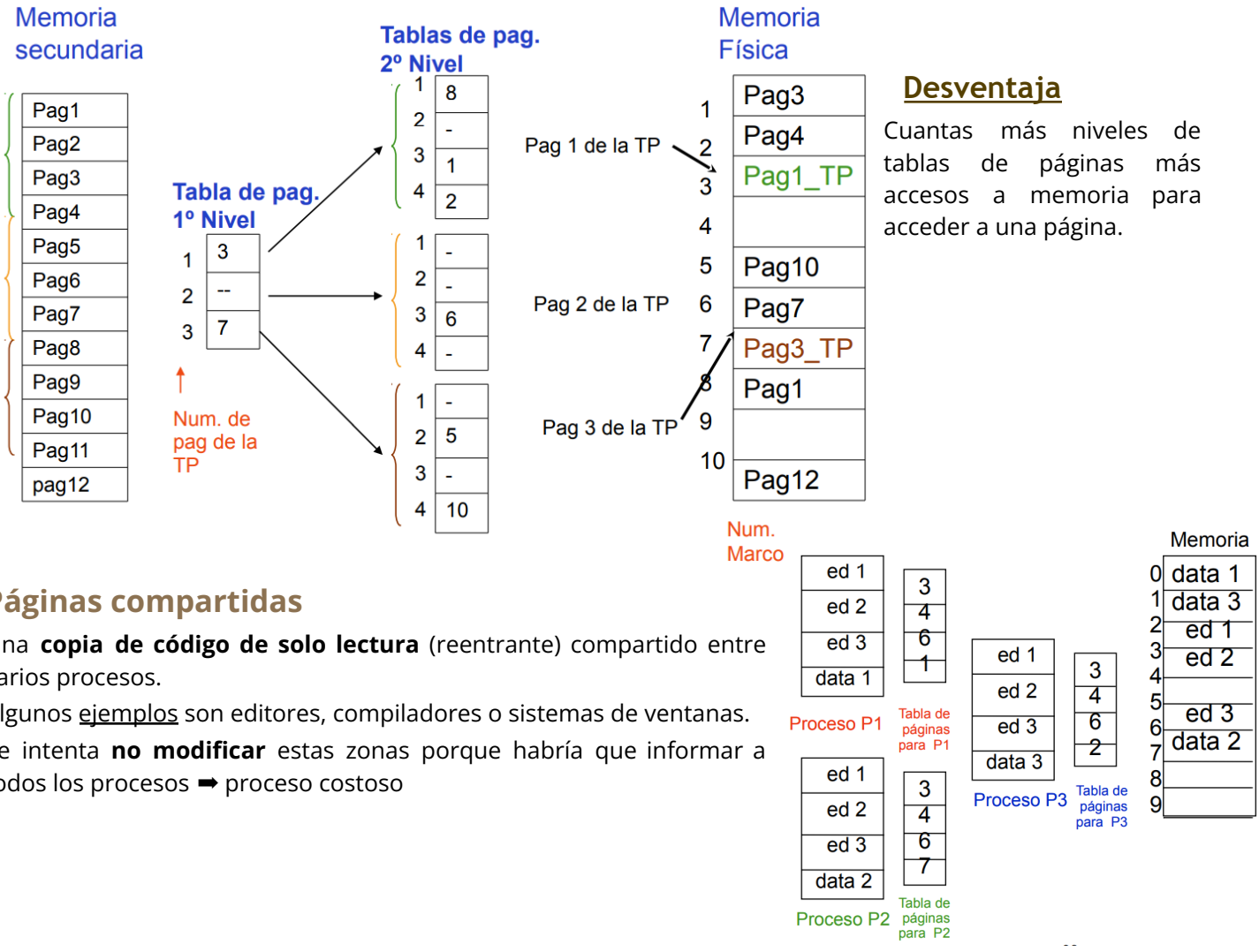
- Número de página (n bits)
  - Un número de página p1 (=k)
  - Desplazamiento de páglna p2 (=n-k)
- Desplazamiento de página d (m bits)

Por lo tanto una dirección lógica es de la forma:

p1	p2	d



EJEMPLO



Desventaja

Cuantas más niveles de tablas de páginas más accesos a memoria para acceder a una página.

Páginas compartidas

Una **copia de código de solo lectura** (reentrante) compartido entre varios procesos.

Algunos ejemplos son editores, compiladores o sistemas de ventanas.

Se intenta **no modificar** estas zonas porque habría que informar a todos los procesos ➡ proceso costoso

Segmentación

Esquema de organización de memoria que soporta mejor la visión de memoria del usuario: un programa es una colección de unidades lógicas -segmentos-

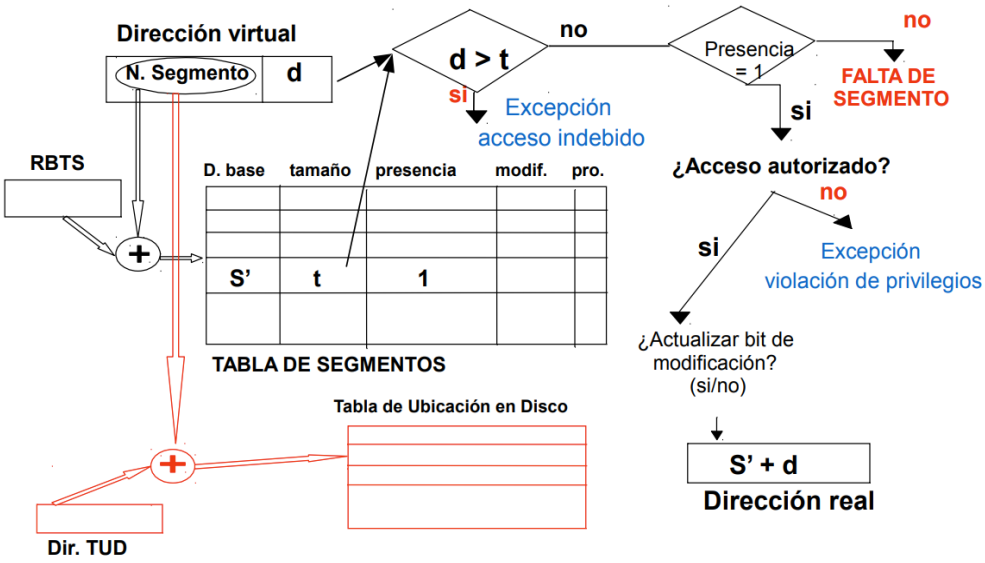
Tabla de Segmentos

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

Esquema de traducción



Implementación de la Tabla de Segmentos

La tabla de segmentos se mantiene **en MP**. El **Registro Base de la Tabla de Segmentos** (RTBS) apunta a la tabla de segmentos . El **Registro Longitud de la Tabla de Segmentos** (STRL) indica el número de segmentos del proceso. Ambos suelen almacenarse en el PCB del proceso). Además, el nº de segmento de una dirección lógica es legal si es menor que STRL.

Segmentación Paginada

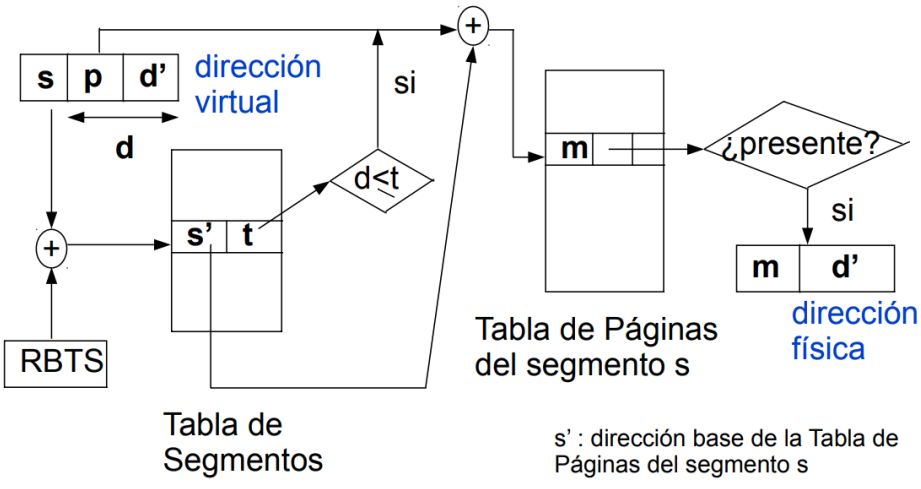
Contexto

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, dando lugar a la **Segmentación Paginada**.

Esquema de traducción



Esta consiste en que cada proceso s**e divide en segmentos** de diferentes tamaños, pero luego los segmentos están **distribuidos en páginas** del mismo tamaño. Por lo que la gestión de la memoria es mucho más sencilla.

Memoria Virtual: Gestión

Vamos a tratar la gestión relativa a la paginación, pero se siguen los mismos critetios con la segmentación.

Criterios de clasificación

Para la clasificación habrá que seguir diversas políticas:

- ➢ **Políticas de asignación**: Fija o Variable
- ➢ **Políticas de búsqueda (recuperación)**:
  - Paginación por demanda
  - Paginación anticipada (!= prepaginación)
- ➢ **Políticas de sustitución (reemplazo)**:
  - Sustitución local
  - Sustitución global

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 ➡ las páginas modificadas necesitan acceder a memoria a la hora de ser sustituidasç
- ➢ **Páginas compartidas**
  - Se pretende reducir el nº de faltas de página ➡ se intenta no sacar de memoria
- ➢ **Páginas especiales**
  - Algunos marcos pueden estar bloqueados (ejemplo: buffers 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 transferencias MP ➡ Disco
  - Reducen la fragmentación interna
  - Aumenta la probabilidad del fallo de página
- Cuanto más **grandes**
  - Grandes cantidades de información que no serán usadas están ocupando MP
  - Aumenta la fragmentación interna

Por lo tanto **se busca 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

Óptimo

Se sustituye la página que no se va a referenciar en un futuro o la que se referencie más tarde.

Dicho algoritmo es casi imposible, pues es muy difícil saber qué se va a ejecutar con análisis estáticos.

FIFO

Se sustituye la **página más antigua**

LRU

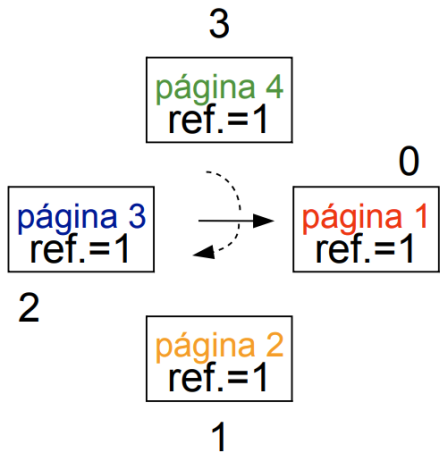
Se sustituye la página que fue objeto de la **referencia más antigua**. Es el más habitual.

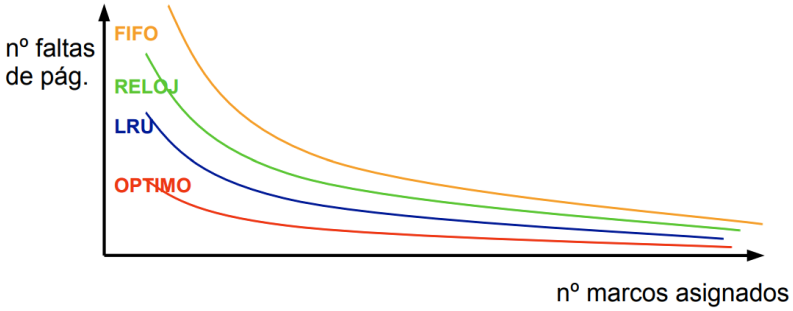
Algoritmo de reloj

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.

Pasos en una selección de página:

1. Consultar el marco actual
2. ¿Es R=0?
   - NO. Se pone R=0, ir al siguiente marco y volver al paso 1.
   - SI. Seleccionar para sustituir e incrementar posición





Comparación

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 una página que realiza el proceso. Es importante para maximizar el rendimiento del sistema de memoria virtual.

Propiedad de localidad

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

Mientras el conjunto de páginas necesarias puedan residir en MP, el nº de faltas de página no crece mucho. Pero si páginas de ese conjunto son eliminadas, la activación de paginación crece mucho. Para ello se define:

Conjunto de trabajo (Working Set): 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

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

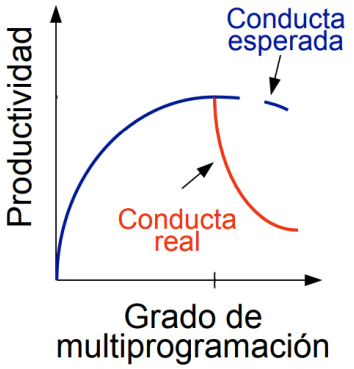
Un proceso sólo puede ejecutarse si su conjunto de trabajo está en MP. Por lo tanto una página no puede retirarse de MP si está dentro del conjunto de trabajo del proceso en ejecución.

Hiperpaginación

Si un proceso no tiene "suficientes" páginas, la tasa de faltas de página es alta, esto provoca:

- Bajo uso de la CPU
- El SO incrementa el grado de multiprogramación
- Más faltas de páginas

Dando lugar a la hiperpaginación: el sistema operativo está ocupado en resolver faltas de página



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. Veamos un ejemplo:

- 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 regerención en t=0, D en t=-1 y E en t=-2

Algoritmo FFP (Frecuencia de Falta de Página)

Idea: para ajustar el conjunto de páginas de un proceso, se usan los intervalos de tiempo entre dos faltas de página consecutivas:

- Si el intervalo de tiempo es grande (mayor que Y) todas las páginas no referenciadas en dicho intervalo son retiradas de MP ➡ como si se hubiese cambiado de Conjunto de Trabajo
- 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

R = conjunto de páginas residentes en MP

Z = conjunto de páginas referenciadas en un intervalo de tº

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

Hay faltas de página en t=1, t=4, t=6, t=9, t=10

EJEMPLO

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

3.4 Gestión de memoria en Linux

Gestión de memoria bajo nivel

La **página física** es la **unidad básica** de gestión de memoria: 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.

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

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.

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.

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().

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

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’
```

Asignación/liberación de memoria en bytes

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

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 actúa 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 múltiples bloques constituidos por una o más páginas físicas contiguas.
- Cada bloque aloja estructuras del tipo correspondiente a la caché.

Funcionamiento

Cada bloque puede estar en uno de **tres estados**: lleno, parcial o vacío.

Cuando el kernel **solicita una nueva estructura**:

1. La solicitud se satisface desde un bloque parcial si existe alguno.
2. Si no, se satisface a partir de un bloque vacío.
3. 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 un 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.** Un área de memoria puede contener:

- 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.

Descriptor de memoria

El **descriptor de memoria** representa en Linux el espacio de direcciones de proceso: **`mm_struct.`**

Un descriptor de memoria se puede **asignar** de las siguientes maneras:

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

Y al **liberarlo** el núcleo decrementa el contador **`mm_users`**, incluido en `mm_struct`. Si este llega a 0 se decrementa el contador de uso **`mm_count`**. Si este contador llega a 0 se libera la mm_struct en la caché.

Área de memoria

Un **área de memoria** describe un intervalo contiguo del espacio de direcciones: **`struct vm_area_struct`**

Un intervalo de direcciones se puede crear con:

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

que permite:

- *Expandir un VMA ya existente (con los mismos permisos)*
- *Crear una nueva VMA que represente el nuevo intervalo de direcciones*

Y se puede eliminar con

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

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.

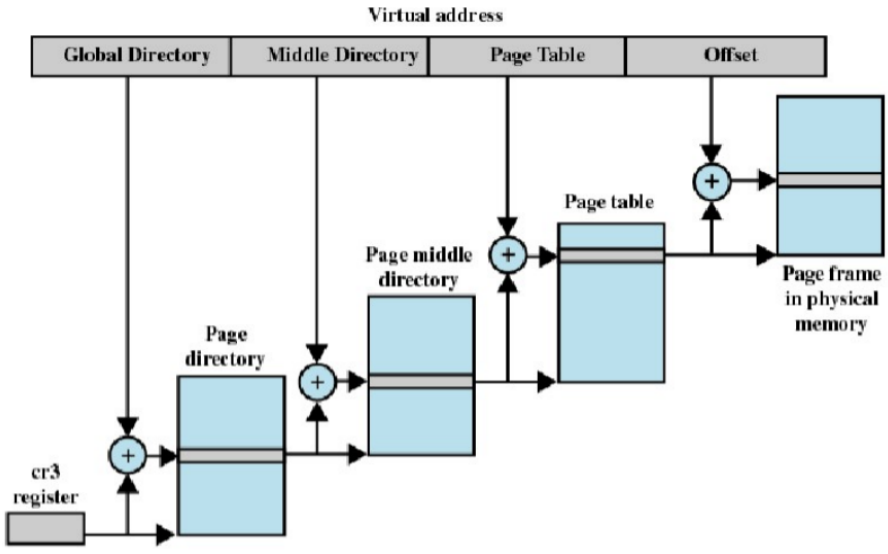
Tablas de Páginas

Las direcciones virtuales deben convertirse a direcciones físicas mediante tablas de páginas. En linux tenemos **3 niveles** de tablas de páginas.

I. La tabla de páginas de más alto nivel es el **directorio global** de páginas (page global directory, PGD), que consta de un array de tipo **`pgd_t`**.

II. 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`**.

III. 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`**.

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), se encarga de la 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

Este es un proceso por el cual se **eliminan datos de la caché** junto con la **estrategia** para **decidir cuáles** datos eliminar. Linux sigue la siguiente estructura:

- 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 qué páginas limpias seleccionar para eliminar (selección de víctima).

Selección de víctima

Algoritmo **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.

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.