CAPÍTULO III: ADMINISTRACIÓN DE MEMORIA

ÍNDICE

Bibliografía

[MORERA.95] Morera Pascual J., Pérez Campanero J. A., *Teoría y diseño de los sistemas operativos*. Anaya Multimedia, 1995.
[TANENB.03] Tanenbaum A. S., *Sistemas operativos modernos*. Prentice-Hall, 2003.
[STALLI.01] Stallings, William. *Sistemas operativos*. Prentice-Hall 2001.
[CARRET 01] Jesús Carretero, Félix García, Pedro De Miguel, Fernando Pérez. *Sistemas operativos*. McGraw-Hill, 2001.

1. Introducción

La memoria es uno de los recursos más importantes en un sistema de multiprogramación.

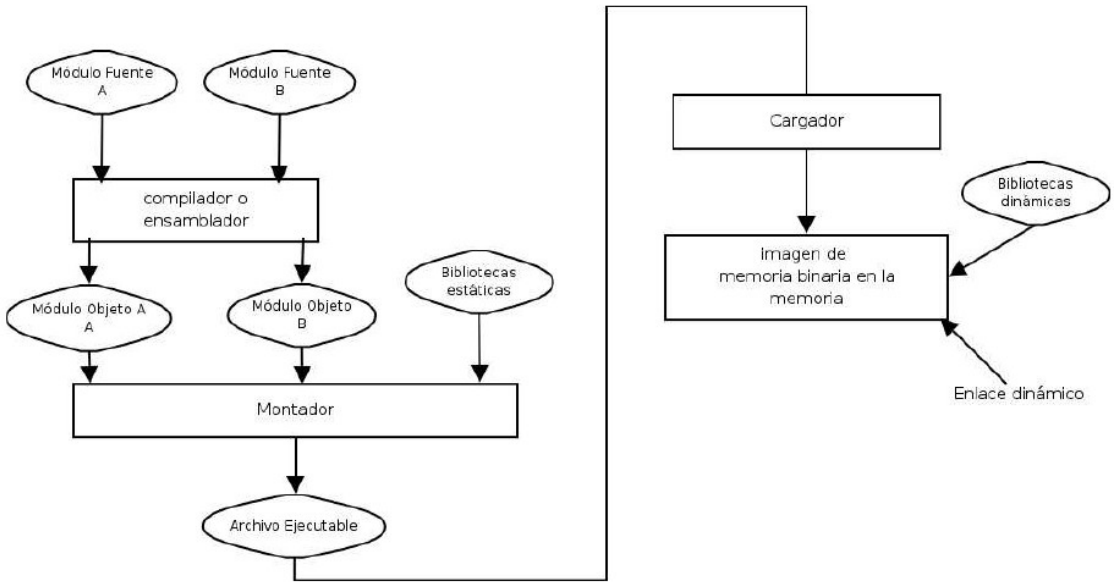
Ningún programa puede ejecutarse en un ordenador sin que, previamente, haya sido cargado en la memoria principal (MP).

Para poder ejecutar más de un proceso a la vez necesitaremos mantener dichos procesos simultáneamente en memoria ⇒ deberán compartirla.

La tarea de subdividir la memoria la lleva a cabo dinámicamente el SO y se conoce como gestión de memoria.

2. Fases de la carga de un programa

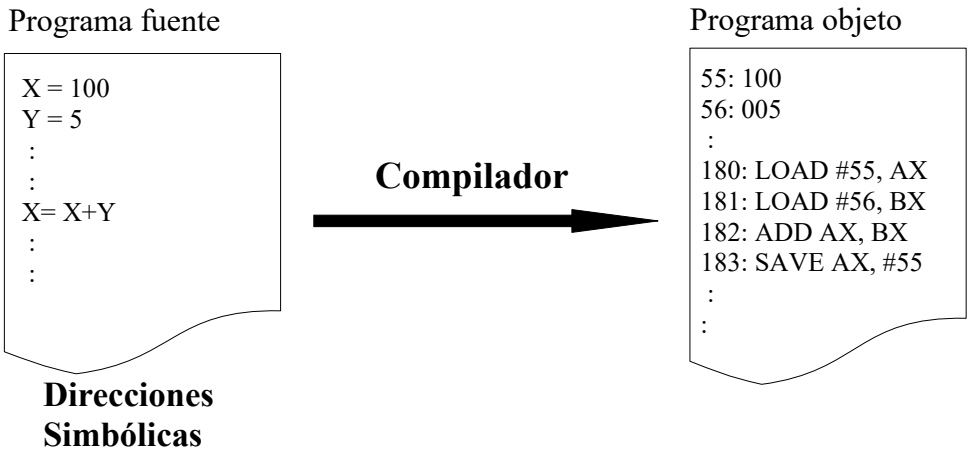
Desde que un programa se desarrolla hasta que es cargado en memoria y se ejecuta pasa por una serie de fases:



Compilación

Se genera el código máquina correspondiente a cada módulo fuente ⇒ asociar direcciones relativas o reutilizables a símbolos (las variables) y resolver referencias dentro de cada módulo.

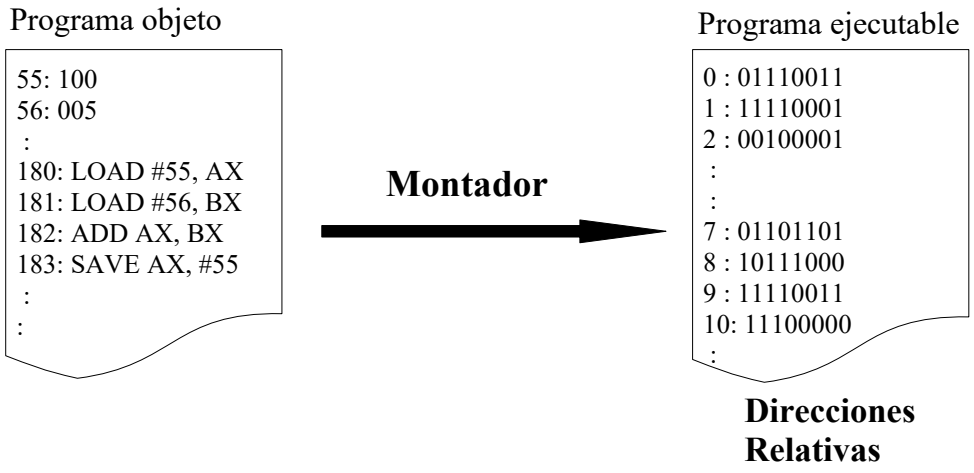
El resultado es un módulo objeto.



Montaje o enlace

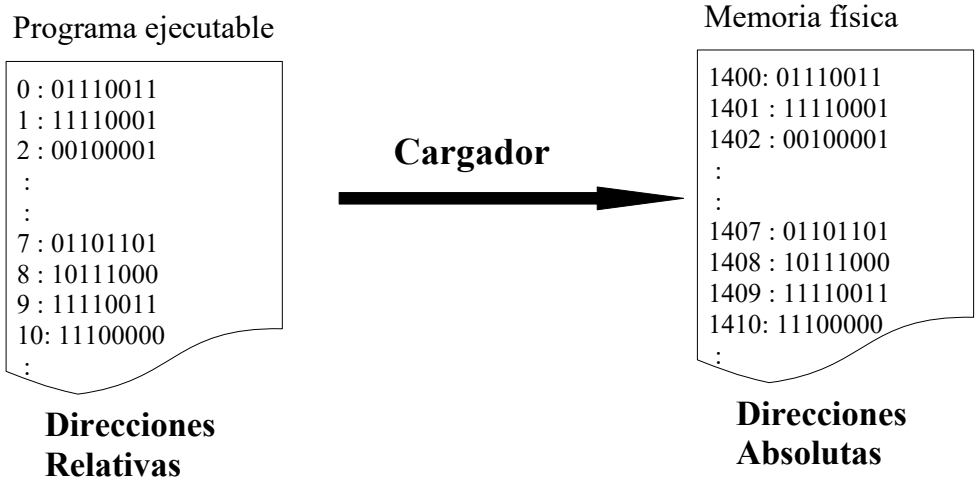
Se agrupan todos los archivos objeto pasando de direcciones simbólicas a relativas y resolviendo referencias entre módulos.

El resultado final es un fichero ejecutable:



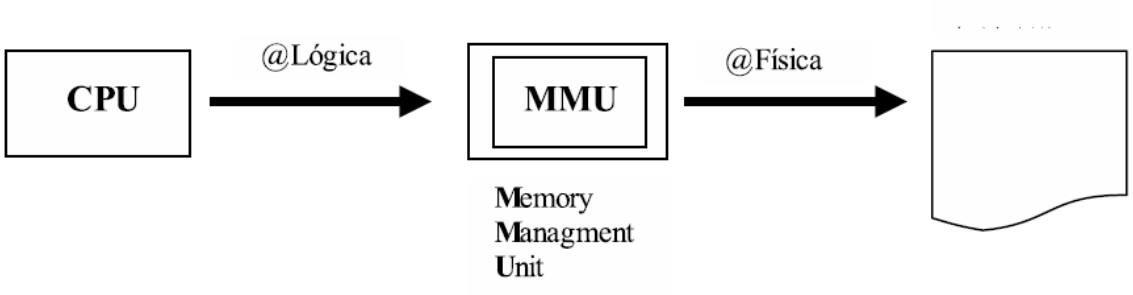
Carga

Para que un programa se ejecute es necesario que se cargue en MP, hay una traducción de direcciones relativas a direcciones físicas o absolutas.



Ejecución

Durante la ejecución del programa, la CPU lanzará direcciones de memoria lógicas y la Unidad de Administración de Memoria (MMU) transformará estas direcciones lógicas en direcciones físicas:



El SO deberá poder indicarle a la MMU qué función de traducción debe aplicarse al proceso.

Para un programa se suele hablar de:

- **Espacio lógico del programa**: conjunto de direcciones lógicas válidas para un determinado programa ejecutable.
- **Espacio lógico del procesador**: conjunto de direcciones lógicas válidas que puede lanzar un procesador.
- **Espacio físico del programa**: conjunto de direcciones físicas válidas para el programa.

3. Características de los gestores de memoria

Monoprogramados/Multiprogramados: se permite sólo un proceso a la vez cargado en memoria o más de uno.

Residentes/No residentes: el proceso debe estar en la memoria durante toda su ejecución o se permite al proceso salir y volver a cargarse.

Inmóvil/Móvil: el proceso no puede cambiar de lugar en la memoria durante su ejecución o puede cargarse en otra zona de memoria y continuar ejecutándose.

Contiguo/No contiguo: el proceso ha de estar cargado en posiciones consecutivas de memoria o no es necesario.

Entero/No entero: el proceso ha de estar cargado completo en la memoria o no es imprescindible.

Los modelos más sencillos de Administración de Memoria cumplían las características de la izquierda, es decir: monoprogramados, residentes, inmóviles, contiguos y enteros.

A medida que surgían nuevas ideas y nuevas técnicas de gestión de memoria se lograban modelos más eficientes:

- Con los modelos de PARTICIONES FÍJAS y PARTICIONES VARIABLES se añadió la característica de multiprogramación, aunque permanecían las restantes.

- Gracias a la técnica de SWAPPING (Intercambios), se incorporaba a los nuevos modelos la característica de que los procesos no tenían porqué permanecer

residentes en memoria a lo largo de su ejecución, podían ir a disco y volver a cargarse posteriormente.

- Con una nueva idea, la REUBICACIÓN DINÁMICA, consistente en retrasar la traducción de direcciones hasta el momento de la ejecución, se evitaba la inmovilidad de los mismos, consiguiendo que a lo largo de la ejecución de los procesos estos pudieran cambiar de posición en la memoria física.

- Los modelos de PAGINACIÓN y SEGMENTACIÓN conseguían que el código de los procesos no tuvieran que situarse en zonas contiguas de la memoria, con lo que el proceso podía dividirse en partes (Páginas o Segmentos) y cada una de estas partes se podían colocar en cualquier zona de la memoria.

- Los modelos más modernos han incorporado una nueva característica, el proceso ya no tiene que estar entero en la memoria para poder ejecutarse, basta con que esté una parte del mismo, la que en ese momento se necesite para su ejecución. Este último modelo, que incorpora, por tanto, las características de: multiprogramado, no residente, móvil, no contiguo y no entero es al que se le conoce con el nombre de **MEMORIA VIRTUAL**.

4. Modelos básicos de traducción de direcciones

Inicialmente los modelos de traducción sólo estaban pensados para sistemas monoprogramados.

Posteriormente se fueron introduciendo técnicas que permitieran la traducción en sistemas multiprogramados.

Los modelos que veremos son:

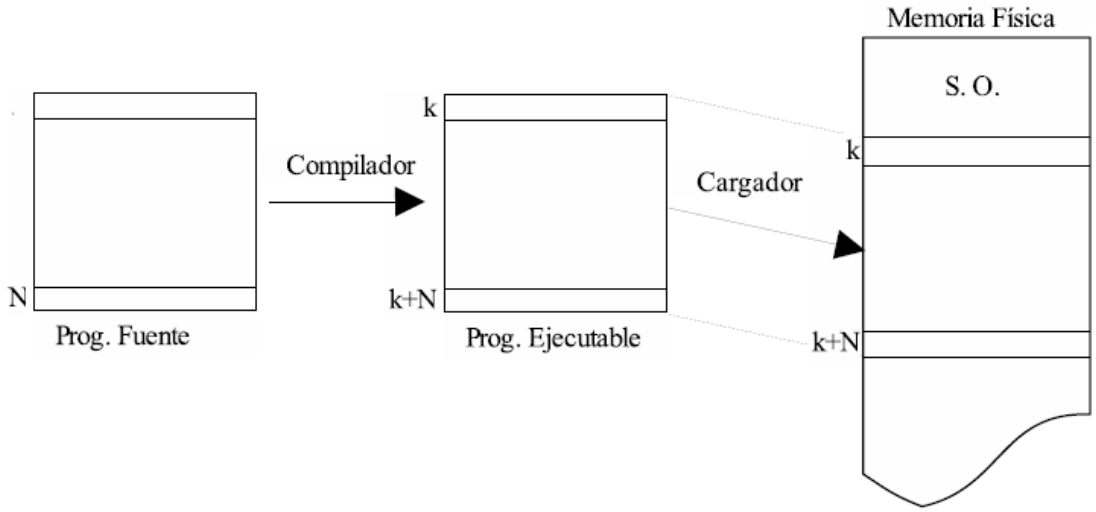
- Modelo 0.
- Modelo 1 (Reubicación estática).
- Modelo 2 (Particiones).
- Modelo 3 (Swapping).
- Modelo 4 (Reubicación dinámica).

4.1. Modelo 0

Características: monoprogramado, residente, inmóvil, contiguo y entero.

La **traducción de direcciones** lógicas a físicas se realiza **durante la generación del ejecutable** ⇒el ejecutable contiene las direcciones físicas.

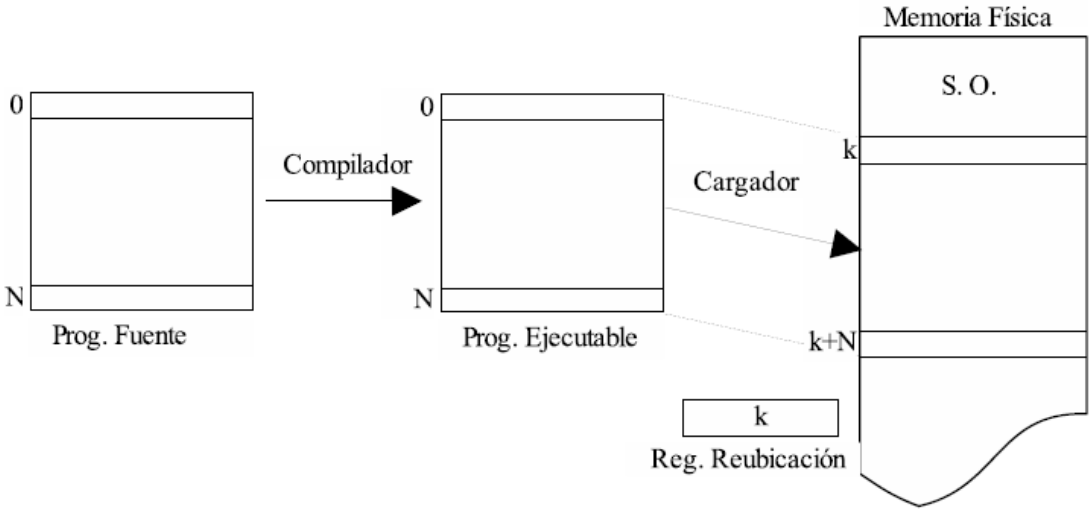
- Problemas:
  - Si crece el SO hay que volver a compilar todos los programas que existen en la máquina.
- Soluciones:
  - Cargar el SO en la parte más alta de la memoria y los procesos en la parte baja.
  - Usar el modelo siguiente.



4.2. Modelo 1 (Reubicación estática)

Las mismas características que el Modelo 0 pero ahora la **traducción de direcciones** lógicas a físicas se realiza **durante la carga del proceso en memoria**.

Disponemos de un registro (Registro de reubicación) que nos dice la posición de memoria donde tenemos que cargar el proceso:



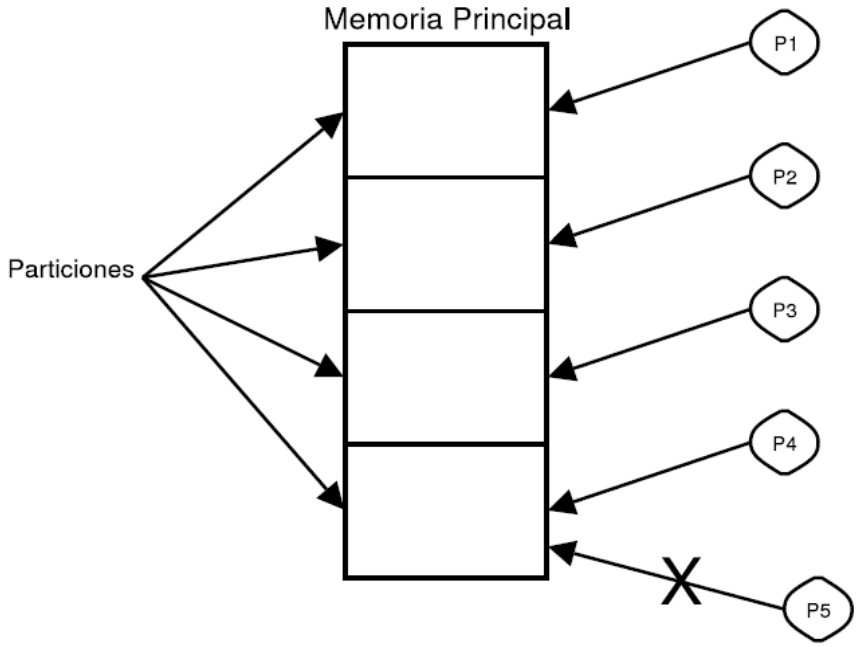
4.3. Modelo 2 (Particiones)

Mismas características Modelo 1 pero ahora hay multiprogramación.

La **traducción de direcciones** lógicas a físicas se realiza **durante la carga** del proceso en memoria.

Al tener más de un proceso en memoria:

- Debe cuidar la protección entre procesos.
- Se debe organizar el espacio lógico de cada proceso, **divide la memoria disponible para los procesos en particiones en cada una de las cuales sólo puede residir un proceso** (influye en el grado de multiprogramación):



Hay dos tipos de enfoques dependiendo de si el número de las particiones es fijo o variable:

- Particiones Fijas (Multiprogramming with Fixed Tasks).
- Particiones Variables (Multiprogramming with Variable Tasks).

4.3.1. Particiones Fijas (MFT)

La memoria se divide en regiones fijas en tamaño y número.

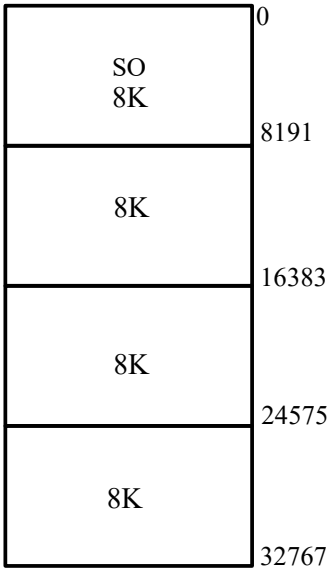
- Todas las particiones pueden ser de **igual tamaño:**

  El SO necesita almacenar la siguiente información:

Nº Part	@Base	Estado
1		
2		
:		
N		

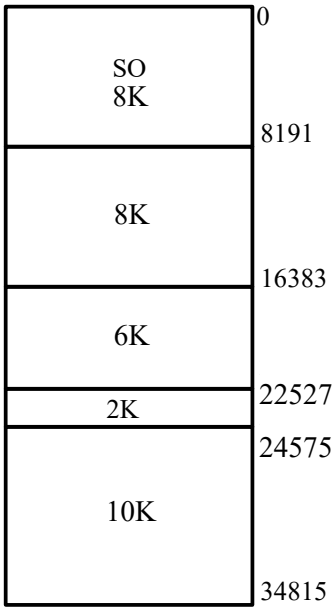
donde:

- **Nº Part** es el número de partición.
- **@Base** es la dirección física donde empieza la partición.
- **Estado** de la partición (libre u ocupado).



	@Base	Estado
0	0	1
1	8192	1
2	16384	1
3	24576	1

- O de **diferentes tamaños**:



	@Base	Estado	Tamaño
0	0	1	8192
1	8192	1	8192
2	16384	1	6144
3	22528	1	2048
4	24576	1	10240

El SO requiere almacenar la siguiente información:

Nº Part	@Base	Estado	**Tamaño**
1			
2			
:			
N			

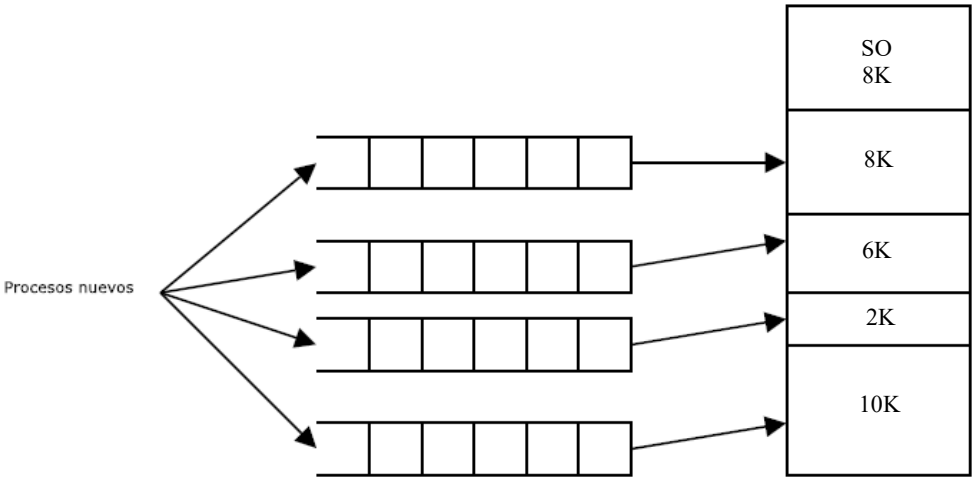
donde:

- **Tamaño** es el tamaño de cada partición.

La MMU usa dos registros (por velocidad): el registro límite (que contiene el tamaño de la partición), el registro base (que contiene la dirección base de la partición).

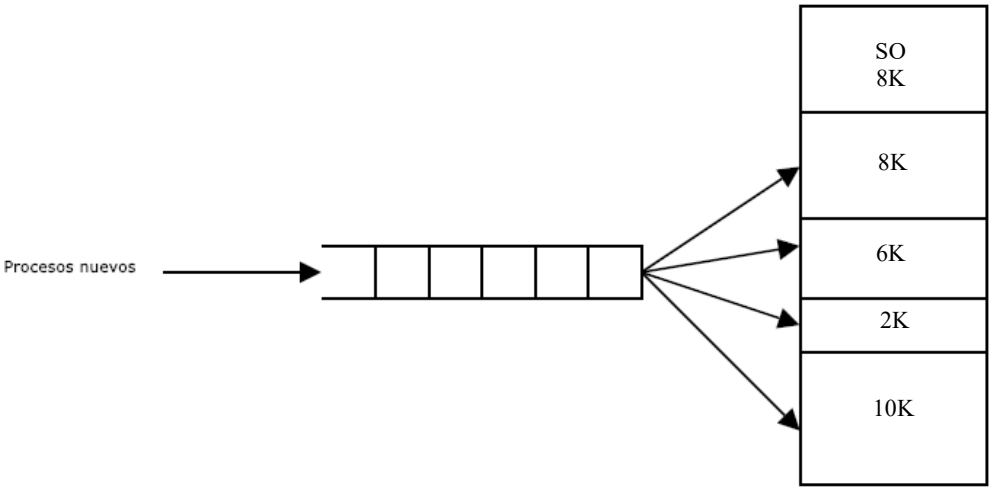
Algoritmo de asignación de particiones

- Para particiones de **igual tamaño**:
  - Buscamos cualquier partición que esté libre y la asignamos al proceso.
  - Si todas están ocupadas hay que buscarle sitio sacando uno de los procesos que actualmente están en memoria.
- Para particiones de **diferente tamaño** tenemos dos maneras:
  - Una cola por partición. Asignarle la **partición más pequeña en la que quepa** (suponiendo que conocemos su tamaño máximo):



Puede suceder que tengamos particiones muy grandes sin usar.

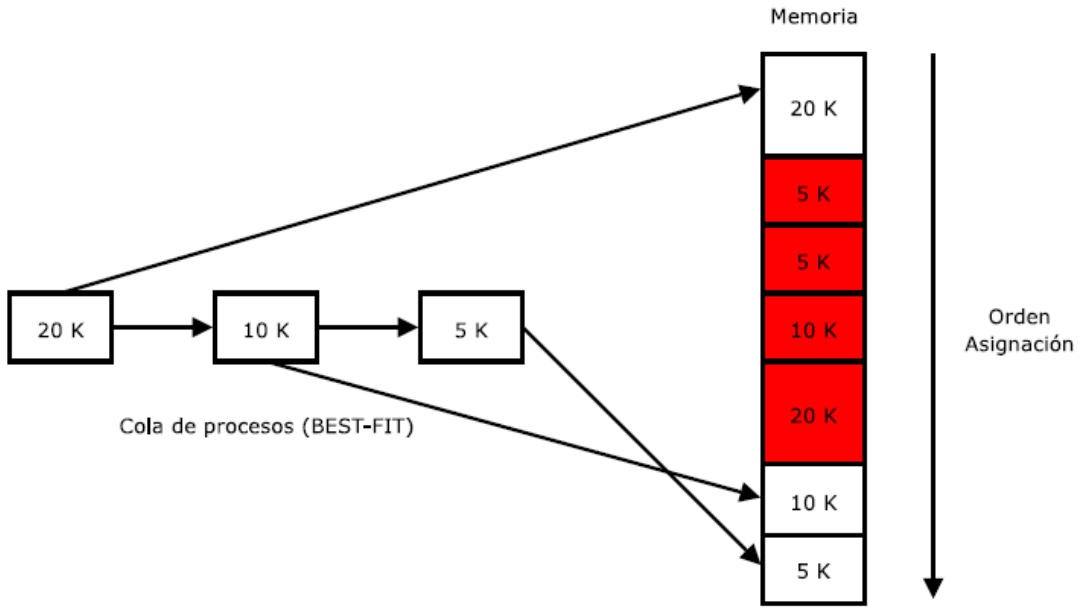
  - Una sola cola para todas las particiones. Asignarle la **partición más pequeña disponible** en la que quepa:



¿Qué proceso pasa a MP?

- **First Fit:** el primer proceso que quepa en la partición que quede libre.
- **Best Fit:** el proceso que mejor se adapte al tamaño de la partición que quede libre.

En particiones fijas las particiones se asocian a procesos y no al revés, para cada partición buscamos entre los procesos en cola:

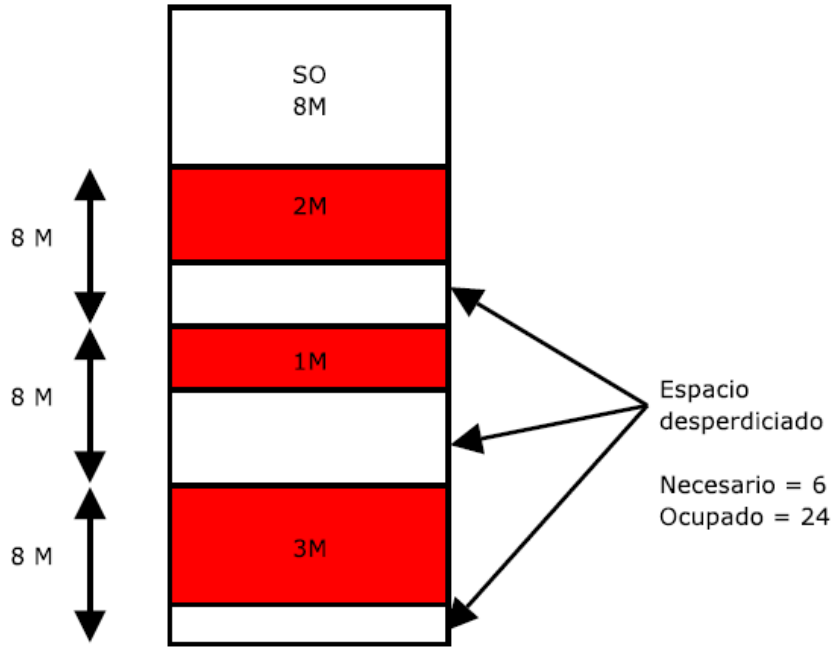


Ventajas

- El uso de particiones de distinto tamaño proporciona cierto grado de flexibilidad a las particiones estáticas.
- Ambas son fáciles de implementar y exigen un software del SO y una sobrecarga mínima.

Inconvenientes

- El número de particiones y su tamaño se especifica en el momento de la generación del sistema. Limita el número de procesos activos (grado de multiprogramación).
- Los trabajos pequeños no hacen uso eficiente del espacio de las particiones (no sabemos de antemano el tamaño de los procesos). Cualquier programa por pequeño que sea, siempre ocupa una partición completa.



- Al problema de malgastar espacio interno en cada partición se le denomina **fragmentación interna.**
- Se reduce la fragmentación interna usando particiones de diferentes tamaños.
- Se produce **fragmentación externa**, hay memoria suficiente para atender la solicitud de un proceso pero no se le puede conceder porque no es contiguo o porque el administrador de memoria utilizado no lo permite.

¿Cómo calculamos esta fragmentación?

- Suponemos que los distintos huecos libres en memoria forman una partición (**no** contamos los huecos dentro de una partición, es decir, la fragmentación interna no se contabiliza nunca como externa).
- Empezamos a asignar los **procesos pendientes** a dicha partición hasta que ya no nos quede más espacio.
- La fragmentación externa será igual a la suma de los tamaños de los procesos contenidos en esta partición.

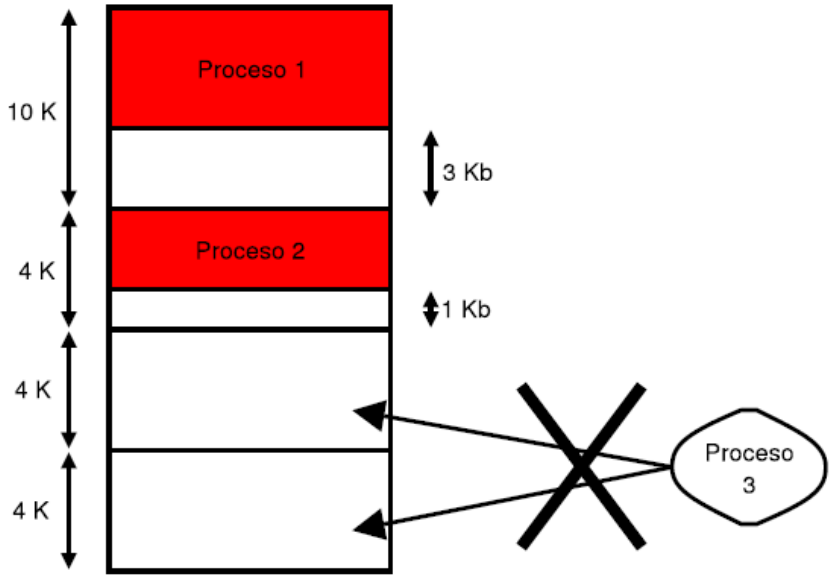
Ejemplo: Disponemos de un sistema con MP de 22 Kb que está dividida según la siguiente tabla de particiones

Partición	Tamaño	Partición	Tamaño
1	10 Kb	3	4 Kb
2	4 Kb	4	4 Kb

Si llegan los siguientes procesos:

Proceso	Tamaño	Proceso	Tamaño
1	7 Kb	3	6 Kb
2	3 Kb		

¿Cuánto vale la fragmentación interna? ¿Y la externa?



Fragmentación interna = 3 + 1 = 4 Kb; Fragmentación externa = 6 Kb

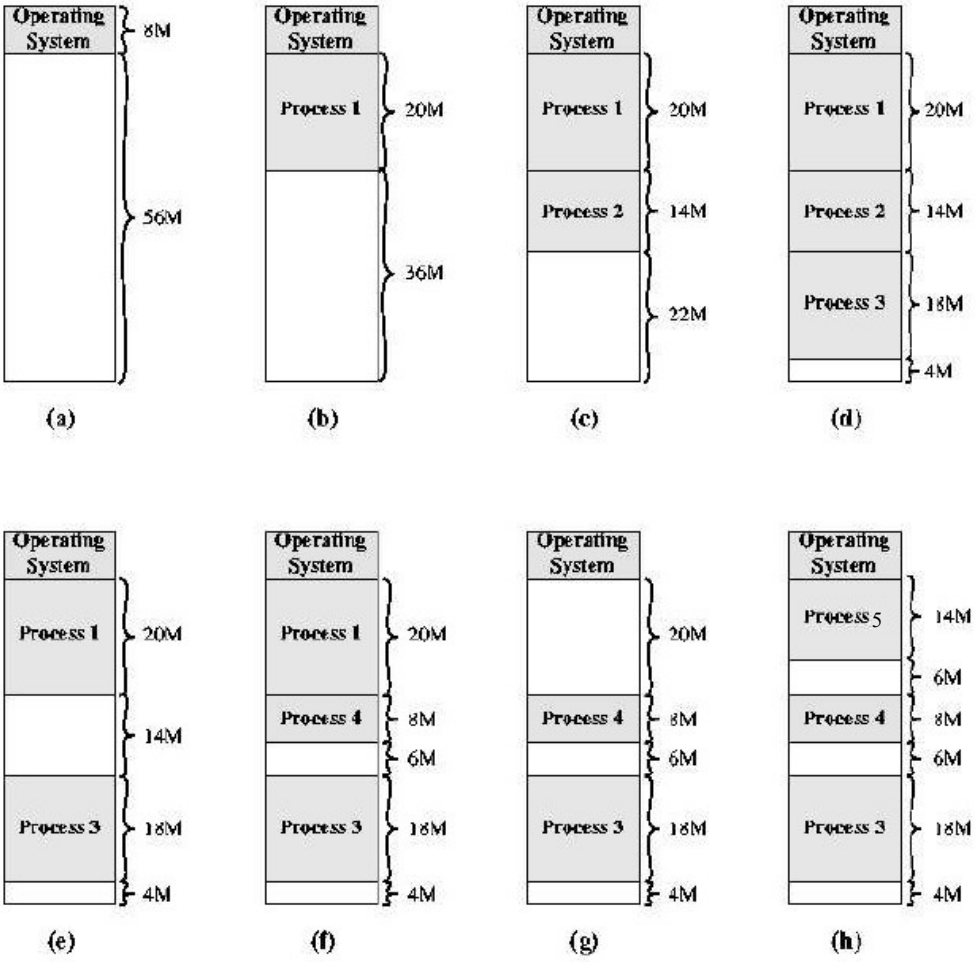
4.3.2. Particiones Variables (MVT)

Cada vez que se crea un proceso, el SO busca un hueco en memoria de tamaño suficientemente grande para alojar al proceso.

Trata de eliminar la fragmentación interna permitiendo que los tamaños de las particiones varían dinámicamente.

El SO mantiene información en la que se indica qué partes de la memoria están disponibles y cuáles están ocupadas.

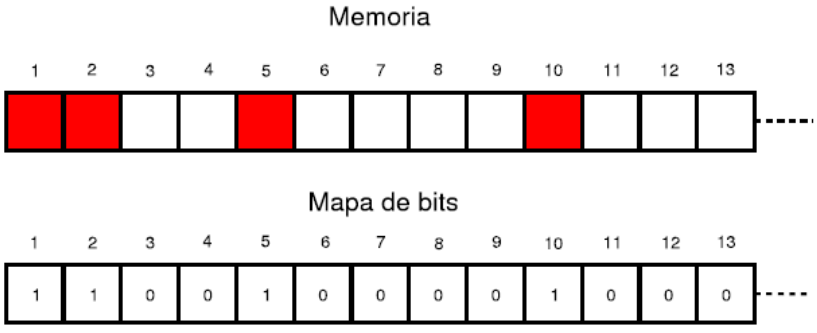
La asignación de espacio consistirá en buscar aquel hueco lo suficientemente grande para albergar al proceso.



Secuencia de sucesos: (a) Situación inicial (b) Llega proceso de 20 M (c) Llega proceso de 14 M (d) Llega proceso de 18 M (e) Termina proceso de 14 M (f) Llega proceso de 8 M (g) Termina proceso de 20 M (h) Llega proceso de 14 M.

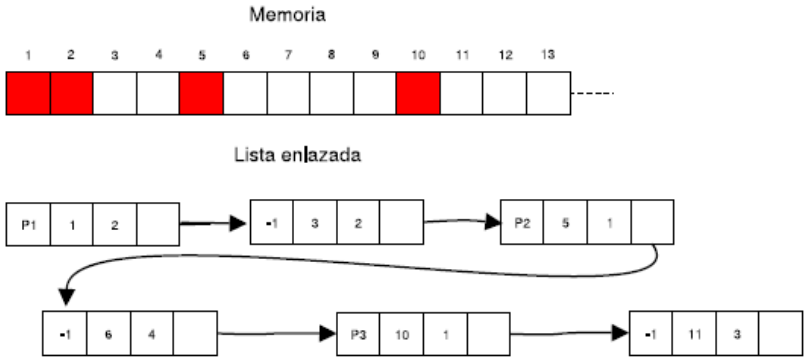
Para llevar la cuenta de los huecos libres podemos usar:

- **Mapas de bits**: La memoria se divide en *clicks* del mismo tamaño. Tendremos una ristra de *bits* donde pondremos un 1 si el click de memoria está ocupado y 0 si está libre:



Mientras más pequeño es el *click* menos fragmentación interna tendremos pero la cantidad de memoria que necesitaremos para almacenarlo será mayor. El mapa de *bits* no indica qué proceso está ocupando los *clicks*.

- **Listas encadenadas o enlazadas**: lista en donde cada celda tiene la siguiente información:
  - Id. proceso: identificador del proceso.
  - @inicio: dirección donde comienza la partición.
  - Tamaño: tamaño de la partición.
  - Siguiente: puntero al siguiente elemento de la lista.



Los elementos de la lista de huecos suelen estar ordenados por direcciones. Esto tiene la ventaja de que la actualización de la lista cuando un proceso termina o se manda a disco es relativamente sencilla.

Planificación de procesos

Partimos de una lista enlazada de huecos libres y ocupados.

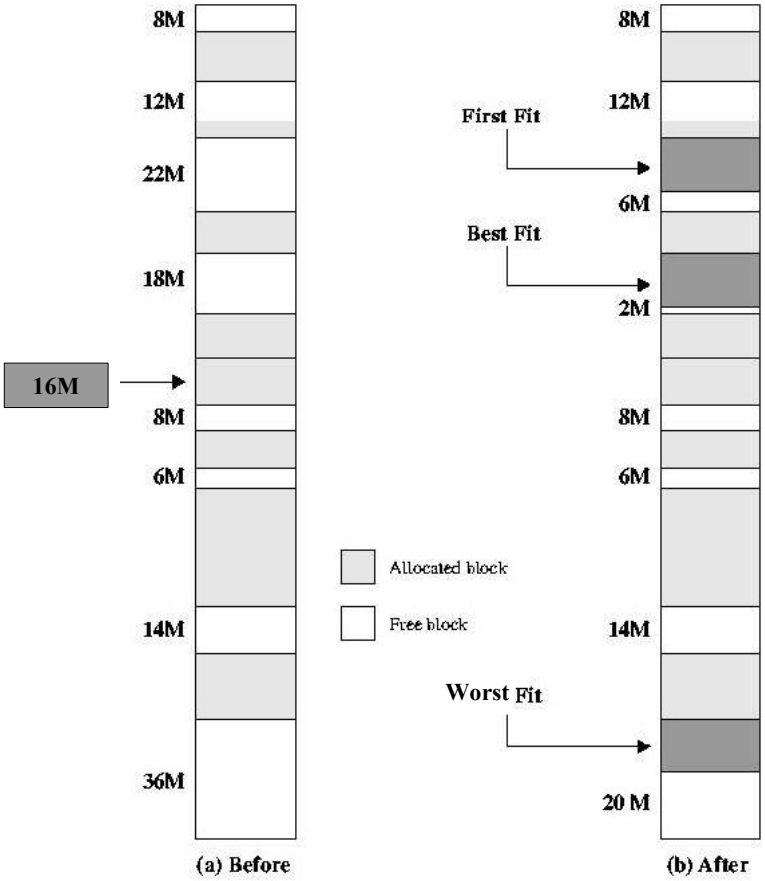
Disponemos de una cola en donde tenemos los procesos que han entrado al sistema por orden de llegada.

Planificamos esta cola de manera ordenada en base a varios algoritmos:

- **First Fit**: buscamos el primer hueco donde quepa el proceso.
- **Next Fit**: buscamos el próximo hueco donde quepa el proceso, pero sin empezar a buscar por el principio como en first fit sino desde donde nos quedamos en la última asignación.
- **Best Fit**: buscamos el hueco donde mejor quepa.
- **Worst Fit**: buscamos el hueco donde peor quepa.

En particiones variables los procesos se asocian a particiones y no al revés: para cada proceso en cola buscamos entre las particiones disponibles

Ejemplo: asignación de un bloque de 16 M

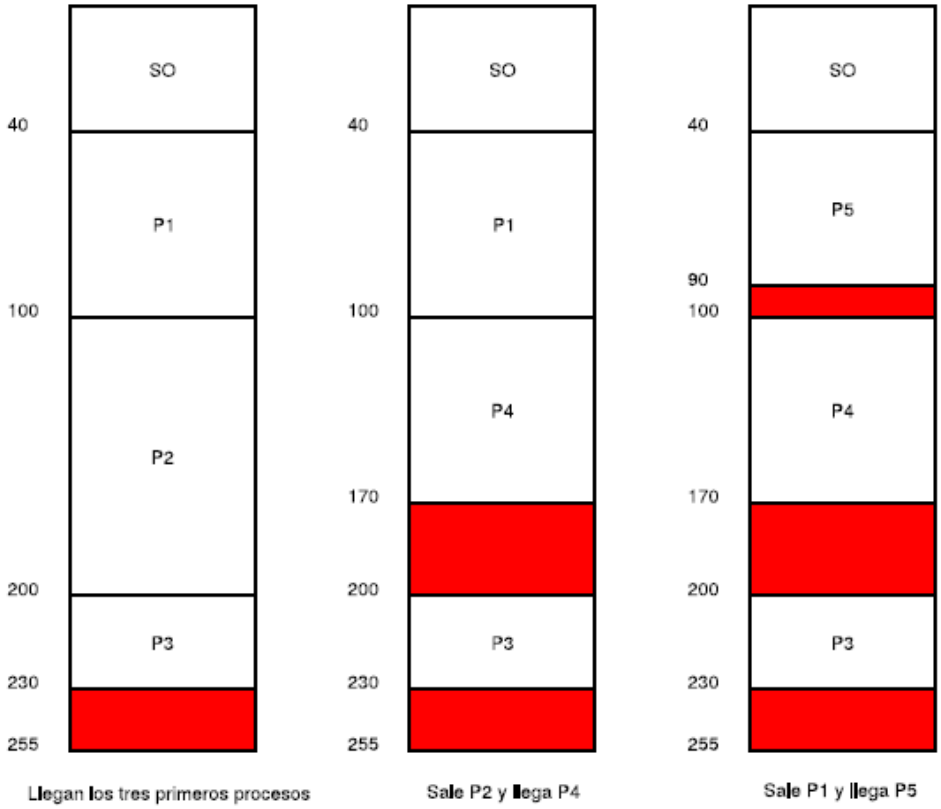


Ejemplo: tenemos una memoria de 256 Kb. 40 Kb de ellos están ocupados por el SO y llegan los siguientes procesos:

Proceso	Tamaño	Tiempo en memoria
P1	60K	10 seg.
P2	100K	5 seg.
P3	30K	20 seg.
P4	70K	8 seg.
P5	50K	15 seg.

Representar las siguientes situaciones suponiendo que usamos first fit:

- Llegan los tres primeros procesos.
- Sale el proceso P2 y entra el proceso P4.
- Sale el proceso P1 y entra el P5.



Ventajas

Soluciona el problema de la fragmentación interna.

Inconvenientes

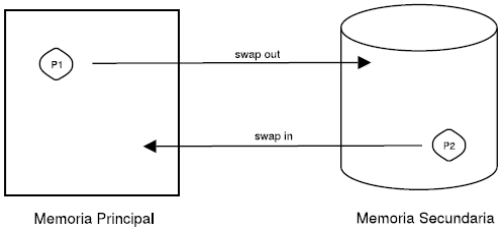
Fragmentación externa, se puede solucionar mediante la **compactación.**

- En la compactación se fusionan los huecos no contiguos en un único hueco.
- Se hace una reubicación de los procesos.
- No hay algoritmo eficiente para la reubicación de los procesos en memoria.

4.4. Modelo 3 (Swapping)

Mismas características Modelo 2 salvo la **no residente** de los procesos.

Se basa en usar un disco como respaldo de la memoria principal, cuando no caben en MP todos los procesos activos, se elige un proceso residente y se almacena en MS (Memoria secundaria).



Es la base de la multiprogramación

Hay dos alternativas en la asignación de espacio en el dispositivo swap:

- Preasignación: al crear el proceso se reserva espacio de swap.
- Sin preasignación: sólo se reserva espacio de swap cuando se expulsa el proceso de MP.

Desventajas

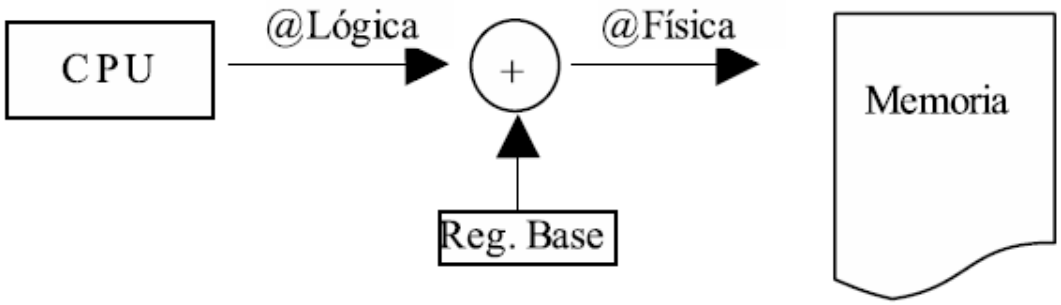
- Inmovilidad ⇒ colocar al proceso en la misma zona de memoria donde se encontraba.
- El dispositivo de almacenamiento secundario consume mucho tiempo.
- Tiempo de intercambio alto. Se mejora:
  - Aumentando la velocidad de transferencia entre Memoria Secundaria y Memoria Principal.
  - Intercambiando exactamente la cantidad de memoria necesaria.
  - Solapando la ejecución de un proceso con el intercambio del proceso de usuario previamente ejecutado y el que se ejecutará.
- Planificación: la cola de preparados consiste en aquellos procesos cuya imagen está en MS y están preparados para ejecutarse, hay que crear un nuevo estado.
- Los procesos que se encuentran en espera de realizar una operación de E/S no pueden salir de memoria.
- Puede producirse interbloqueo.

4.5. Modelo 4 (Reubicación dinámica)

Mismas características Modelo 3 salvo la **movilidad** de los procesos.

Se retrasa la traducción de direcciones lógicas en físicas hasta el momento de la ejecución.

El registro de reubicación puede cambiar su valor a lo largo de la ejecución de un mismo proceso, ahora hablamos de registro base:



5. Modelos avanzados de traducción de direcciones

Se caracterizan por: multiprogramación, no residente, móvil, **no contiguo** y entero.

La traducción se realiza en tiempo de ejecución.

Existen 3 modelos que cumplen estas características: Paginación, Segmentación y Modelos que combinan la paginación y la segmentación.

5.1. Paginación

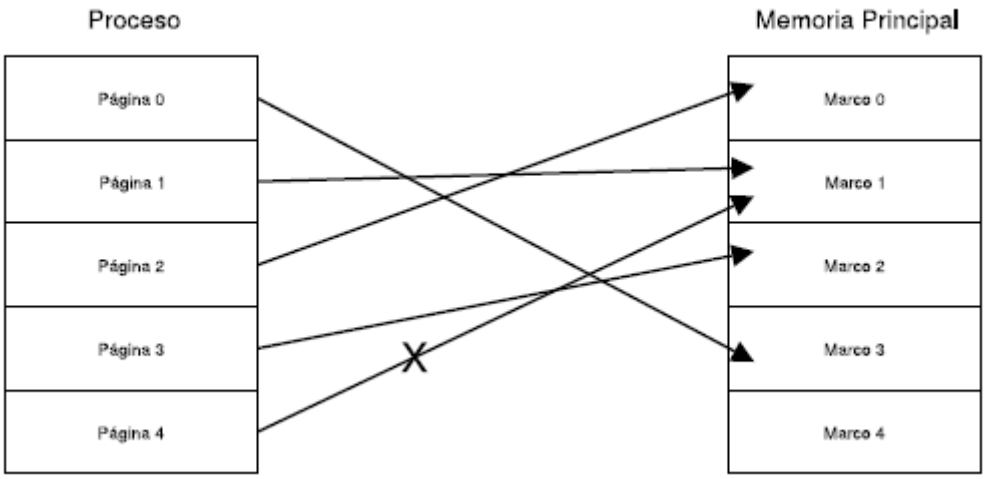
La fragmentación externa se debe a la imposibilidad de asignar espacios no contiguos de memoria.

La paginación intenta solucionar este problema sin usar compactación.

La memoria se divide en trozos denominados **marcos o tramas** de un tamaño fijo.

El proceso se divide en bloques denominados **páginas** del mismo tamaño que los marcos.

El SO realiza esta división y asigna páginas a marcos:



Para establecer la relación entre páginas y marcos se usa la tabla de páginas.

Estructuras

El SO necesita nuevas estructuras para implementar la paginación.

Una es la **tabla de marcos** de página que indicará los marcos que están libres y ocupados.

Otra es la **tabla de páginas** que servirá para indicar en qué marco está situada una página. Es una tabla indexada por paginas cuya estructura (aunque depende de la computadora) es la siguiente:

- **BV** es un bit de validez (si vale 1 indica que la entrada es válida)(Obligatorio)

	t	BV	R	W	X
0					
1					
N					

- **R,W,X** son bits de protección (Opcional)
- **t** indica el marco que contiene la página (Obligatorio)

Hay una tabla de páginas por proceso. Esta tabla tiene un tamaño fijo.

Planificación de procesos

Estructuras de datos: cola de trabajos (necesidades de memoria), lista de marcos no asignados, tablas de páginas (en cada proceso).

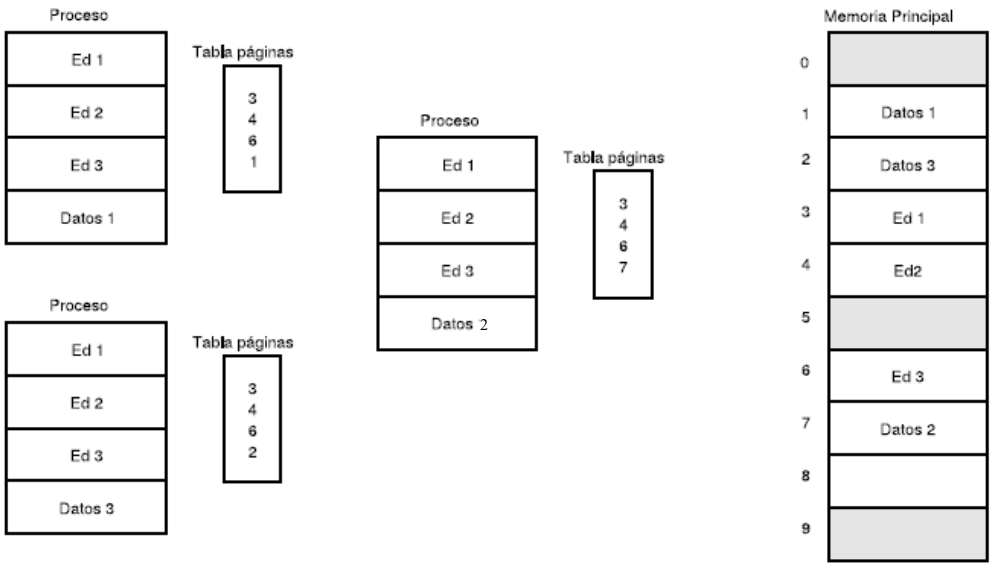
Algoritmo

- Si un trabajo requiere **n** páginas, buscamos **n** marcos disponibles y se los asignamos.
- Cargamos cada página en el marco que le corresponde.
- Actualizamos los valores de la tabla de páginas.

Ventajas

No se produce fragmentación externa.

Compartición de páginas.



Protección de páginas y control de direcciones ilegales.

Inconvenientes

Fragmentación interna, se puede reducir disminuyendo el tamaño de página –> tabla de páginas mayor.

¿**Cómo se almacena esta tabla**?

- Tabla de paginas con un único nivel.
- Tablas de páginas multinivel.
- Tablas de páginas invertidas.
- Buffer de traducción adelantada.

Objetivos de estas posibilidades:

- **Reducir** en lo posible los **recursos en memoria** para almacenarla.
- La **asociación** marco/página debe ser **rápida**.

5.1.1. Tabla de páginas con un único nivel

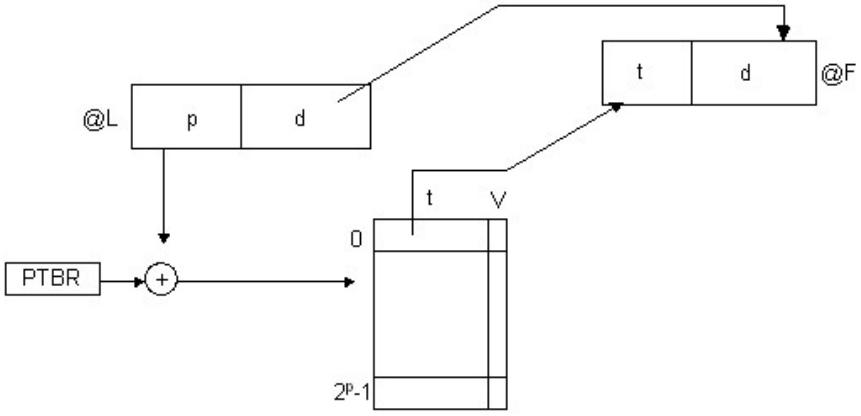
Cuando un proceso se ejecuta se carga completamente su tabla de páginas en memoria en posiciones consecutivas.

¿Dónde se almacena esta tabla?

- Usando **registros** de propósito general, la tabla de paginas almacenada en algún dispositivo se carga en registros. Gran rapidez de acceso pero las tablas no suelen caber.
- Almacenamos la tabla en **memoria principal**, en el registro PTBR (Page Table Base Register) guardaremos la dirección de la tabla en memoria. Este valor se actualiza cuando hay cambio de proceso.
- El esquema anterior presenta problemas de velocidad. Se opta por una solución basada en **registros asociativos** (memoria caché) denominado TLB (ver punto siguiente).

Traducción de direcciones, hardware de paginación

El esquema de ejecución de una instrucción es:



La CPU genera direcciones lógicas **L** que tienen dos componentes:

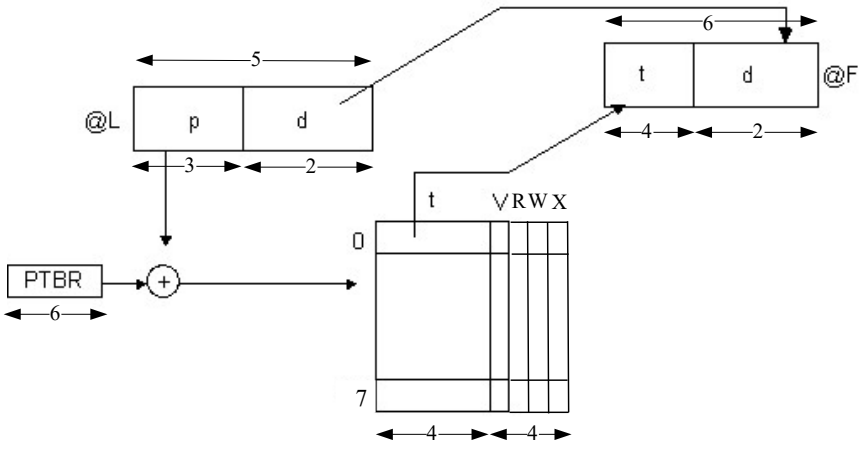
- Un número de página (**p**).
- Un desplazamiento en la página (**d**).

En general, **d** se obtiene del tamaño de la página y **p** de la diferencia entre **L** y **d**.

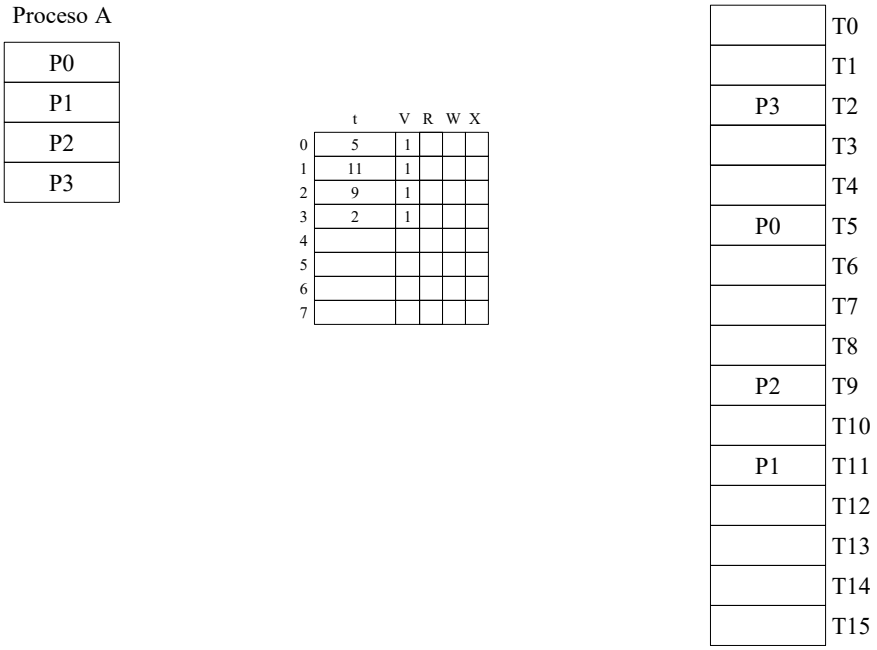
Una vez que tenemos calculado **p**, usamos la entrada **p** de la tabla de páginas para calcular el **número del marco** que la contiene.

Este valor (**t**) lo concatenamos (||) con **d**, y el resultado final es la dirección física.

Ejemplo 1: Tenemos una mini memoria de 64 bytes. Las páginas son de 4 bytes y la @Lógica tiene 5 bits. Las tablas de páginas tienen bits de protección de R,W,X. Dibuja el esquema de traducción de direcciones e indica el tamaño de cada campo.



Si un proceso A de 16 bytes se almacena en las tramas 5, 11, 9 y 2 respectivamente, ¿cómo queda la tabla de páginas?



¿Si el proceso A lanza la @Lógica 9 que @Física se genera?

9 = 1001 = 010 || 01 = @Lógica (5 bits)

p= 010 d= 01 (Segunda instrucción de la página 2)

Accedemos a la 2ª posición de la Tabla de páginas, donde hay almacenado que t=9.

La @Física es de 6 bits, 4 bits para la t y 2 bits para la d. Accedemos por lo tanto a la segunda instrucción almacenada en la trama 9, cuya dirección física exacta es:

@Física = 1001 || 01 = 37

Ejemplo 2: Si tenemos una máquina con páginas de 256 bytes y la dirección lógica ocupa 10 bits, y la tabla de páginas siguiente:

	t
0	0
1	3
2	2
3	1

La siguiente dirección lógica 1101111101, ¿a qué marco y dirección física accede?

$256 = 2^8 \Rightarrow d$=8 *bits* menos significativos = 01111101 = 125

p=10–8=2 *bits* más significativos=11 ⇒ accedo a la página 3 que está en la trama 1.

La dirección física se calcula concatenando a la dirección del marco el desplazamiento:

@física = t || d = 1 || 125 = 01 || 01111101 = 101111101 = 381

5.1.2. Buffer de traducción adelantada

Si la tabla de páginas está en memoria, acceder a una instrucción supone ahora más de un acceso a memoria, lo que retrasa el acceso a la misma.

Para reducir el tiempo de búsqueda del marco de página se hace uso de una cache especial, incluida en la MMU, para las entradas de la tabla de páginas llamada **buffer de traducción adelantada** (TLB).

Esta caché almacena las **entradas completas de la tabla de páginas más usadas** junto con el identificador de la página**:**

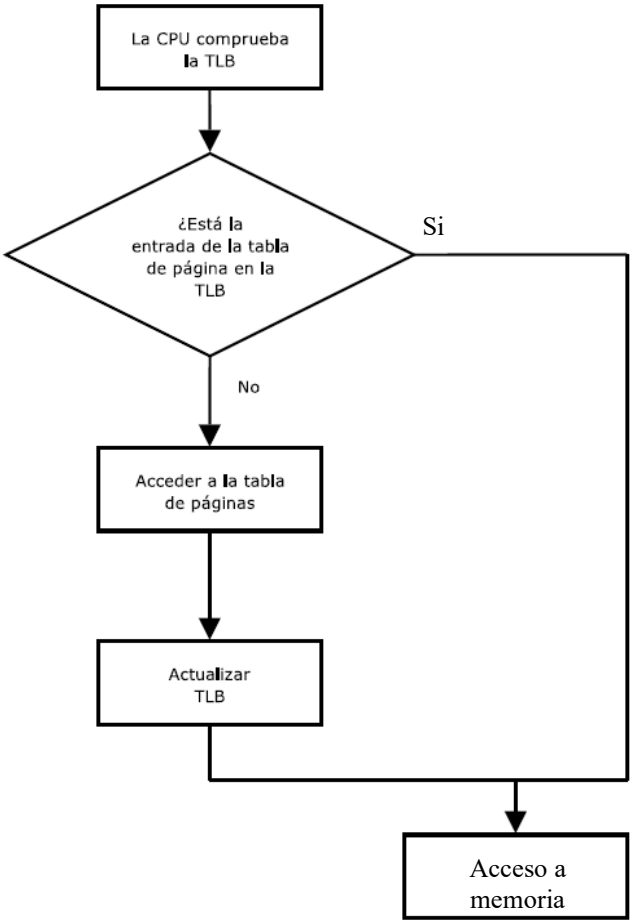
Número página	Entrada tabla página
0	
...	...
n	

El identificador de páginas sirve para hacer búsquedas asociativas.

Dos alternativas para el diseño TLB:

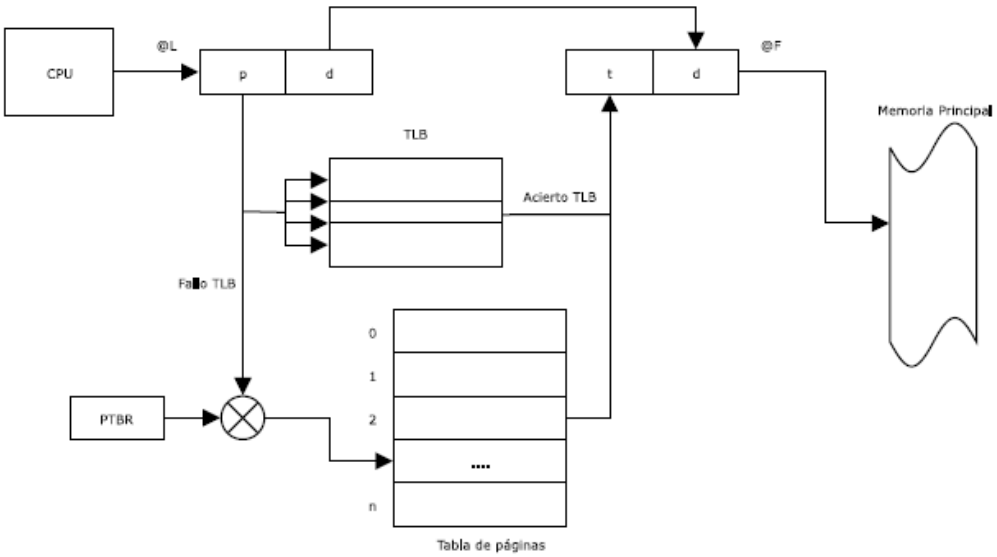
- TLB sin identificador de proceso: cuando hay un cambio de proceso se invalida la TLB completa
- TLB con identificador de proceso: se añade un campo más para identificar al proceso, no es necesario invalidar la TLB cuando hay cambio de contexto.

Una TLB sin identificador de proceso se actualiza siguiendo el siguiente diagrama de flujo:



El uso de TLB se pueden aplicar tanto a tablas de página un único nivel, tablas de páginas multinivel y a tablas de paginas inversas (lo veremos más adelante).

Si usamos tablas de un único nivel la MMU quedaría de la siguiente manera:



5.1.3. Tablas de páginas multinivel

Se divide la tabla de paginas en secciones.

Cada proceso tiene asociado una **jerarquía de tablas de páginas**.

Si esta jerarquía es de 2 niveles:

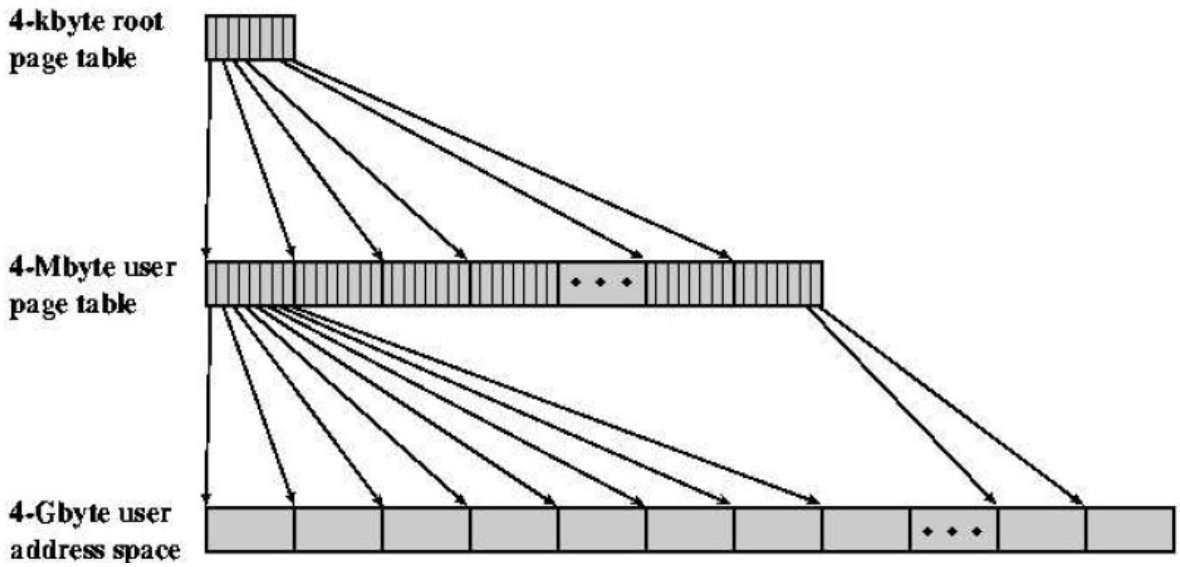
- El **primer nivel** es un directorio **raíz de páginas**, cada entrada apunta a tablas de páginas (secciones):

	tabla_paginas	BV
0		
1		
:		
N		

  - **tabla_paginas** es la **dirección en memoria** de la tabla de páginas.
  - **BV** es un bit de validez (si vale 1 indica que la entrada es válida).

- Un **segundo nivel** con las tablas de páginas.

El esquema general es:

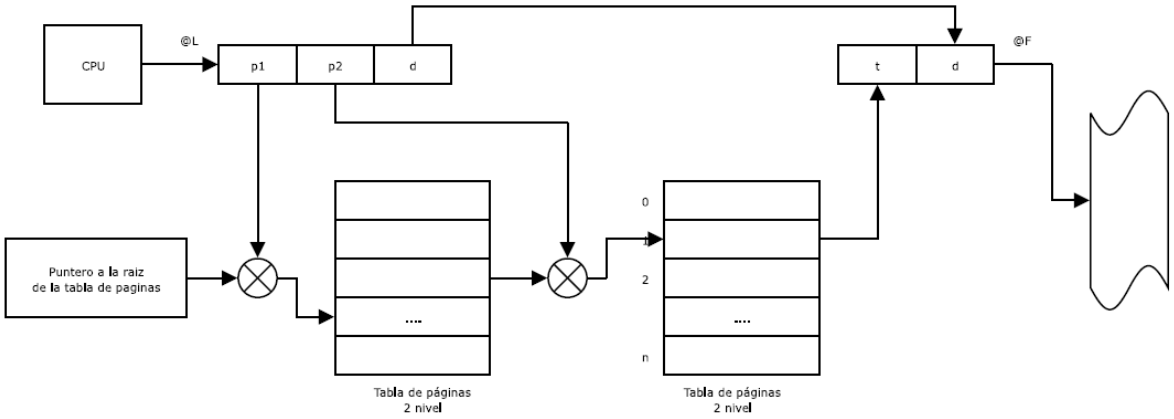


Pueden existir más de dos niveles.

El tamaño del directorio raíz de páginas y las tablas de páginas del segundo nivel suele estar limitado a una página.

Con este esquema se permiten compartir tablas de páginas intermedias y sólo la tabla de páginas de primer nivel debe residir en memoria.

El esquema de ejecución de una instrucción es:



La CPU genera direcciones lógicas **L** que tienen tres componentes:

- La tabla de páginas a la que queremos acceder (**p1**).
- El número de página (**p2**).
- Un desplazamiento en la página (**d**).

Ejemplo 1: Si tenemos direcciones lógicas de 32 bits, un tamaño de pagina de 4 Kb, cada entrada de la tabla de paginas ocupa 4 bytes y se dedican 10 bits de la dirección a cada nivel ¿Qué espacio de direcciones cubre cada tabla de segundo nivel? ¿Qué espacio total se puede direccionar?

Las tablas de páginas de 2º nivel tienen 2^{10} entradas.

Como cada marco ocupa 4Kb, cada tabla de 2º nivel direcciona $\Rightarrow 2^{10}$ x 4 Kb $\Rightarrow$ 4 Mb.

El espacio total que se puede direccionar con este esquema es de 2^{10} x 4 *Mb*= 4Gb.

Ejemplo 2: Teniendo en cuenta los datos del ejemplo anterior, si un proceso utiliza sólo los 12 Mb de la parte superior de su mapa de memoria y 4 Mb de la parte inferior ¿cuánto espacio necesitaríamos para almacenar toda su tabla de páginas? ¿y si usáramos un esquema con un único nivel?

Gastamos el espacio de la tabla de 1º nivel (2^{10} entradas x 4 bytes = 4 Kb), como tenemos que direccionar 16 Mb necesitamos 4 tablas secundarias (cada una ocupa 4 Kb y cada una direcciona 4 Mb). En total necesitamos almacenar 4 Kb de la tabla de 1º nivel + 16 Kb de tablas secundarias = 20Kb.

Si usáramos un único nivel es necesario tener almacenada completamente la tabla de páginas $\Rightarrow 2^{20}$ x 4 bytes = 4 Mb.

Ventajas

- Requieren menos espacio en memoria principal.
- Permite compartir páginas intermedias.
- Sólo es preciso que esté residente en memoria la tabla del primer nivel.

5.1.4. Tablas de páginas invertida

Para reducir el tamaño se usa una **tabla de páginas invertida**.

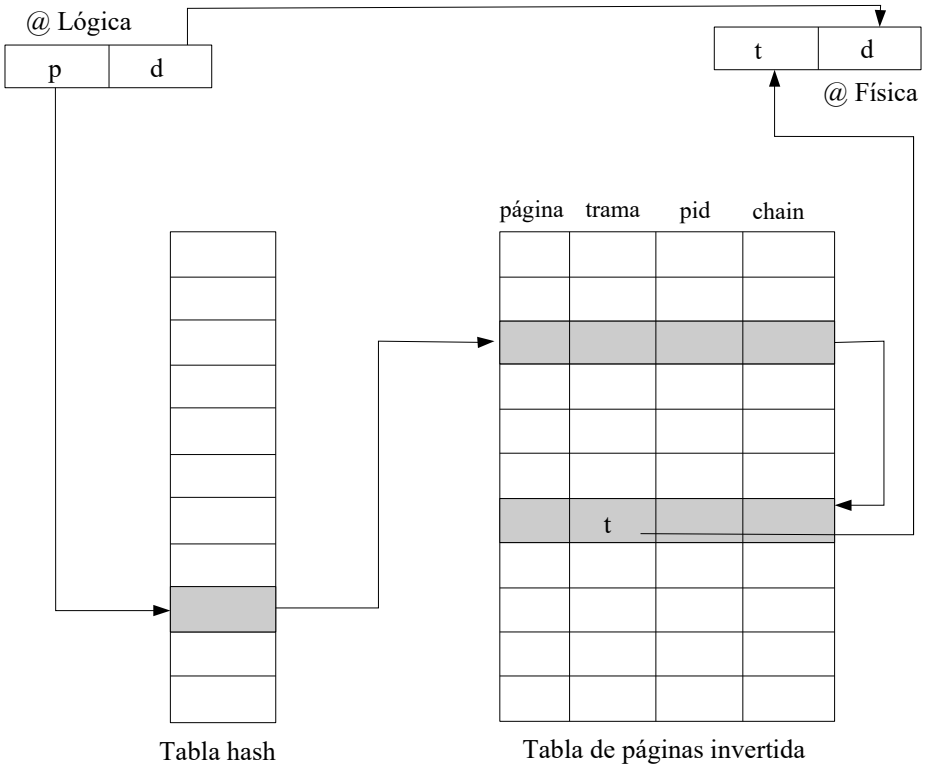
Esta tabla contiene **una entrada por cada marco de página** y está indexada por marco:

	Página	PID
0	7	
1	4	
..	..	
N	3	

- **PID** es el identificador de proceso que ocupa el marco.
- **Página** es la página que ocupa el marco.

Como la tabla está indexada por marcos no se puede hacer una búsqueda directa por páginas.

Para mejorar la velocidad de búsqueda la parte del número de página de una dirección virtual se traduce a una **tabla hash** por medio de una función hash simple:



Es necesario incluir el campo **encadenamiento** (chain) que es un apuntador a la siguiente entrada de la tabla de páginas invertida que comparte el mismo código hash.

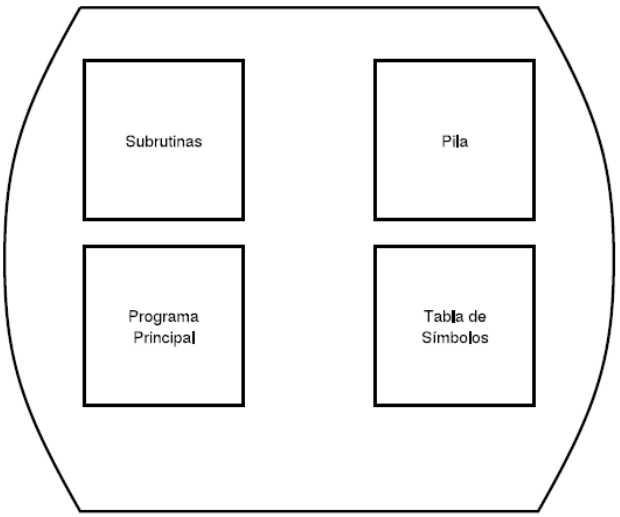
A partir de la página **p** se usa la función hash para acceder a la primera posición de la tabla de paginas invertidas.

Si el identificador de la página contenida en la tabla de paginas invertidas coincide con p, leemos la entrada de la tabla de páginas para obtener el número del marco.

Si el identificador de la pagina no coincide, buscamos en la siguiente entrada de la tabla de páginas invertida indicada en el campo **encadenamiento**.

5.2. Segmentación

La segmentación apareció en las arquitecturas Intel para permitir a los programadores dividir sus programas en “entidades lógicamente relacionadas entre sí”:



Estas entidades se denominan segmentos y el SO les irá asignando el espacio en memoria que necesiten (similar al particionamiento variable) –> no hay fragmentación interna.

Tabla de segmentos

El SO mantiene **una tabla de segmentos por proceso** cuyo objetivo es almacenar la información necesaria para localizar los segmentos en memoria. Estos segmentos **se almacenan en posiciones consecutivas de memoria**.

Necesitamos conocer donde se encuentra cada segmento y el tamaño que ocupa en memoria. Esa será la información básica que se almacenará en esta tabla.

La estructura de la misma será:

	@Base	Límite
0		
1		
..		
N		

- **@Base** es la dirección física donde se encuentra el segmento.
- **Límite** es el número de posiciones del segmento (realmente se pone el número de posiciones del segmento -1 como veremos más adelante).

Ejemplo:

Memoria de 7 Kb (7168 bytes). 5 Segmentos de 1000, 468, 400, 1100 y 1000 bytes respectivamente.

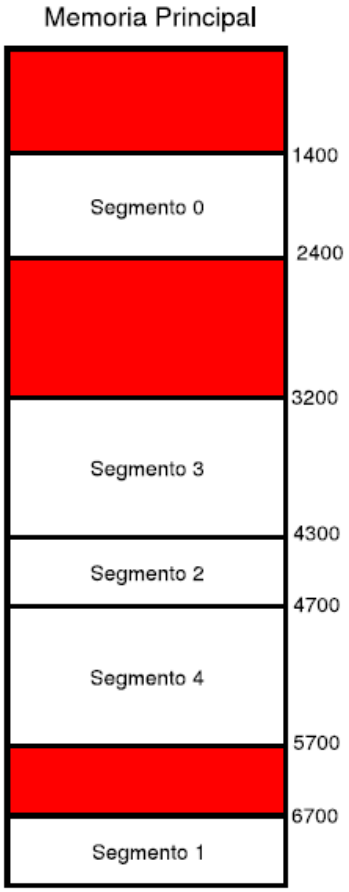


Tabla de segmentos

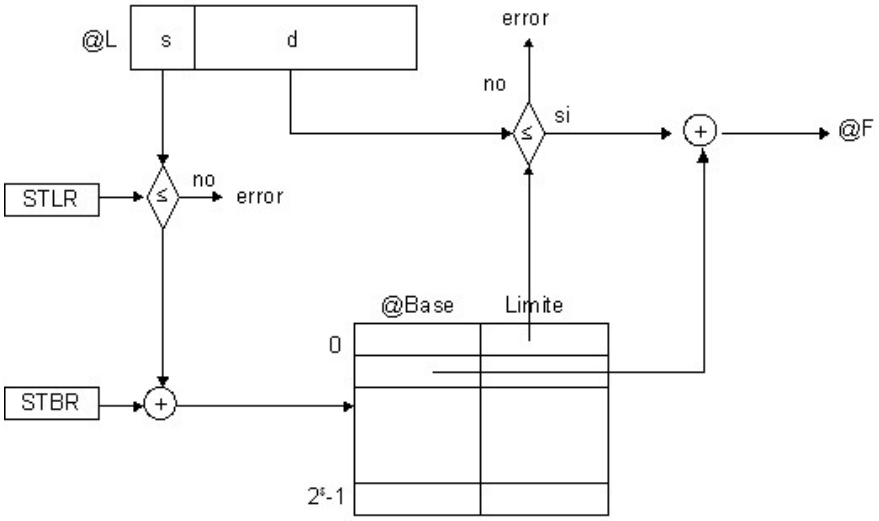
	base	límite
0	1400	999
1	6700	467
2	4300	399
3	3200	1099
4	4700	999

¿Dónde se carga esta tabla?

- En **registros** de propósito general (CS, SS, DS). Gran rapidez de acceso pero las tablas no suelen caber.
- En **memoria principal**, el registro STBR (Segment Table Base Register) guarda la dirección de la tabla en memoria y el STLR (Segment Table Limit Register) el número de registros máximo en la tabla (**entonces no es necesario el BV**).
- **Registros asociativos** (memoria caché), mejoran la velocidad.

Traducción de direcciones, hardware de segmentación

El esquema de ejecución de una instrucción es:



La CPU genera direcciones lógicas que tienen dos componentes:

- Un número de segmento **s** (bits más significativos)
- Un desplazamiento en el segmento **d** (bits menos significativos)

La **s** está relacionada con el número de segmentos máximo que puede tener un proceso y la **d** con el tamaño máximo del segmento.

Una vez que tenemos calculado **s**, usamos la entrada **s** de la tabla de segmentos para obtener la dirección física base en donde está el segmento y comprobamos que el desplazamiento está entre los límites (<=).

¿Por qué el límite se debe de poner el número de posiciones del segmento -1? Si el tamaño del segmento es de 1024 posiciones –> necesitamos 10 bits para poder direccionarlos luego tanto d como Limite tienen un tamaño de 10 bits pero es que en 10 bits el número máximo que podemos representar es 1111111111 = 1023 y no 1024

A esta dirección le sumamos el valor de **d**, el resultado final es la dirección física.

Ejemplo 1: Si el máximo número de segmentos que puede tener un programa es 4 y la dirección lógica ocupa 10 bits, ¿cuántos bits necesitamos para codificar el segmento y el desplazamiento?:

Necesitamos 2 bits para direccionar los 4 segmentos ⇒ s= 2 *bits* más significativos

$d=L-s=8$ *bits* menos significativos ⇒ el número de posiciones máximo de un segmento puede ser $2^8=256$ posiciones.

Ejemplo 2: Usando la arquitectura anterior, y la tabla de segmentos siguiente:

	@base	Límite
0	0	99
1	300	255
2	1000	199
3	1500	99

La siguiente dirección lógica 1101111101, ¿a qué segmento y dirección física accede? ¿y si la dirección fuera 1100110010?

- Para 1101111101

s=2 *bits* más significativos = 11 accedo al segmento 3 de límite 99.

$d=L-s=10-2=8$ *bits* menos significativos = 01111101 = 125

El desplazamiento supera al tamaño del segmento (125>99) ⇒ infracción de memoria.

- Para 1100110010

s=2 *bits* más significativos = 11 accedo al segmento 3 de límite 99.

$d=L-s=10-2=8$ *bits* menos significativos = 00110010 = 50.

La dirección física se calcula sumando a la dirección del segmento (1500) el desplazamiento (50):

@física = @Base + d = 1500 + 50 = 1550 = 11000001110 (11 bits).

Planificación de procesos

Se realiza de forma similar a Particiones Variables (MVT).

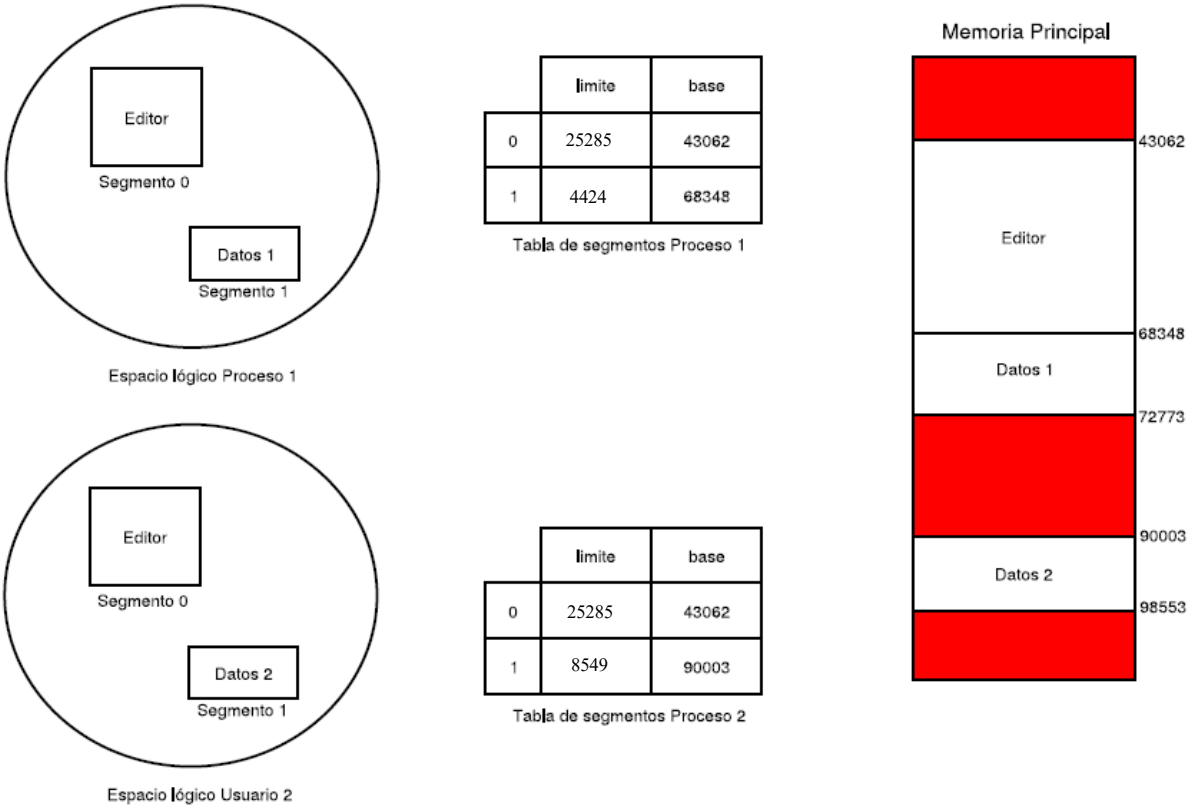
Estructuras de datos: cola de trabajos (necesidades de memoria), lista de huecos no asignados.

Algoritmo

- Si un trabajo tiene **n** segmentos, buscamos **n** huecos disponibles y se los asignamos:
  - First fit.
  - Best fit.
  - Otros esquemas.
- Cargamos cada segmento en el hueco que le corresponde.
- Actualizamos los valores de la tabla de segmentos.

Ventajas

- Protección de segmentos.
- Compartir segmentos.



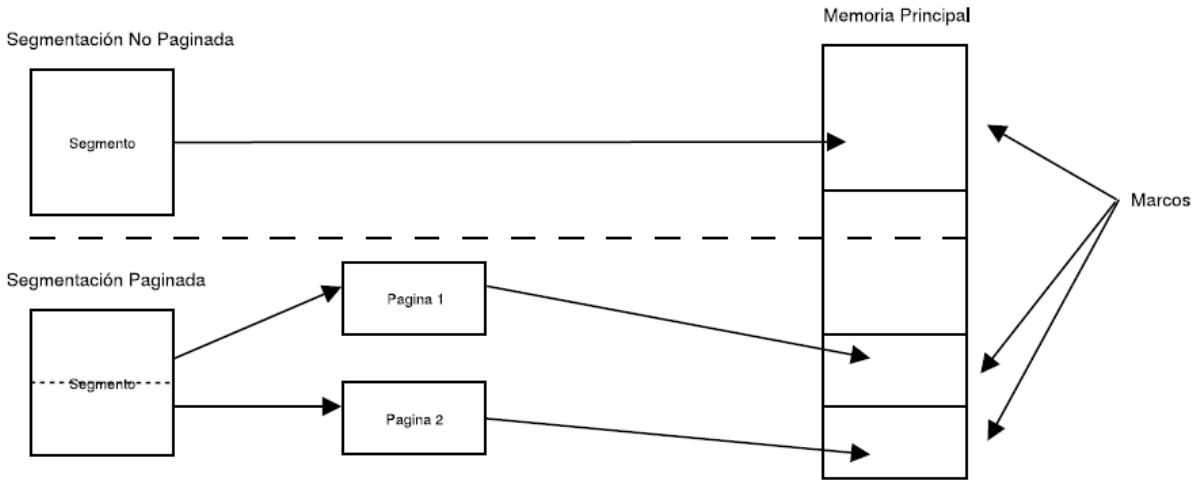
Inconvenientes

- Fragmentación externa.
- Gasto de tiempo en la búsqueda de un número de huecos adecuado.

5.3. Métodos combinados

Se beneficia de las ventajas que aporta tanto la paginación como la segmentación.

Reduce la fragmentación provocada por la segmentación mediante la paginación de cada segmento.



Tendremos **una tabla de páginas por cada segmento**, **cada entrada de la tabla de segmentos apunta a su tabla de páginas**.

Tabla de segmentos

El SO mantiene **una tabla de segmentos por proceso** cuyo objetivo es almacenar la información necesaria para localizar las tablas de páginas.

La estructura de esta tabla es:

	@Base	límite
0		
1		
..		
N		

- **@Base** es la **dirección física donde está la tabla de páginas del segmento.**
- **Límite** es el número de posiciones del segmento (realmente se pone el número de posiciones del segmento -1).

Tabla de páginas

El SO mantiene **una tabla de páginas por cada segmento** que forme el proceso, su objetivo es almacenar a qué marcos están asignadas las páginas.

Esta tabla está indexada por páginas.

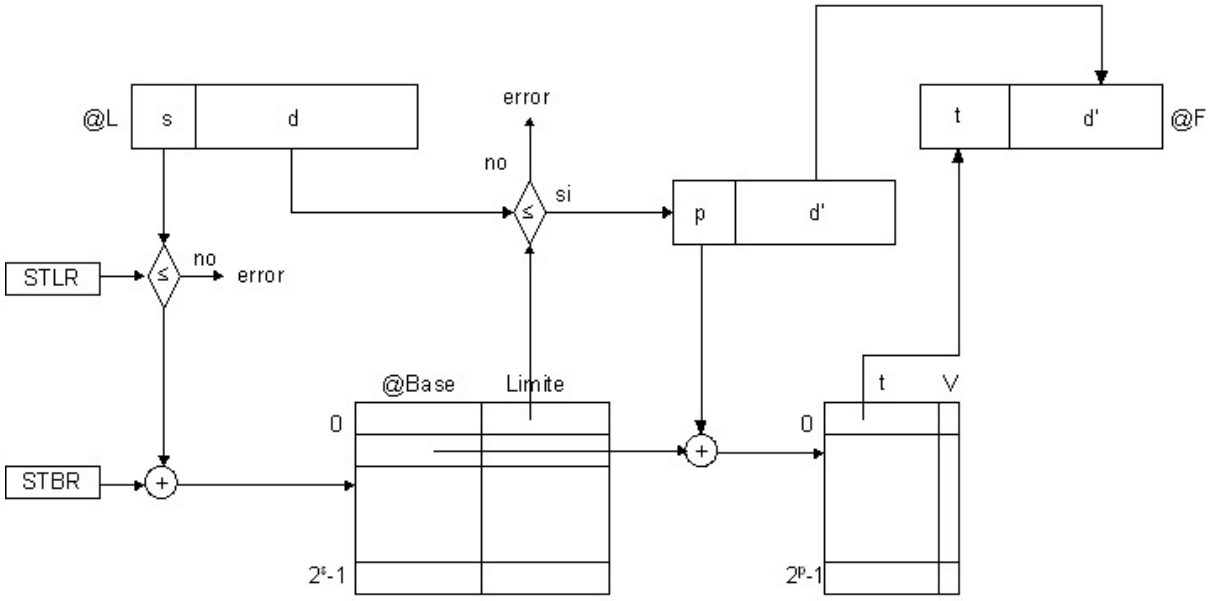
La estructura de esta tabla es la siguiente:

	t	BV
0		
1		
..		
N		

- **t** es el número marco que contiene la página.
- **BV** es un *bit* de validez

Traducción de direcciones, hardware de segmentación paginada

El esquema de traducción de una instrucción es:



Los bits más significativos (**s**) indican el número de segmento al que estamos accediendo.

A este valor se le suma **STBR** para acceder a la posición adecuada de la tabla de segmentos.

De esta posición obtenemos:

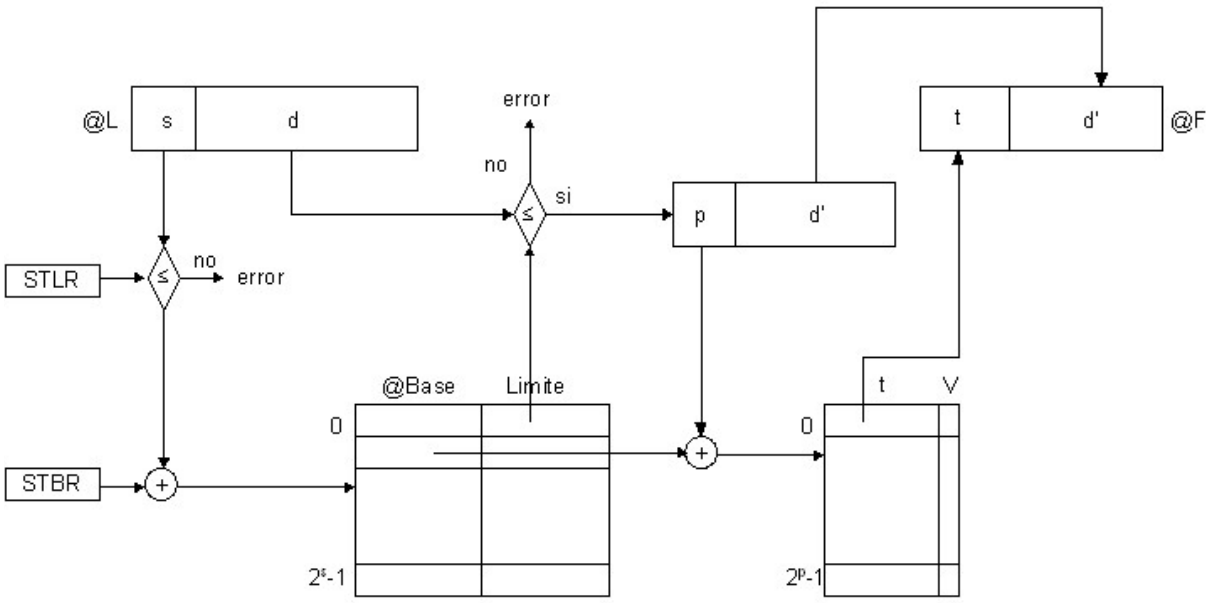
- El límite del segmento que lo deberemos comparar con **d** para asegurarnos que accedemos a una posición adecuada del segmento (d <= limite).
- La posición base **(@Base)** de memoria donde se encuentra la tabla de páginas del segmento **s**.

Los bits menos significativos (**d**) se dividen en dos campos: los más significativos (**p**) indican la página y los menos (**d'**) el desplazamiento dentro de esta.

Al valor de **p** se le suma la posición base (**@base**) de la tabla de páginas para obtener el número del marco (**t**).

A este número se le concatena **d'** para acceder a la posición física adecuada.

Ejemplo 1: Si tenemos una MMU que trabaja con segmentación paginada, direcciones lógicas de 28 bits, la memoria principal es de 16 Mb, el tamaño de la página es de 1 Kb y el tamaño máximo de un segmento es 1 Mb. Indicar la longitud en bits que se necesitan para los distintos elementos de la MMU:



Si MP es de 16 Megas ⇒el tamaño de las direcciones físicas y del campo @Base de la tabla de segmentos es:

@Base = *t*+*d*' = 24.

Si el tamaño máximo de una página es 1Kbyte el número de bits que necesitamos para referenciar a cualquier posición de la página es:

d' = 10

Si el tamaño máximo de un segmento es 1 Mega el número de bits que necesitamos para referenciar a cualquier posición del segmento es:

limite = *d* = 20

A partir de estos valores obtenemos el resto:

s=28−*d*=28−20=8 *t*=@*base*−*d*'=24−10=14 *p*=*d*−*d*'=10

Como el STBR apunta a cualquier dirección de memoria y nuestra memoria es de 16 Mb, este registro mide 24 bits.

Ventajas

- Protección de segmentos sin fragmentación externa.

Inconvenientes

- Necesitamos tres accesos a memoria.
- Puede producir fragmentación interna.

6. Memoria virtual

Todos los esquemas vistos hasta ahora exigen que el programa esté entero en memoria.

El principio de cercanía indica que probablemente cuando se ejecute un programa nunca se vaya a ejecutar todo su código.

Si tenemos esquemas con:

- Reubicación dinámica.
- División del espacio lógico del proceso.

No es necesario tener todos los fragmentos del programa en memoria principal para ejecutarse.

A la parte del programa residente en memoria principal se la llama **conjunto residente**.

Cuando el proceso encuentra una dirección lógica que no está en la memoria principal, se genera una excepción que provoca la carga del fragmento deseado.

Ventajas

- El espacio lógico del programa puede ser mayor que la memoria física disponible.
- Aumenta el grado de multiprogramación.
- Aumentará la velocidad de ejecución.

Inconvenientes

- La máxima velocidad de ejecución de un programa en un sistema con memoria virtual no puede ser mayor a la obtenida en un sistema sin memoria virtual.
- Se requiere hardware y software especial.

En los siguiente puntos veremos el software del sistema operativo que se encarga de realizar:

- La política de lectura.
- La política de ubicación.
- La política de reemplazo.
- Gestión del conjunto residente.
- Política de vaciado.
- Control de carga.

6.1. Paginación por demanda

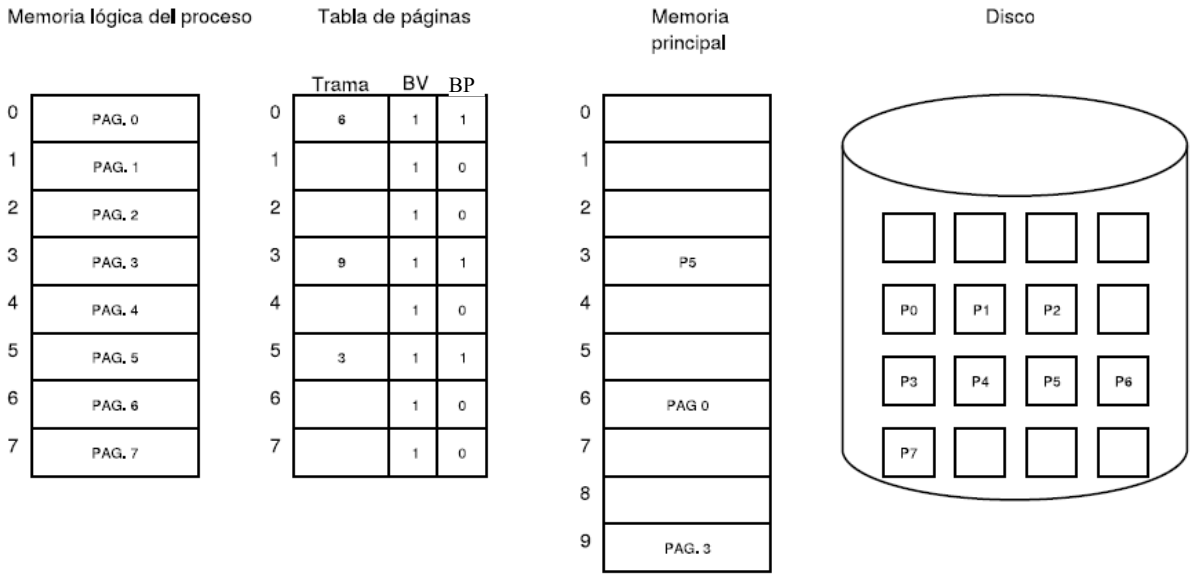
Cuando se habla de memoria virtual se asocia, normalmente, a sistemas que emplean paginación con intercambio.

Los programas se encuentran en un dispositivo auxiliar y tienen que ser cargados en memoria.

¿Cuándo se debe cargar una página en la memoria principal?:

- **Paginación por demanda pura**, se trae una página sólo cuando se hace referencia a una posición en dicha página.
- **Paginación previa**, se carga la página que se necesita y algunas más.

Ejemplo: Estado típico de la memoria en un gestor de paginación por demanda.



Es necesario que en la tabla de páginas se almacene un bit de presencia BP (0 indicará que no está presente en memoria principal).

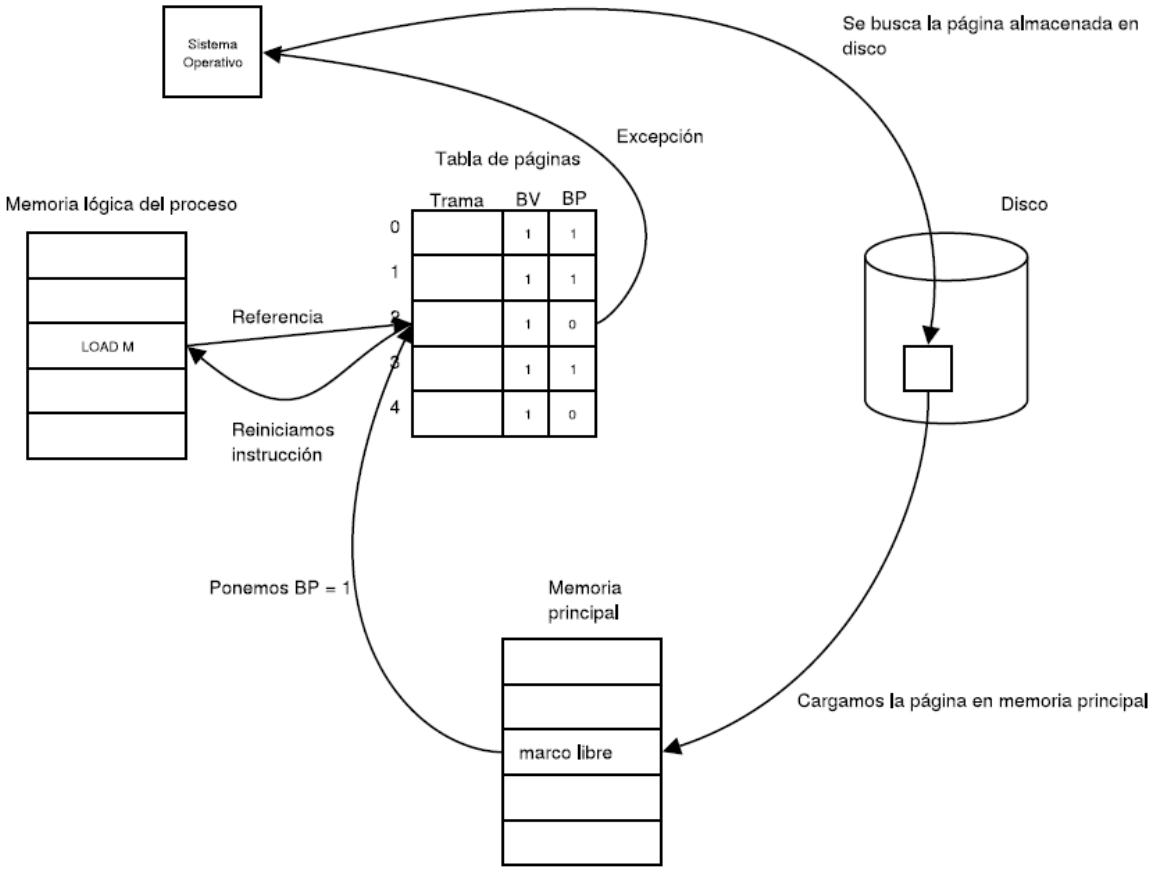
Realmente en los SO actuales se usa el mismo bit de validez para indicar presencia:

- Si el BV vale 1 sabemos que la página está presente.
- Si el BV vale 0 la página no está en memoria principal pero no sabemos si la página es válida. En el PCB del proceso se guardará información sobre las páginas que son válidas para el proceso.

Cuando se intente acceder a alguna página no cargada se produce un fallo de página.

¿Qué pasos se realizan ante un fallo de página?

- Se detecta si el error es realmente un fallo de página o una dirección inválida.
- Si es una dirección inválida finalizo.
- Si es por fallo de página, la buscamos en memoria secundaria.
- Le asignamos un marco de memoria principal.
- Cargamos el contenido de la página en el marco.
- Ponemos el bit de presencia de la tabla a 1 para indicar que la página está en memoria.
- Reiniciamos la ejecución de la instrucción.



6.2. Hardware necesario

Para la traducción necesitamos añadir a la tabla de páginas un bit de presencia **BP** para detectar las páginas cargadas en memoria.

	t	BV	**BP**
0			
1			
N			

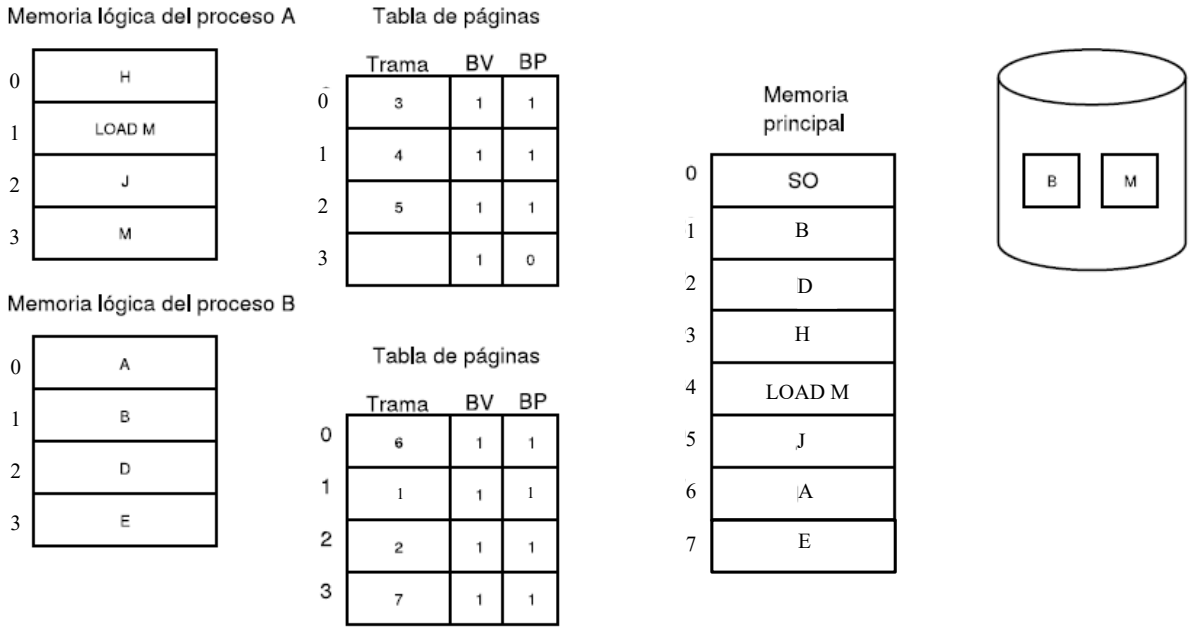
Un dispositivo auxiliar de alta velocidad. Además las operaciones de E/S se hacen mediante controladores DMA (Acceso Directo a Memoria).

Hardware necesario para distinguir entre fallo de página y la interrupción por direccionamiento incorrecto.

Hardware para poder reiniciar la instrucción interrumpida.

6.3. Reemplazo de páginas

¿Qué pasa cuando se da un fallo de página y no hay espacio suficiente en memoria para cargar la página?



La solución la encontramos en el reemplazo de páginas:

- Encontramos una trama que no se esté usando en ese momento y se libera.
- Al liberarla copiamos su contenido en memoria secundaria.

La rutina de atención a la interrupción de fallo de página realiza lo siguiente:

- Busca una trama libre
    a. Si hay una trama libre se utiliza.
    b. De lo contrario, utilizar un algoritmo de reemplazo de páginas para elegir la trama que debe ser descargada.
    c. Guarda la trama elegida en el disco; modifica la tabla de tramas libres y la tabla de páginas.
- Carga la página deseada en memoria en la trama libre
- Modifica la tabla de tramas libres y la tabla de páginas.
- Reinicia la instrucción.

Un fallo de página puede provocar dos transferencias de páginas ⇒ aumenta el tiempo de acceso.

Para reducir este tiempo añadimos a la tabla de páginas un bit de modificación, *dirty bit*, **BM,** que nos indica si los valores que contiene el marco son los mismo que originalmente tenía la página. Sólo si este bit está a 1 se transferirá el marco a disco.

	t	BV	BP	**BM**
0				
1				
N				

El concepto de reemplazo de página está relacionado con:

- El número de marcos de página a asignar a un proceso (Aptdo. 6.5).
- Si el conjunto de páginas a considerar para el reemplazo debe limitarse a las del proceso (asignación local) que provocó el fallo de página o a todos los procesos (asignación global).
- Del conjunto de páginas seleccionadas, ¿cuál se elige para reemplazarla? (Aptdo. 6.4).

6.4. Algoritmos de reemplazo de páginas

Estos algoritmos buscan aquella página que será reemplazada.

Para evaluar este tipo de algoritmos se ejecuta una serie de referencias a memoria y se calcula el número de fallos de página.

La secuencia de referencias se puede generar de forma artificial o modelizando algún sistema real.

Nosotros usaremos la secuencia:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

y supondremos que contamos con 3 tramas.

6.4.1. Algoritmo óptimo

Selecciona la trama que **tardará más en volver a referenciarse**.

Es imposible saber cuál es esta trama ⇒ algoritmo irrealizable pero vale como referencia.

Ejemplo:

*	*	*	**		**		**			**	
7	7	7	**2**	2	2	2	2	2	2	2	2
	0	0	0	0	0	0	**4**	4	4	**0**	0
		1	1	1	**3**	3	3	3	3	3	3
7	0	1	2	0	3	0	4	2	3	0	3

	**				**		
2	2	2	2	2	**7**	7	7
0	0	0	0	0	0	0	0
3	**1**	1	1	1	1	1	1
2	1	2	0	1	7	0	1

9 Fallos, 6 Reemplazos

6.4.2. FIFO

Se reemplaza aquella página que lleve **más tiempo en memoria**.

Es fácil de implementar pero provoca sobrecarga en sistemas de tiempo compartido.

Ejemplo:

*	*	*	**		**	**	**	**	**	**	
7	7	7	**2**	2	2	2	**4**	4	4	**0**	0
	0	0	0	0	**3**	3	3	**2**	2	2	2
		1	1	1	1	**0**	0	0	**3**	3	3
7	0	1	2	0	3	0	4	2	3	0	3

	**	**			**	**	**
0	0	0	0	0	**7**	7	7
2	1	1	1	1	1	0	0
3	3	2	2	2	2	2	1
2	1	2	0	1	7	0	1

15 Fallos, 12 Reemplazos

Presenta la **anomalía de Belady**, un aumento de tramas en la memoria no siempre lleva consigo una disminución de los fallos de página.

Ejemplo: Secuencia 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 para 3 y cuatro tramas

*	*	*	**	**	**	**			**	**	
1	1	1	**4**	4	4	**5**	5	5	5	5	5
	2	2	2	**1**	1	1	1	1	**3**	3	3
		3	3	3	**2**	2	2	2	2	**4**	4
1	2	3	4	1	2	5	1	2	3	4	5

9 Fallos, 6 Reemplazos

*	*	*	*			**	**	**	**	**	**
1	1	1	1	1	1	**5**	5	5	5	**4**	4
	2	2	2	2	2	2	**1**	1	1	1	**5**
		3	3	3	3	3	3	**2**	2	2	2
			4	4	4	4	4	4	**3**	3	3
1	2	3	4	1	2	5	1	2	3	4	5

10 Fallos, 6 Reemplazos

6.4.3. Algoritmo LRU (Least Recently Used)

Usada Menos Recientemente. Sustituye **la página que se usó por última vez hace más tiempo**.

Es quizás la solución que más se aproxime a la óptima.

Ejemplo:

*	*	*	**		**		**	**	**	**	
7	7	7	**2**	2	2	2	**4**	4	4	**0**	0
	0	0	0	0	0	0	0	0	**3**	3	3
		1	1	1	**3**	3	3	**2**	2	2	2
7	0	1	2	0	3	0	4	2	3	0	3

	**		**		**		
0	**1**	1	1	1	1	1	1
3	3	3	**0**	0	0	0	0
2	2	2	2	2	**7**	7	7
2	1	2	0	1	7	0	1

12 Fallos, 9 Reemplazos

Este algoritmo no es fácil de poner en práctica, algunas formas de implementarlo son:

- **Contadores Hardware**: un nuevo campo en la tabla de páginas que almacena el instante (I) en el que accedemos a la página. Se reemplaza aquella página con contador menor. Es necesario un contador que se incremente con cada referencia a memoria.

	t	BV	BP	Contador
0				
1				
..				
N				

- **Pilas**: una pila de números de las páginas utilizadas. Cada vez que se accede a una página, su número se pone al principio de la pila (si ya estaba se borra antes). La página que se reemplaza es la que ocupa el fondo de la pila.

- **Matrices Hardware**: para una memoria de N tramas, el hardware debe mantener una matriz de NxN puestos a 0. Cuando se accede a marco K, el hardware pone a 1 todos los bits de la fila K y a 0 todos los de la columna K. Se reemplaza la página contenida en el marco cuya fila tiene un valor binario menor.

6.4.4. Otros algoritmos (Aproximaciones al LRU)

Se basan en el uso del **bit de referencia**. A cada entrada de la tabla de páginas se añade un bit de referencia **BR**:

	t	BV	BP	**BR**	BM
0					
1					
..					
N					

- Cada vez que se referencia una página BR=1.
- Se selecciona aquella trama con BR=0.
- Periódicamente todos los BR = 0.

NRU (Not Recently Used)

NRU o clases de páginas: usamos el bit de modificación y un bit de referencia. Busca reemplazar la página no recientemente usada. Las páginas se clasifican según los valores del BR y BM:

Grupo	BR	BM	
0	0	0	Ni referencia ni modificación reciente
1	0	1	Modificada hace tiempo
2	1	0	Se uso recientemente pero no se modificó
3	1	1	Se modificó recientemente

Se elige la página que pertenezca al grupo 0, luego las del 1 y así sucesivamente.

Ejemplo: W=escritura, en caso de igualdad eliminamos la que lleve más tiempo sin referenciarse

*	*	*	**		**		**	**	**
7(1,0)	7 (1,0)	7 (1,0)	**2(1,0)**	2 (1,0)	2 (1,0)	2 (1,0)	**4(1,1)**	4 (1,1)	4 (1,1)
	0 (1,0)	0 (1,0)	0 (1,0)	**0 (1,1)**	0 (1,1)	**0 (1,1)**	0 (1,1)	0 (1,1)	0 (1,1)
		1 (1,0)	1 (1,0)	1 (1,0)	**3 (1,0)**	3 (1,0)	3 (1,0)	**2 (1,0)**	**3 (1,0)**
7	0	1	2	W 0	3	0	W 4	2	3

		**	**	**		**	**		**
4 (1,1)	4(1,1)	4 (1,1)	4 (1,1)	4 (1,1)	4 (1,1)	4 (1,1)	4 (1,1)	4 (1,1)	4 (1,1)
0 (1,1)	0 (1,1)	0 (1,1)	0 (1,1)	0 (1,1)	**0 (1,1)**	0 (1,1)	0 (1,1)	**0 (1,1)**	0 (1,1)
3 (1,0)	**3 (1,0)**	**2 (1,0)**	**1 (1,0)**	**2 (1,0)**	2 (1,0)	**1 (1,0)**	**7 (1,0)**	7 (1,0)	**1 (1,0)**
W 0	3	2	1	2	W 0	1	7	W 0	1

14 Fallos, 11 Reemplazos

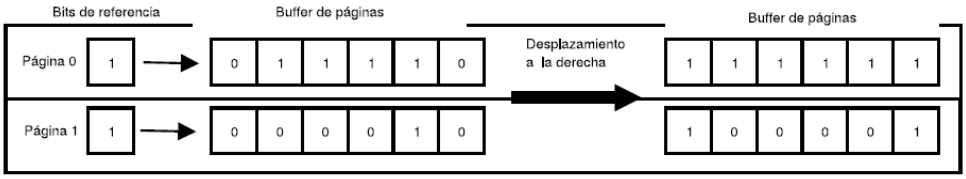
Algoritmo de envejecimiento (AGING)

Se tienen tantos buffer como páginas.

En estos se irán almacenando el histórico de bits de referencia de las páginas.

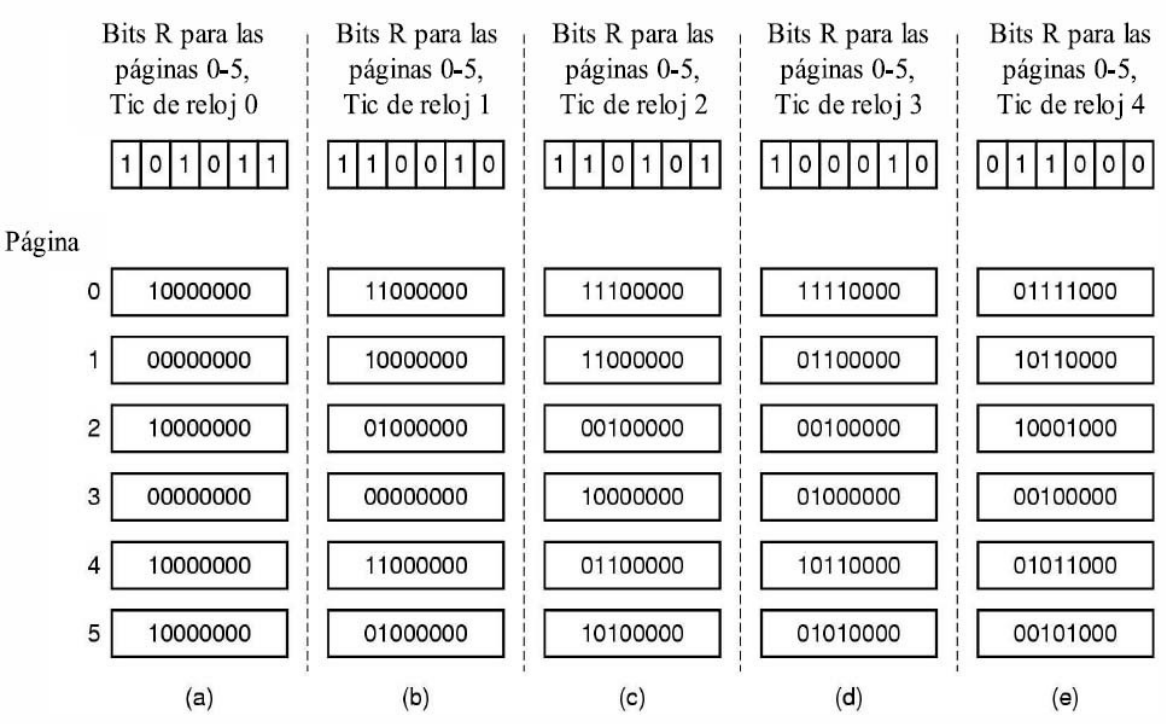
¿Cuándo se actualiza el contenido? Antes de poner a 0 los BR.

¿Cómo se actualiza? En cada uno de estos buffer se almacena el bit de referencia de la página correspondiente. El bit que entra ocupa la posición más significativa y se desplaza hacia la derecha a los bits ya existentes:



Cuando sea necesario reemplazar una página se seleccionará aquella cuyo buffer tenga un número en binario menor.

Ejemplo 1: Algoritmo AGING con 8 bits.



En t= 5 se produce un fallo de página, se reemplazara la página 3 que tiene un valor en bits inferior al resto.

Ejemplo 2: AGING con 3 bits

*	*	*	**		**		**	**	**	**	
7	7	7	**2**	2	2	2	**4**	4	4	**0**	0
	0	0	0	0	0	0	0	0	**3**	3	3
		1	1	1	**3**	3	3	**2**	2	2	2
7	0	1	2	0	3	0	4	2	3	0	3

	**		**		**		
0	**1**	1	1	1	1	1	1
3	3	3	**0**	0	0	0	0
2	2	2	2	2	**7**	7	7
2	1	2	0	1	7	0	1

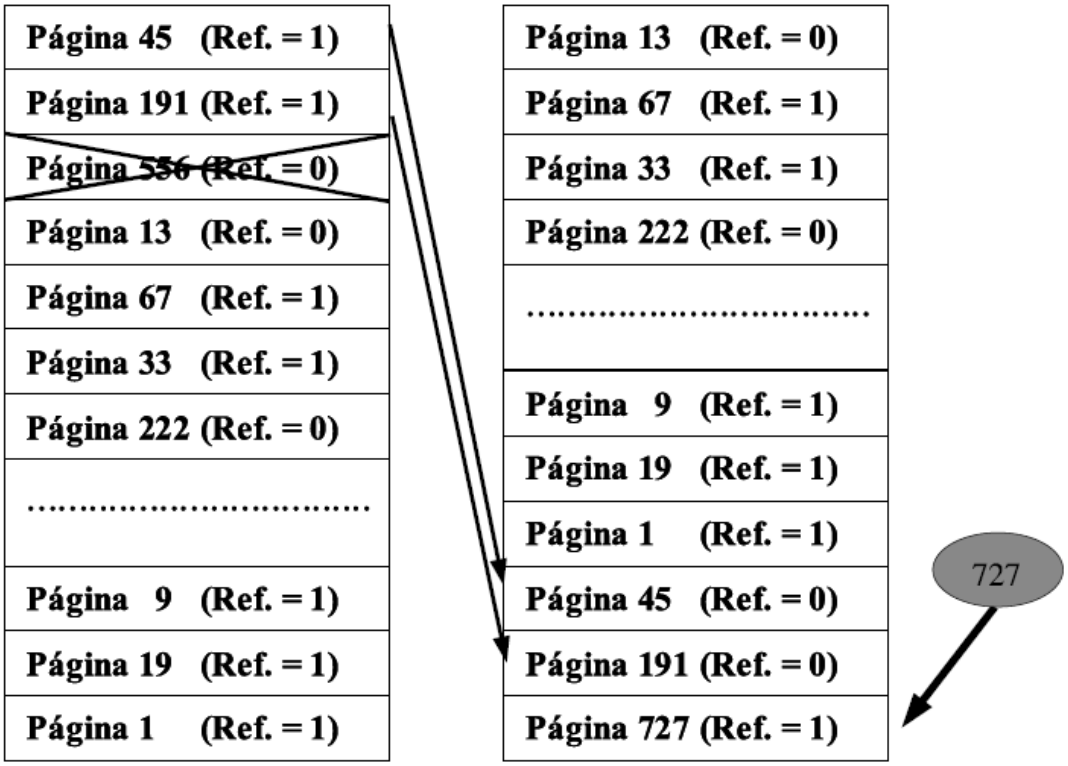
Estado de los buffers (se rellena de derecha a izquierda):

0			1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1
1			0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0
2			0	0	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0	0

Algoritmo del reloj

Funciona como la estrategia FIFO salvo una excepción, la página que entró hace más tiempo sólo se reemplazará si su bit de referencia vale 0. Si vale 1 la página se pondrá al final de la cola con el bit a 0 (se le da una segunda oportunidad) y se sigue buscando otra página.

Ejemplo: Acceso a la Página 727. Hay que reemplazar.



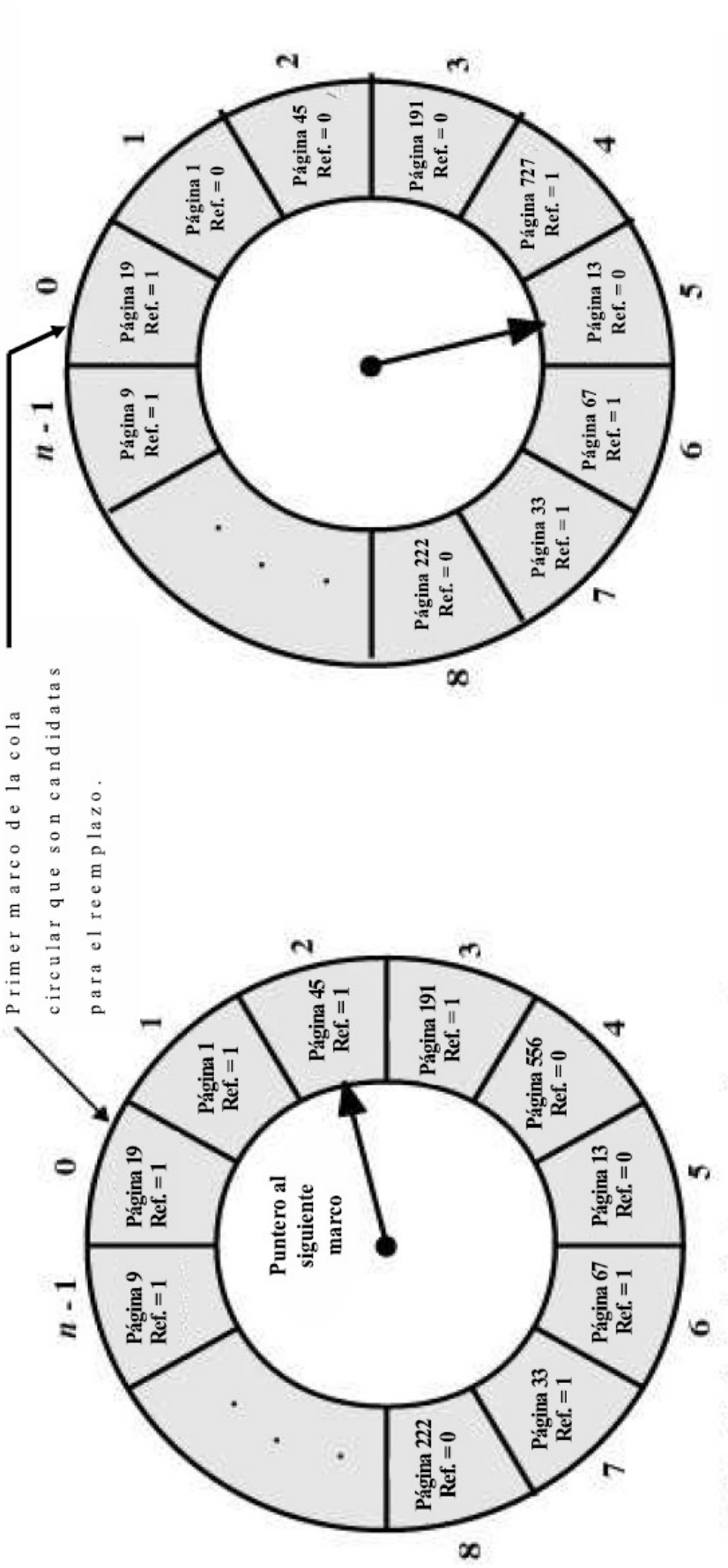
A este algoritmo se le llama del reloj porque se suele representar mediante un buffer circular.

Cada vez que se reemplaza una página el puntero señala a la siguiente página en el buffer (primera en la cola).

Cuando hay que reemplazar el puntero recorre la lista buscando una posición con el bit de referencia a cero.

Cada vez que el puntero pasa por una posición con el bit de referencia a uno lo pone a cero y sigue buscando.

Cuando encuentra una posición a 0 realiza el reemplazo de páginas y mueve el puntero a la siguiente posición del buffer:



6.4.5. Otros algoritmos (Algoritmos de conteo)

LFU (Least Frequently Used)

Menos frecuentemente usada.

Tenemos un contador en la tabla de páginas en la que vamos almacenando el número de referencia a cada página. Se elimina aquella con menos referencias.

	t	BV	BP	Contador
0				
1				
..				
N				

Ejemplo: Suponemos que a igualdad de referencias reemplazamos la que lleva más tiempo:

*	*	*	**		**		**	**	**
7 (1)	7 (1)	7 (1)	**2 (1)**	2 (1)	2 (1)	2 (1)	**4 (1)**	4 (1)	**3 (1)**
	0 (1)	0 (1)	0 (1)	0 (2)	0 (2)	**0 (3)**	0 (3)	0 (3)	0 (3)
		1 (1)	1 (1)	1 (1)	**3 (1)**	3 (1)	3 (1)	**2 (1)**	2 (1)
7	0	1	2	0	3	0	4	2	3

			**				**		
3 (1)	**3 (2)**	3 (2)	**1 (1)**	1 (1)	1 (1)	**1 (2)**	**7 (1)**	7 (1)	**1 (1)**
0 (4)	0 (4)	0 (4)	0 (4)	0 (4)	**0 (5)**	0 (5)	0 (5)	**0 (6)**	0 (6)
2 (1)	2 (1)	**2 (2)**	2 (2)	**2 (3)**	2 (3)	2 (3)	2 (3)	2 (3)	2 (3)
0	3	2	1	2	0	1	7	0	1

10 Fallos, 7 Reemplazos

MFU (Most Frequently Used)

Más frecuentemente usada.

Igual que la LFU pero se elimina aquella página con más referencias.

6.5. Políticas de asignación de tramas

La política por demanda de páginas no se aplica tal cual porque puede dar lugar a problemas de **hiperpaginación** (thrashing).

La hiperpaginación es el aumento incontrolado de faltas de página que se produce cuando un proceso no tiene asignadas el número de marcos suficiente para su correcta ejecución. Lleva a graves problemas de rendimiento.

En la práctica un proceso que se vaya a ejecutar se quedará con un mínimo de tramas.

Las políticas de asignación de tramas calculan cuál debe de ser ese mínimo.

Estas políticas se dividen en dos grandes grupos:

- **las fijas**: si no se calcula bien el número de marcos asignados, **el proceso se verá afectado** pero no el resto del sistema.
- **las variables**: si el número de marcos de páginas existentes en el sistemas no es el adecuado para almacenar los conjuntos de trabajo, **el sistema sufre hiperpaginación:**

a su vez las variables de dividen en locales y globales.

6.5.1. Políticas de asignación de tramas fijas

Asignan un número fijo de tramas a cada proceso.

Equipartición

Se reparte el número de tramas de la memoria por proceso (NTP) de forma equitativa entre todos los procesos en ejecución:

$$NTP = \frac{Tramas\ totales}{Num\ procesos}$$

Proporcional

Mayor número de tramas mientras mayor sea el proceso.

Prioritaria

Cuanto más prioritario es el proceso mayor número de tramas se asignan.

6.5.2. Políticas de asignación de tramas variables

Asignan un número variable de tramas a cada proceso.

Aquí estudiaremos:

- Working set
- Frecuencia de los fallos de página
- Control de carga para algoritmos de reemplazo **globales**.

Working set

Parte de la localidad de referencias de los procesos (los procesos se ejecutan por fases).

Define un conjunto de trabajo de un proceso en un instante *t* para un tamaño de muestra *s* como:

$$W(s,t)=np \quad en\ [t-s,\ t]$$

donde **np** es el número de páginas **diferentes usadas.**

El algoritmo se activa cada s instantes de tiempo.

Ejemplo:

Para un valor de s=10 y con las referencias a páginas:

Ref.	2	6	1	5	7	7	7	7	5	1	6	2	2	3	4	4	4	3	4	3	4	4	4	2
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

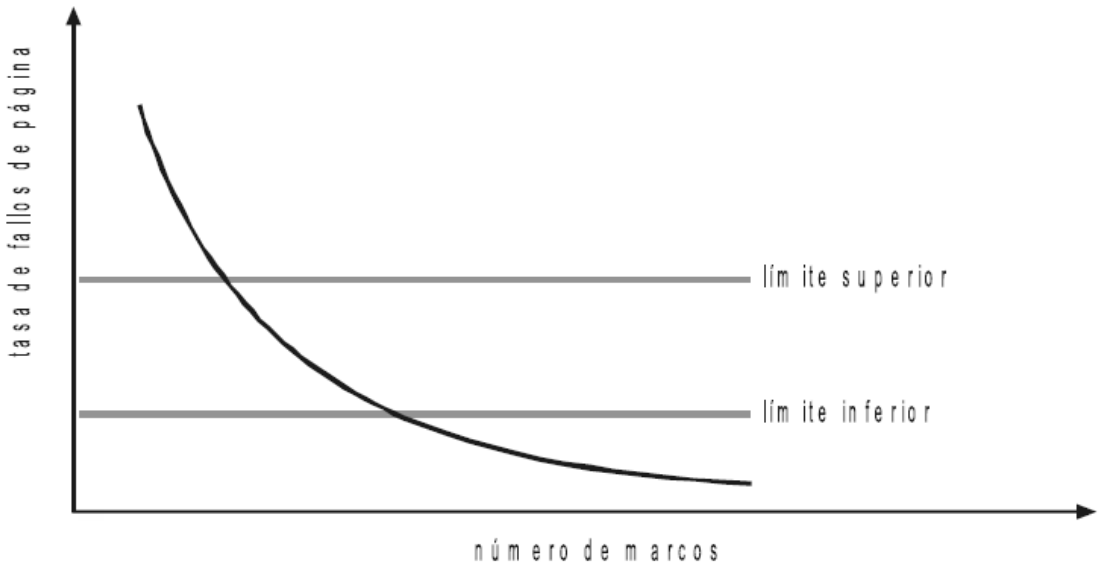
W(10,10)= 5 W(10,23)=2

Para evitar la hiperpaginación el SO:

- Determina los conjuntos de trabajo de los procesos activos.
- Mantiene estos conjuntos de trabajo en memoria.
- Si el conjunto de trabajo decrece se liberan los marcos que ya no forman parte de él.
- Si el conjunto de trabajo crece se le asignan nuevos marcos. Si no hay marcos libres se debe suspender algún proceso.

Frecuencia de los fallos de página

Se establece una cuota superior y otra inferior de la frecuencia de fallos de página de un proceso:



Si la frecuencia de fallos de página supera el límite superior se asignan nuevos marcos. Si no hay marcos libres se suspende un proceso.

Si la frecuencia de fallos de página es menor que el límite inferior, se liberan marcos.

Control de carga para algoritmos de reemplazo globales

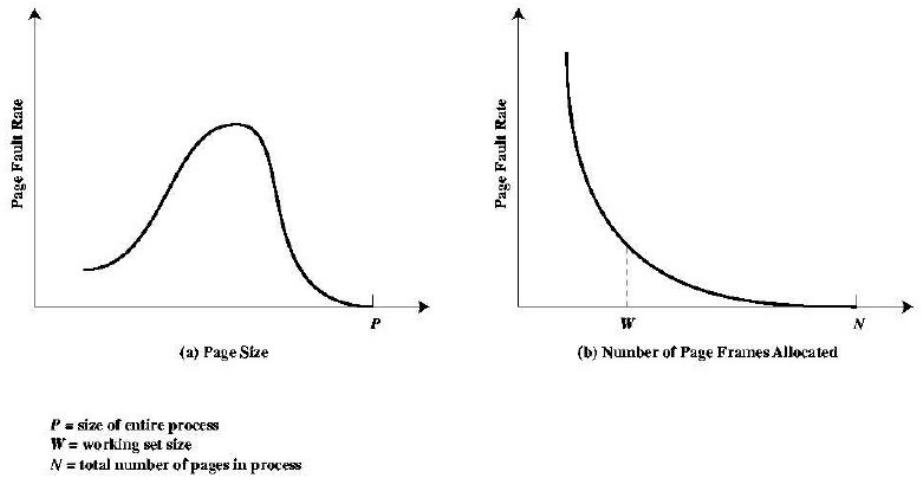
Un proceso (*page daemon*) comprueba periódicamente el número de marcos libres.

Si no hay suficientes aplica algún algoritmo de reemplazo global de páginas.

Si la tasa de paginación es alta y el número de marcos disponibles está frecuentemente por debajo del mínimo otro proceso denominado *swapper* se encarga de suspender procesos y liberar sus marcos de página.

6.6. Hiperpaginación y tamaño de página

El tamaño de la página también influye, un tamaño pequeño reduce la tasa de fallos:



7. Casos de estudio

En los siguientes puntos estudiaremos muy resumidamente la gestión de memoria en:

- Linux.
- Windows.

7.1. Linux

Se utiliza Memoria Virtual Paginada.

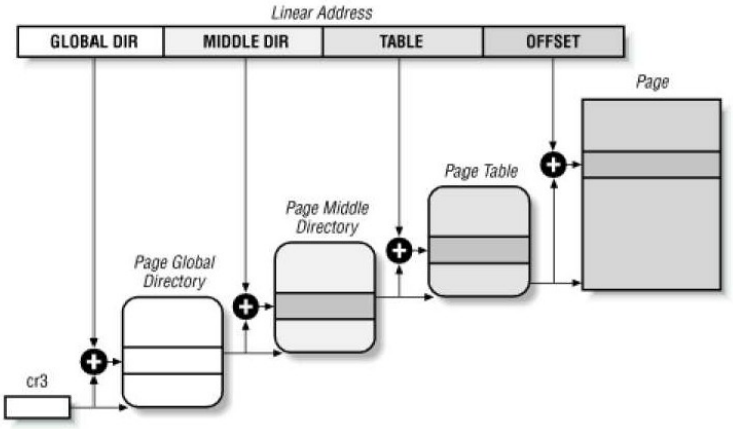
Linux hace un uso limitado de la segmentación (sólo las usa en arquitecturas i386).

Prefiere usar paginación a la segmentación por:

- La gestión de memoria es más simple.
- Linux busca la portabilidad y no todas las arquitecturas dan soporte para segmentos.

Gestión de la Memoria Virtual

Adopta un modelo de tres niveles lo que lo hace muy flexible en arquitecturas de 64 bits:



1. **Directorio de páginas:** Cada proceso activo tiene un único directorio de páginas del tamaño de una página. Cada entrada apunta a una página en el directorio intermedio de páginas. Siempre estará en memoria principal si el proceso está activo.

2. **Directorio intermedio de páginas:** Puede ocupar múltiples páginas. Cada entrada apunta a una página que contiene una tabla de páginas.

3. **Tabla de páginas:** Puede ocupar múltiples páginas. Cada entrada hace referencia al marco donde está situada la página.

Con direcciones de 64 bits se usa el esquema de 3 niveles explicado anteriormente, sin embargo, en plataformas de 32 bits, arquitecturas Intel, se usa un sistema de

paginación de 2 niveles. Linux se acomoda al esquema de 2 niveles definiendo el tamaño del directorio intermedio de páginas como 1.

Se considera que el Directorio de Páginas intermedio contienen una única página para mantener compatibilidad en 32 y 64 bits.

La páginas suelen ser de 4 Kb.

Algoritmo de reemplazo

Se basa en el algoritmo de reloj pero sustituye el bit de referenciada por un contador de 8 bits (bits de edad).

Cada vez que se accede a una página se incrementa el contador.

Durante el proceso de búsqueda vamos decrementando en 1 la edad de las páginas que no tengan una edad de 0.

Cuando se necesita reemplazar una página se busca aquella que tenga una edad=0.

El algoritmo de reemplazo de Linux es, por lo tanto, una variante de la política LRU.

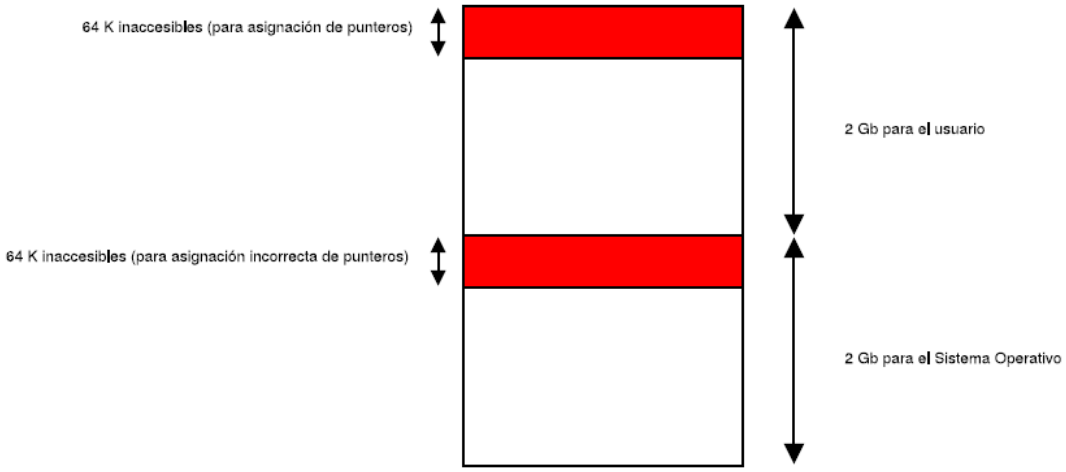
7.2. Windows

Usa memoria virtual con paginación por demanda con preasignación de un determinado número de tramas.

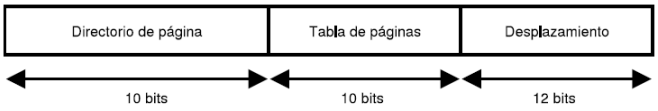
Usa tamaños de páginas entre 4 y 64 Kb.

En plataformas Intel el tamaño es de 4 Kb.

El espacio lógico por proceso tiene direcciones de 32 bits = 4 Gb:



Usa una estructura a dos niveles de directorios de página:



Asignación de páginas a procesos y algoritmo de Reemplazo

Se usa una política de asignación variable. Inicialmente se asigna un determinado número de tramas, aplicando posteriormente Working Set. Las páginas que se van a reemplazar se eligen de entre los marcos asignados al proceso (ámbito de reemplazo local).

¿Cómo se calcula el conjunto de trabajo?

- Cuando hay memoria disponible, el Conjunto de Trabajo de los procesos activos crece. Cuando se produce un fallo de página, se trae la nueva página a memoria sin expulsar una página antigua, incrementándose el conjunto residente del proceso en una página.
- Si la memoria escasea, el sistema operativo mueve las páginas que se han utilizado hace más tiempo, de cada uno de los procesos activos, a la zona de swap, reduciendo el tamaño de los conjuntos residentes de los procesos.

Si no se puede aumentar el conjunto residente de un proceso se usa reemplazo mediante LRU.