# Memoria Virtual

Se explican los motivos por los que se adopta una memoria virtual. Se estudia la paginación bajo demanda, los algoritmos de…

## Tabla de Contenidos



Autor: Lina García Cabrera

# 1 Introducción

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Las técnicas de **asignación no continua de memoria** permiten una nueva gestión de memoria: la **memoria virtual**.

La **Memoria Virtual** es un tipo de gestión de memoria que permite ejecutar procesos que no están ubicados por completo en memoria principal.
En este módulo se **justifican los motivos y ventajas** que nos reporta la ejecución de un programa parcialmente cargado en memoria.

Durante este módulo siempre hablaremos de la **paginación pero todo lo que se explica aquí es aplicable a la segmentación y, por supuesto, a la segmentación paginada** (porque finalmente los procesos son un conjunto de páginas).

Implementar la Memoria Virtual no es fácil. Los sistemas operativos deben minimizar la proporción de fallos de páginas que pueden ocurrir. La paginación bajo demanda será la respuesta más simple a la pregunta de cuándo llevar una página a memoria. Este tipo de política tiene un efecto inmediato en el rendimiento de un sistema, hay que intentar **reducir el tiempo efectivo de acceso a memoria**.
Cabe la posibilidad de que no exista ningún marco de página libre donde ubicar una nueva página. Esta decisión traerá consigo una ramificación de cuestiones, tales como:

- la **repercusión otra vez en el tiempo de acceso**, y
- el desarrollo de **algoritmos de reemplazo de páginas** y
- de **asignación de marcos**.

Para cada uno de estos algoritmos se realizará una reflexión sobre su funcionamiento, y valoración de su rendimiento.
Se deben evitar las situaciones que provocan reemplazo, y asignar un número de marcos adecuado a cada proceso para caer una altísima tasa de fallos de página denominada ***hiperpaginación***. Se puede paliar esta molesta problemática, por ejemplo con el **conjunto de trabajo**. Será necesario acudir al concepto de localidad, por ser el principio en el que se basa esta estrategia. Se concluye el estudio sobre la hiperpaginación proponiendo una aproximación más simple que tan sólo considera la **frecuencia de fallos de página**. Vistos todos los aspectos, estrategias y políticas relativas a la gestión de la memoria estaremos en condiciones de realizar una valoración sobre el parámetro **tamaño de página**.

**OBJETIVOS**

- Justificar el uso de un sistema de memoria virtual.
- Conocer la paginación bajo demanda, los algoritmos de reemplazo de páginas y asignación de marcos de páginas.
- Funciones del Sistema Operativo para minimizar el número de fallos de página.
- Saber las causas y las soluciones de la hiperpaginación.

## 2 Motivos y Ventajas

Ubicar el espacio de direcciones lógicas de un proceso por completo en memoria física antes de que el proceso se pueda ejecutar limita el tamaño de un programa al tamaño de la memoria física.

Al examinar programas reales nos percatamos de que, en muchos casos, no se requiere el programa completo.

- A menudo los programas contienen código para tratar **condiciones de error poco frecuentes**. Como en la práctica estos errores rara vez ocurren, o no se presentan, este código casi nunca se ejecuta.
- A las matrices, listas y tablas frecuentemente se les **asigna más memoria de la que necesitan realmente**. Pueden declararse matrices de 100 por 100 elementos, aunque pocas veces sean mayores de 10 por 10 elementos. Una tabla de símbolos de un ensamblador puede tener espacio para 3000 símbolos, aunque el programa típico tenga menos de 200 símbolos.
- **Ciertas opciones** y características de un programa **rara vez se usan**, como la opción de un editor que convierte los caracteres del texto marcado de minúsculas a mayúsculas o cálculos complejos de un hoja de cálculo.

Incluso en aquellos casos donde se necesita todo el programa, es probable que **no se requiera todo al mismo tiempo**.

La capacidad de ejecutar un programa que se encuentra parcialmente en memoria tendría varias ventajas:

- Un programa ya no estaría restringido por la cantidad de memoria física disponible. Los usuarios podrían **escribir programas** para un espacio de direcciones virtuales **muy grande**, simplificando las labores de programación.
- Como cada programa de usuario ocuparía menos memoria física, podrían ejecutarse más programas al mismo tiempo, **aumentando la utilización de la CPU** y la productividad, pero sin incrementar el tiempo de respuesta.
- Se **requeriría menos E/S** para cargar o intercambiar cada uno de los programas de usuario, por lo que se ejecutarían más rápido.

Un programa en ejecución que no se encuentre totalmente en memoria **beneficiaría tanto al usuario como al sistema**.

**La memoria virtual es la separación de la memoria lógica del usuario de la memoria física**. Esta separación permite proporcionar a los programadores una gran memoria virtual cuando sólo se dispone de una memoria física más pequeña. La memoria virtual facilita las tareas de programación, ya que el programador no tiene que preocuparse por la cantidad de memoria física disponible.

## 3 Estrategias de Gestión de Memoria Virtual

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Las **estrategias de gestión No Continua** (paginación, segmentación y segmentación paginada) de memoria son las que permiten implantar una **gestión virtual de la memoria**. Para cualquiera de las tres formas de organizar esta memoria virtual habrá que determinar:

- **Estrategias de obtención**. Determinan **cuándo se debe transferir** una página o un segmento del almacenamiento secundario al primario. Las estrategias de **obtención por demanda** esperan a que un proceso en ejecución haga referencia a una página o a un segmento antes de traerla/lo. Los esquemas de **obtención anticipada** intentan determinar por adelantado a qué páginas o segmentos hará referencia un proceso. Si la probabilidad de una referencia es alta y hay espacio disponible, entonces se trae al almacenamiento primario la página o segmento antes de que se haga la referencia explícita

- **Estrategias de colocación**. Determinan **en qué lugar de la memoria principal** se debe colocar una página o un segmento entrante. Los sistemas de paginación vuelven trivial la decisión de colocación, porque una página entrante se puede ubicar en cualquier marco de página disponible. Los sistemas con segmentación requieren estrategias de colocación como las tratadas en el contexto de los sistemas de multiprogramación con particiones dinámicas.
- **Estrategias de reemplazo**. Sirven para decidir **qué página o segmento se debe reemplazar** para dejar espacio a una página o segmento entrante cuando está completamente ocupada la memoria principal.

## 4 Paginación bajo Demanda

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Un sistema de paginación por demanda es similar a un sistema de paginación con intercambios. Los procesos residen en disco. Cuando queremos ejecutar un proceso, lo metemos en memoria. Sin embargo, en vez de intercambiar todo el proceso hacia la memoria, utilizamos un **intercambiador perezoso**.

> Un intercambiador perezoso nunca reincorpora una página a memoria a menos que se necesite.

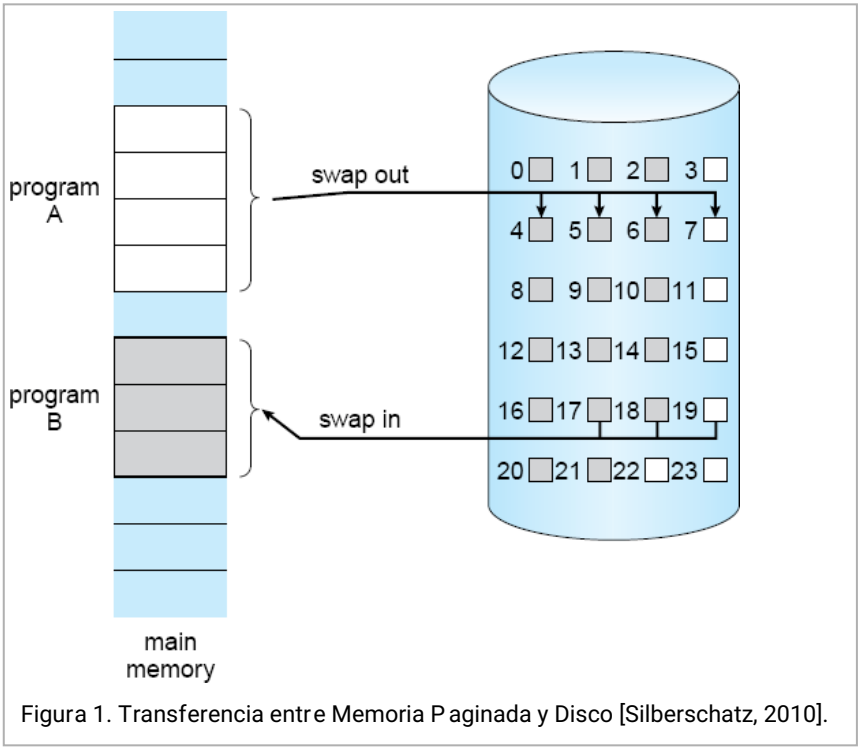


Figura 1. Transferencia entre Memoria Paginada y Disco [Silberschatz, 2010].

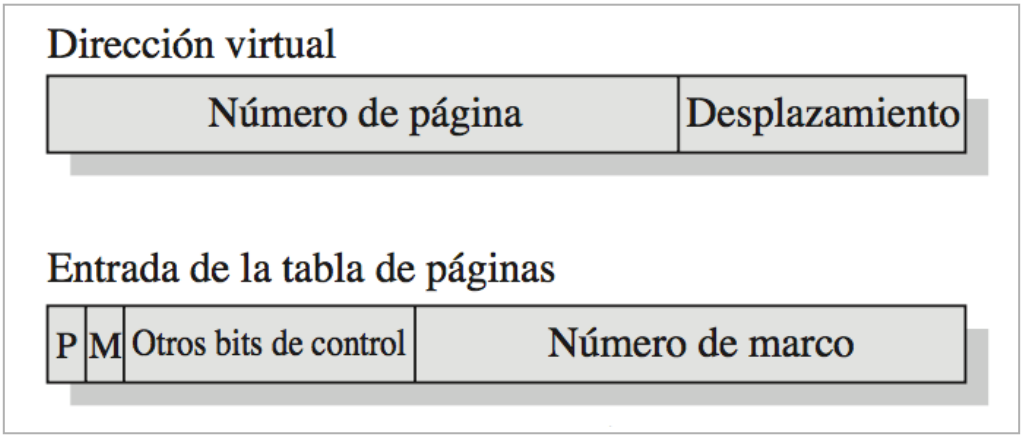
Ahora un proceso es una secuencia de páginas, en vez de un gran espacio contiguo de direcciones, el término intercambio es técnicamente incorrecto. Un intercambiador manipula procesos enteros, mientras que un paginador (***pager***) trata con las páginas individualmente de un proceso.

Cuando un proceso se reincorpora, el **paginador** lleva a memoria las páginas necesarias. Así evita colocar en la memoria páginas que no se utilizarán, reduciendo el tiempo de intercambio y la cantidad de memoria física necesaria.

Este esquema requiere **apoyo del hardware**. La *traducción de direcciones lógicas a físicas se hace por hardware*. Generalmente se añade un bit más a cada entrada de la tabla de páginas: un **bit válido-inválido o de presencia**. Cuando este bit está asignado como válido, indica que la página asociada se encuentra en memoria. Si el bit está como inválido, este valor indica que la página está en disco. Una página marcada como inválida no tendrá ningún efecto si el proceso nunca intenta acceder a ella.

# 4.1 Tabla de Páginas

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**




El mecanismo básico de **lectura de una palabra de la memoria** implica la **traducción de la dirección virtual, o lógica**, consistente en un número de página y un desplazamiento, a la dirección física, consistente en un número de marco y un desplazamiento, usando para ello la tabla de páginas.

Debido a que **la longitud de la tabla de páginas es variable dependiendo del tamaño del proceso**, no podemos suponer que se encuentra almacenada en los registros. En lugar de eso, debe encontrarse en la memoria principal para poder ser accedido.

En la mayoría de sistemas, existe una **única tabla de página por proceso**. Pero **cada proceso puede ocupar una gran cantidad de memoria virtual**.

> Por ejemplo, en la arquitectura VAX, cada proceso puede tener hasta $2^{31}$ = 2 Gbytes de memoria virtual. Usando páginas de $2^9$ = 512 bytes, que representa un total de $2^{22}$ entradas de tabla de página por cada proceso.

## Tabla de páginas en dos niv eles

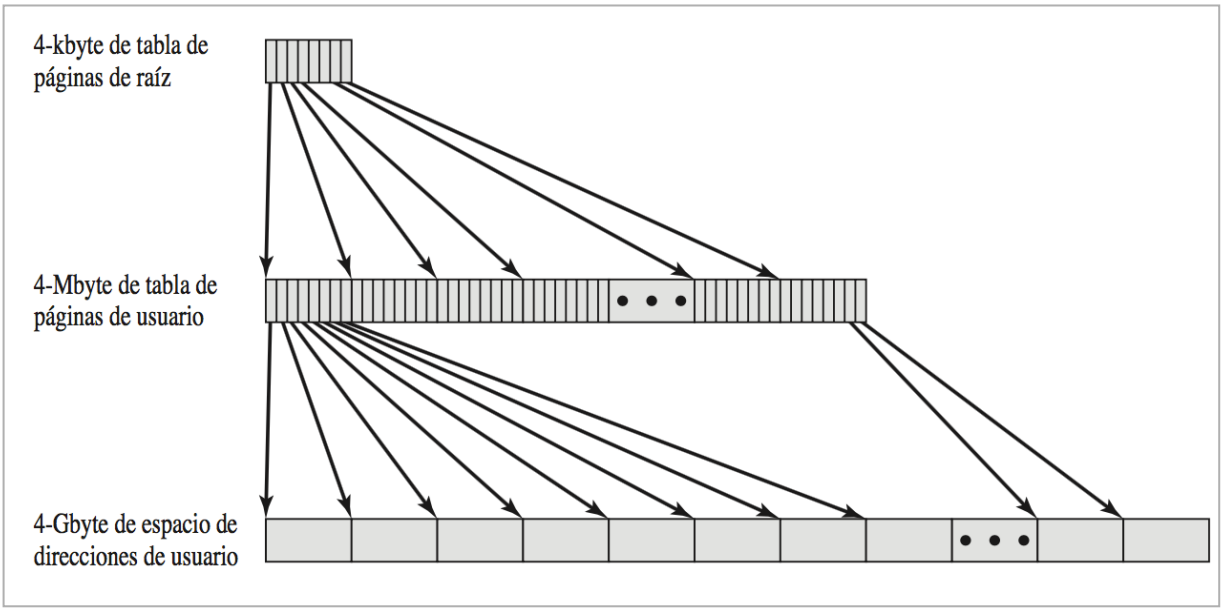
Evidentemente, la cantidad de memoria demandada por las tablas de página únicamente puede ser inaceptablemente grande. Para resolver este problema, la **mayoría de esquemas de memoria virtual almacena las tablas de páginas también en la memoria virtual**, en lugar de en la memoria real.

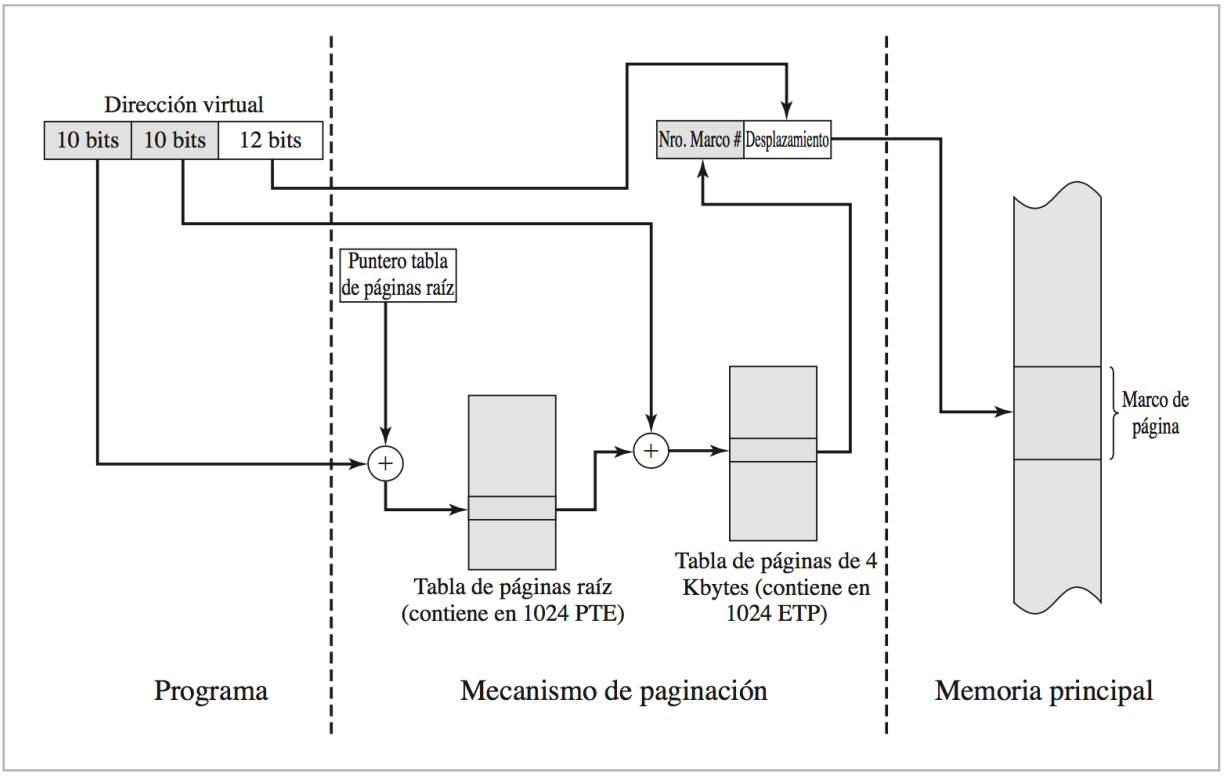
Las **tablas de páginas están sujetas a paginación** igual que cualquier otra página. Cuando un proceso está en ejecución, al menos **parte de su tabla de páginas debe encontrarse en memoria**, incluyendo **la entrada de tabla de páginas de la página actualmente en ejecución**.

Algunos procesadores utilizan un **esquema de dos niveles para organizar las tablas de páginas de gran tamaño**. En este esquema, existe un **directorio de páginas**, en el cual **cada entrada apuntaba a una tabla de páginas**. Normalmente, la longitud máxima de la tabla de páginas se restringe para que sea igual a una página.

> Supongamos, un ejemplo de un esquema típico de dos niveles que usa `32 bits` para la dirección. Asumimos un **direccionamiento a nivel de byte** y páginas de `4 Kbytes` ($2^{12}$), por tanto el espacio de direcciones virtuales de `4 Gbytes` ($2^{32}$) se compone de $2^{20}$ páginas.
>
> Si cada una de estas páginas se referencia por medio de una entrada la tabla de páginas (ETP) de 4-bytes, podemos crear una tabla de página de usuario con $2^{20}$ la ETP que requiere 4 Mbytes ($2^{22}$ bytes). Esta enorme tabla de páginas de usuario, que ocupa $2^{10}$ páginas, puede mantenerse en memoria virtual y hacerse referencia desde una tabla de páginas raíz con $2^{10}$ PTE que ocuparía 4 Kbytes ($2^{12}$) de memoria principal.
>
> La Figura muestra los pasos relacionados con la traducción de direcciones para este esquema. La página raíz siempre se mantiene en la memoria principal. Los primeros 10 bits de la dirección virtual se pueden usar para indexar en la tabla de páginas raíz para encontrar la ETP para la página en la que está la tabla de páginas de usuario. Si la página no está en la memoria principal, se produce un fallo de

página. Si la página está en la memoria principal, los siguientes 10 bits de la dirección virtual se usan para indexar la tabla de páginas de usuario para encontrar la ETP de la página a la cual se hace referencia desde la dirección virtual original.




## Tabla de Páginas Invertida

Una desventaja del tipo de tablas de páginas que hemos visto es que su tamaño es proporcional al espacio de direcciones virtuales.

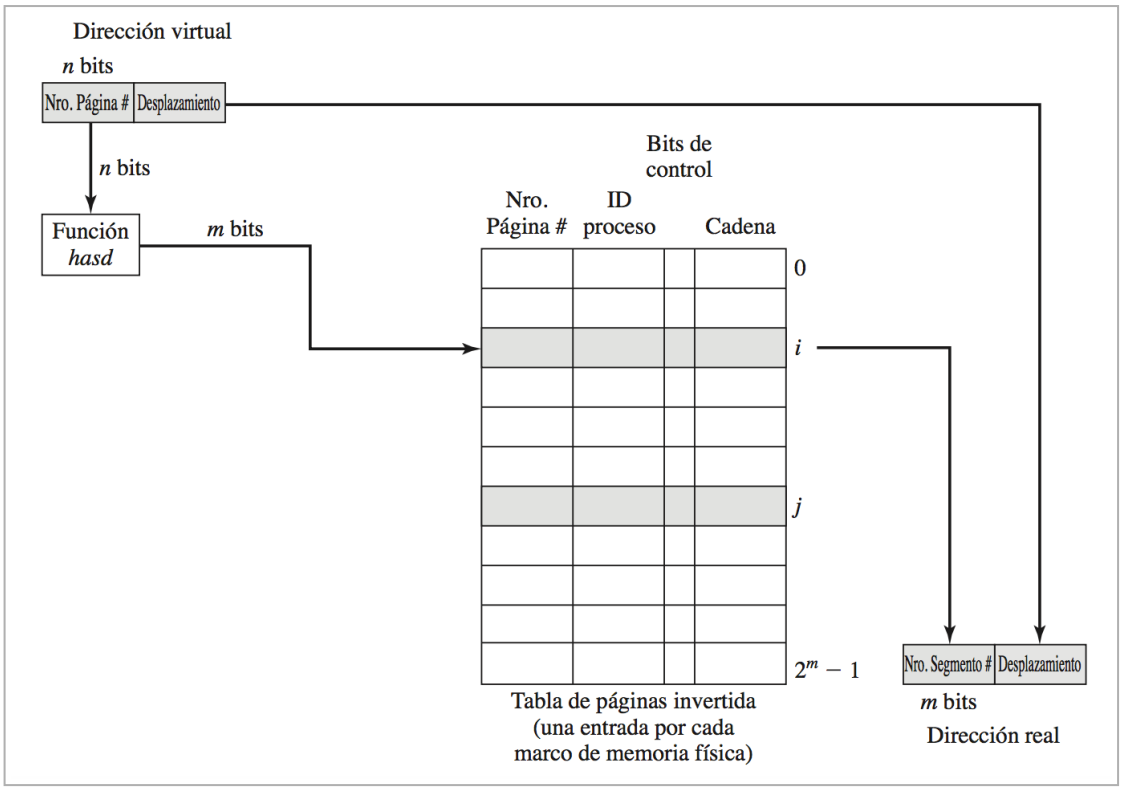
Una estrategia alternativa al uso de tablas de páginas de uno o varios niveles es el uso de la **estructura de tabla de páginas invertida**. En esta estrategia, la parte correspondiente al número de página de la dirección virtual se referencia por medio de un valor hash usando una función hash sencilla. El valor hash es un puntero para la tabla de páginas invertida, que contiene las entradas de tablas de página.

Hay **una entrada en la tabla de páginas invertida por cada marco de página real** en lugar de uno por cada página virtual. De esta forma, lo único que se requiere para estas **tablas de página siempre es una proporción fija de la memoria real**, independientemente del número de procesos o de las páginas virtuales soportadas.

Debido a que **más de una dirección virtual puede traducirse en la misma entrada de la tabla hash**, una técnica de encadenamiento se utiliza para gestionar el desbordamiento. Las técnicas de hashing proporcionan habitualmente cadenas que no son excesivamente largas —entre una y dos entradas. La estructura de la tabla de páginas se denomina invertida debido a que se indexan sus entradas de la tabla de páginas por el número de marco en lugar de por el número de página virtual.

La entrada en la tabla de páginas inclu- ye la siguiente información:

- **Número de página**. Esta es la parte correspondiente al número de página de la dirección virtual.
- **Identificador del proceso**. El proceso que es propietario de está página. La combinación de número de página e identificador del proceso identifica a una página dentro del espacio de direcciones virtuales de un proceso en particular.
- **Bits de control**. Este campo incluye los *flags*, como por ejemplo, válido, referenciado, y mo- dificado; e información de protección y cerrojos.
- **Puntero de la cadena**. Este campo es nulo (indicado posiblemente por un bit adicional) si no hay más entradas encadenadas en esta entrada. En otro caso, este campo contiene el valor del índice (número entre 0 y $2^{m-1}$) de la siguiente entrada de la cadena.




## 4.2 Gestión de un Fallo de Página



**¿Pero qué sucede si el proceso trata de usar una página que no se incorporó a la memoria?** Si adivinamos mal y el proceso trata de acceder a una página que no se trajo a memoria, ocurrirá una trampa de **fallo de página**.

El **hardware de paginación**, al traducir la dirección mediante la tabla de páginas, observará que el valor del **bit es inválido**, generando una trampa para el sistema operativo (error de dirección no válido). Normalmente, un error de dirección no válida es consecuencia de intentar utilizar una dirección de memoria ilegal; en este caso, el proceso deberá terminar. Sin embargo, en esta situación la trampa es el resultado del fallo de página del sistema operativo al no transferir a memoria una parte válida del proceso, tratando de minimizar el tiempo adicional de transferencia de disco y los requisitos de memoria. Por tanto, debemos corregir esta omisión.

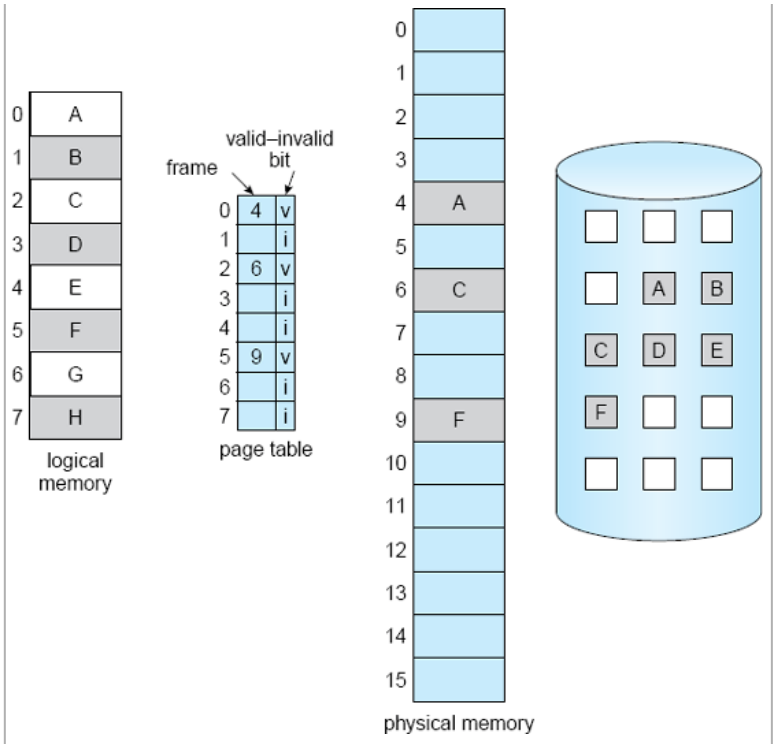



Figura 2. Tabla de páginas con páginas en disco [Silberschatz, 2010].

El procedimiento para **gestionar un fallo de página** (Figura 3):

1. **Consultamos una tabla interna** (que por lo general se conserva en el PCB del proceso) para determinar **si la referencia** fue un acceso a memoria fue **legal** (si es alguna de las páginas posibles del proceso).
2. **Si fue ilegal, abortamos el proceso**. Si se trató de una **referencia legal**, pero aún no hemos traído la página será **inválida**, la incorporamos.
3. Encontramos un **marco libre** (por ejemplo, seleccionando uno de la tabla de marcos libres).
4. Planificamos una **operación para leer de disco la página** deseada en el marco recién asignado.
5. Cuando ha concluido la lectura de disco, **modificamos la tabla interna** que se conserva junto con el proceso y la **tabla de páginas** para indicar que ahora la **página se encuentra en memoria** (bit de presencia o bit válido).
6. **Reiniciamos la instrucción interrumpida** por la trampa de dirección no-válida. El proceso ahora puede acceder a la página como si siempre se hubiera encontrado en memoria.

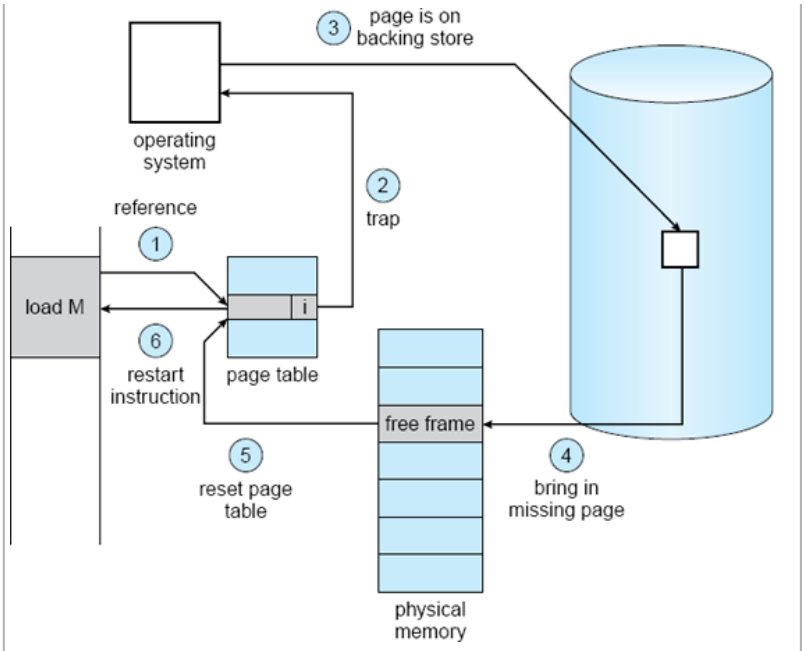


Figura 3. Pasos en la gestión de un F allo de Página [Silberschatz, 2010].

Como almacenamos el contexto (registros, código de condición, contador de instrucciones) del proceso interrumpido al ocurrir un fallo de página, podemos reanudarlo exactamente en el mismo punto y estado, excepto que ahora la página deseada se encuentra en memoria, y se puede acceder a ella.

El caso extremo es comenzar la ejecución de un proceso sin páginas en memoria. De inmediato, con la primera instrucción, el proceso presentaría una fallo de página. Después de traer de memoria esta página, el proceso continúa su ejecución, provocando fallos cuando sea necesario hasta que todas las páginas que necesita se encuentren en memoria. Esto es la paginación por demanda pura: no traer una página a memoria hasta que se requiera.

En teoría, algunos programas pueden acceder a una nueva página de memoria con cada instrucción que ejecutan, provocando posiblemente un fallo de página por cada instrucción. Esta situación provocaría un rendimiento inaceptable. Por fortuna, los análisis de procesos en ejecución han mostrado que esta situación es muy poco probable. Los programas suelen poseer una **localidad de referencias**, que brinda un rendimiento aceptable en la paginación bajo demanda.

## 4.3 Hardware paginación bajo demanda

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
El hardware para apoyar la paginación bajo demanda es el mismo que se usa para la paginación, segmentación y los intercambios:

- **Tabla de páginas**. Esta tabla tiene la capacidad para marcar una entrada como inválida usando un bit válido-inválido, o un valor especial de los bits de protección.
- **Memoria secundaria**. Esta memoria contiene las páginas que no se conservan en la memoria principal. La memoria secundaria casi siempre es un disco de alta velocidad y se dedica una zona llamada de intercambio (***swap space***) para este propósito.

Además de esta ayuda hardware, se requiere un considerable apoyo software, como veremos luego.
Hay que imponer algunas restricciones arquitectónicas. Un aspecto crítico es la capacidad para **reiniciar cualquier instrucción** después de un fallo de página. En la mayoría de los casos es fácil cumplir este requisito. Si falla la página al buscar la instrucción, podemos reiniciarla efectuando de nuevo la búsqueda. Si ocurre el fallo al buscar el operando, debemos buscar de nuevo la instrucción, decodificarla y luego buscar el operando.

> Como el peor de los casos, considere una instrucción de tres direcciones como ADD (suma) de A y B, colocando el resultado en C. Los pasos para ejecutar esta instrucción serían

1. Buscar y decodificar la instrucción ADD.
2. Buscar A.
3. Buscar B.
4. Sumar A y B.
5. Guardar el resultado en C.

Si se presentara un fallo al almacenar C, tendríamos que obtener la página deseada, traerla a memoria, corregir la tabla de páginas y reiniciar la instrucción. Este inicio implica buscar de nuevo la instrucción, decodificarla, buscar una vez más los dos operandos y volver a sumar. Esto significa repetir la instrucción.

La principal dificultad surge cuando una instrucción puede modificar varias localidades distintas.

Por ejemplo, considere una instrucción MVC (mover carácter), que puede mover hasta 256 bytes de una localidad a otra (que pueden coincidir parcialmente). Si alguno de los bloques (fuente o destino) sobrepasa un límite de página, puede ocurrir un fallo de página después de haber efectuado parte de la transferencia. Además, si los bloques fuente y destino se superponen es probable que se modifique el bloque fuente, por lo que no podríamos reiniciar la instrucción.

Este problema se resuelve de dos maneras:

- En una de las soluciones, el microcódigo calcula y trata de acceder a ambos extremos de los bloques. Si va a ocurrir una fallo de página, sucederá en esta etapa, antes de modificar algo. Entonces puede efectuarse la transferencia si sabemos que no ocurrirá ningún fallo de página, ya que todas las páginas en cuestión se encuentran en memoria.
- La otra solución utiliza registros temporales para contener los valores de las localidades sobrescritas. Si se presenta un fallo de página, todos los valores que había antes de que ocurriera la trampa vuelven a la memoria. Esta acción restablece la memoria a su estado anterior a la ejecución de la instrucción, por lo que podemos repetir la instrucción.

Estos no son los únicos problemas arquitectónicos que pueden surgir al añadir paginación, pero sí ilustran sus dificultades. En los sistemas de computación, la paginación debe ser completamente transparente para el proceso de usuario.

## 4.4 Rendimiento de la Paginación bajo Demanda



La paginación por demanda puede tener un efecto considerable en el rendimiento de un sistema de computación. Para ver la razón, calculemos el tiempo de acceso efectivo para la memoria paginada por demanda.

El tiempo de acceso a memoria, ***am***, para la mayoría de los sistemas de computación actuales va de 10 a 200 nanosegundos.

Mientras **no tengamos fallos de página**, el tiempo de acceso efectivo es igual al tiempo de acceso a memoria en un sistema paginado (que sería **dos accesos a memoria**, uno a la tabla de páginas y otro a la dirección de memoria). Sin embargo, si ocurre un **fallo de página**, primero debemos **leer de disco la página** indicada y luego acceder a la palabra deseada.

Sea **p** la probabilidad de un fallo de página ($0 \leq p \leq 1$). Podemos esperar que **p** tenga un valor muy cercano a cero; es decir, que habrá sólo unos cuantos fallos de página. Entonces, el tiempo efectivo de acceso es

**tiempo efectivo de acceso** = $(1 - p)$ x am + $p$ x tiempo de fallo de página

Para calcular este tiempo necesitamos saber cuánto tiempo se requiere para **servir un fallo de página**. Un fallo de página desencadena la siguiente secuencia:

1. Trampa para el sistema operativo.
2. Guardar el contexto del proceso.
3. Determinar si la interrupción fue un fallo de página.

4. Verificar que la referencia a la página haya sido legal y determinar la ubicación de la página en el disco.
5. Leer de disco a un marco libre:
    - a) Esperar en la cola del dispositivo hasta que se atienda la solicitud de lectura.
    - b) Esperar durante el tiempo de búsqueda y de latencia del dispositivo.
    - c) Comenzar la transferencia de la página al marco libre.
6. Durante la espera, asignar la CPU a otro proceso (planificación).
7. Interrupción del disco de fin de la E/S.
8. Guardar el contexto del otro proceso.
9. Determinar si la interrupción provino de disco.
10. Corregir la tabla de páginas y las demás tablas para indicar que la página deseada se encuentra en memoria.
11. Esperar a que la CPU se asigne nuevamente a este proceso.
12. Restablecer el contexto y la nueva tabla de páginas, y reanudar la instrucción interrumpida.

Es posible que no sean necesarios todos los pasos para todos los casos. Por ejemplo, en el paso 5 suponemos que la CPU se asigna a otro proceso mientras se efectúa la E/S. Esta disposición permite que la multiprogramación mantenga la utilización de la CPU pero requiere tiempo adicional para reanudar la rutina de servicio del fallo de página cuando concluye la transferencia de E/S.

En cualquier caso, nos enfrentamos a tres componentes principales del tiempo de servicio del fallo de página:

1. Servir la interrupción de fallo de página.
2. Leer la página.
3. Reanudar el proceso.

La primera y tercera tareas pueden reducirse a varios cientos de instrucciones. Estas tareas pueden costar de 1 a 100 microsegundos cada una. El tiempo de **cambio de página**, por otra parte, será cercano a los 8 milisegundos. Un disco de cabeza móvil tiene una **latencia** de 3 milisegundos, un tiempo de **búsqueda** de 5 milisegundos y un tiempo de **transferencia** de 0,05 milisegundo. Por tanto, el tiempo total de paginación será cercano a los 8 milisegundos, incluyendo tiempo de hardware y software. Recuerde que sólo consideramos el **tiempo de servicio** del dispositivo. Si una cola de procesos espera al dispositivo, tenemos que añadir el **tiempo de cola** mientras esperamos a que el dispositivo de paginación esté libre para atender nuestra solicitud, lo que incrementa aún más nuestro tiempo de intercambio.

Si tomamos como tiempo promedio de servicio de fallo de página 8 milisegundos y un tiempo de acceso a memoria total de 200 nanosegundos, entonces el tiempo efectivo de acceso, expresado en nanosegundos, es

**tiempo efectivo de acceso** = $(1 - p$ x 200 + $p$ x 8 miliseg = $(1 - p)$ x 200 + $p$ x 8000000= 200 + 7999800 x $p$.

Vemos entonces que el **tiempo efectivo de acceso** es directamente proporcional a la **tasa de fallos de página**. Si 1 acceso de cada 1000 provoca un fallo de página, el tiempo de acceso efectivo es de 8,2 microsegundos. El computador sería 40 por ciento más lento debido a la paginación por demanda. Si queremos una degradación menor al 10%, necesitamos

220 $>$ 200 + 7999800 x $p$,
20 $>$ 7999800 x $p$,
p $<$0.0000025

O sea, para mantener un nivel razonable de lentitud adicional provocada por la paginación por demanda, sólo podemos permitir que se tenga menos de un fallo de página por cada casi 400 mil accesos (399.990 accesos = 7999800/20). Es muy importante **mantener baja la tasa de fallos de página** en un sistema de paginación por demanda; de lo contrario aumenta el tiempo efectivo de acceso, frenando considerablemente la ejecución de los procesos.

Otro aspecto de la paginación por demanda es la gestión y **uso del espacio de intercambios**. La E/S de disco para el espacio de intercambios generalmente **es más rápida que con el sistema de ficheros**. Esto es así porque **se asigna espacio en bloques de mayor tamaño**, y **no se usan las búsquedas de ficheros ni los métodos de asignación indirecta**. Por tanto, es posible que el sistema obtenga una mejor productividad de la paginación copiando todo el contenido de un fichero al espacio de intercambios al iniciar el proceso, y luego efectuar la paginación por demanda desde el espacio de intercambios.

## 5 Reemplazo de Páginas



En nuestra presentación, la tasa de fallos no ha sido un problema serio, ya que, como máximo, hay un fallo de página por cada página cuando se hace referencia a ella por primera vez. Esto no es muy exacto.

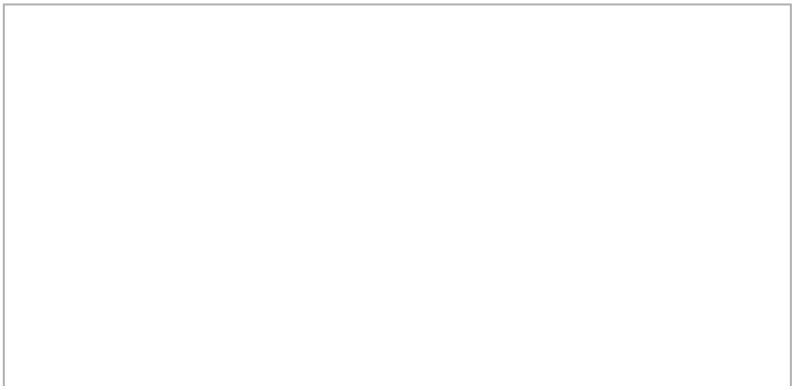
> Considere que, si un proceso de 10 páginas sólo emplea la mitad de ellas, entonces la paginación por demanda ahorra la E/S necesaria para 5 páginas que nunca se usarán. Podemos aumentar el nivel de multiprogramación ejecutando el doble de procesos. Así, si tuviéramos 40 marcos, podríamos ejecutar 8 procesos, en lugar de 4.
>
> Si aumentamos nuestro nivel de multiprogramación, estamos **sobreasignando la memoria**. Si ejecutamos 6 procesos, cada uno con un tamaño de 10 páginas, pero que sólo utilizan 5, tenemos mayor utilización de la CPU y productividad, con 10 marcos de sobra. Sin embargo, es posible que cada uno de estos procesos, para un conjunto de datos en particular, requiera sus 10 páginas, lo que supondría una necesidad de 60 marcos cuando tan sólo se dispone de 40.

Aunque esta situación puede ser poco probable, la posibilidad aumenta con el nivel de multiprogramación. Puede entonces darse el caso de que se **produzcan demandas de páginas para las que no hay marcos**. El sistema operativo podría abortar el proceso de usuario, pero la paginación dejaría de ser transparente, ya que, por razones desconocidas para el usuario se ha acabado con la ejecución de su proceso.

La solución está en **reemplazar páginas**. Si no hay ningún marco libre, encontramos uno que no se esté utilizando en ese momento y lo liberamos. Podemos liberar un marco escribiendo en disco todo su contenido y modificando la tabla de páginas (y todas las demás tablas) para indicar que la página ya no se encuentra en memoria (Figura 4). El marco liberado puede usarse ahora para contener la página por la que falló el proceso.

Ahora se modifica la **rutina de servicio de fallo de página** para incluir el **reemplazo de página**:

1. Encontrar la ubicación en disco de la página deseada.
2. Buscar un marco libre:
    - a) Si hay un marco libre, utilizarlo.
    - b) De lo contrario, utilizar un algoritmo de reemplazo de página con el fin de seleccionar un marco víctima.
    - c) Escribir la página víctima en disco; ajustar las tablas de marcos y páginas.
3. Leer la página deseada en el nuevo marco libre; modificar las tablas de páginas y marcos.
4. Reanudar el proceso de usuario.



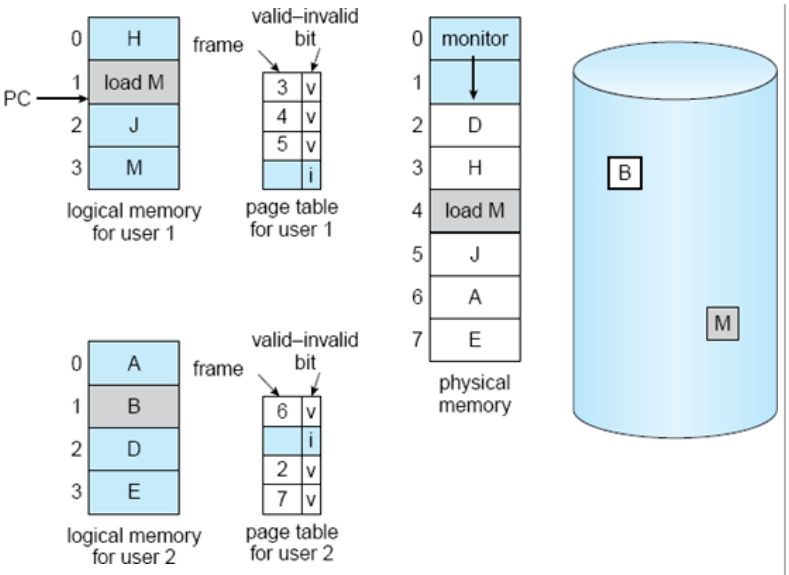


Figura 4. Reemplaz o de una página [Silberschatz, 2010].

Si no quedan marcos libres, se requieren **2 transferencias de páginas** (una de entrada y otra de salida). Esta situación **duplica el tiempo de servicio de fallo de página** y en consecuencia **aumentará el tiempo efectivo de acceso**.

Este tiempo de procesamiento adicional se puede reducir utilizando un **bit de modificación** (*dirty bit*). Cada página o marco puede tener asociado un bit de modificación en el *hardware*. El *hardware* pone a uno el bit de modificación para una página cuando en ella se escribe una palabra o un byte, indicando que ha sido modificada. Cuando seleccionamos una página para reemplazo, examinamos su bit de modificación. Si está activo, sabemos que la página se debe escribir en disco. Esta técnica también se aplica a páginas de sólo lectura. Con esto se reduce a la mitad el tiempo de E/S, si la página no ha sido modificada.

Para implantar la paginación por demanda debemos resolver dos grandes problemas:

- **desarrollar un algoritmo de asignación de marcos** y
- un **algoritmo de reemplazo de páginas**.

Si tenemos varios procesos en memoria, debemos decidir **cuántos marcos asignar a cada uno**. Además, **cuando se requiere un reemplazo de página es necesario seleccionar los marcos que se reemplazarán**. El diseño de algoritmos adecuados para resolver estos problemas es una tarea importante, ya que la E/S de disco es muy costosa. Una pequeña mejora en los métodos de paginación por demanda produce amplias ganancias en el rendimiento del sistema.

## 6 Algoritmos de Reemplazo

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Se buscan **algoritmos que presenten la menor tasa de fallos de página**. Para ello, necesitamos evaluarlos para una serie determinada de referencias a memoria y calcular el número de fallos de página.

Podemos generar las **cadenas de referencias** aleatoriamente o bien coger una muestra del sistema. Esto producirá un gran número de datos (alrededor de **1 millón de referencias por segundo**). Se puede reducir este número, ya que tan sólo hemos de considerar el **número de página y no toda la dirección**. Y además, si tenemos una referencia a una página $p$, entonces ninguna referencia a la página $p$ que se presente inmediatamente después provocará fallo de página. La página $p$, ya se encuentra en memoria, y las siguiente referencias a ella no provocan fallo de página. **Se podría producir un fallo de página (no siempre) cuando se cambia de página**.

> Por ejemplo, si rastreamos un proceso, podemos anotar la siguiente secuencia de direcciones: 0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0609,

0102, 0105, que, a 100 bytes por página, se reduce a la siguiente serie de referencias: 1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1.

Para determinar el número de fallos de página para una serie de referencias y un algoritmo de reemplazo de página, necesitamos también el número de marcos de página por proceso. Si aumenta el número de marcos, se reducirá el número de fallos de página.

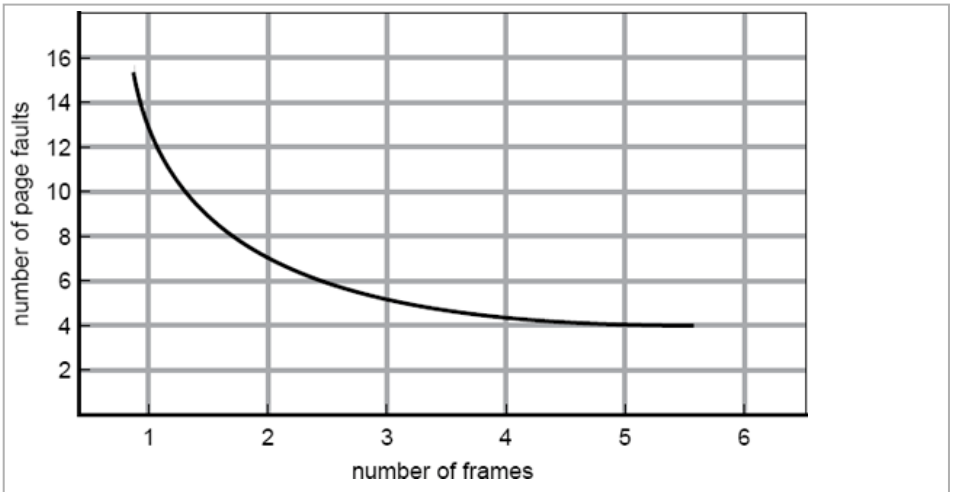


Figura 5. Gráfica Número de fallos de página/Número de Marcos [Silberschatz, 2010].

Para ilustrar los algoritmos de reemplazo de páginas, usaremos la siguiente secuencia de referencias 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

## 6.1 Algoritmo FIFO

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

El algoritmo de reemplazo más sencillo es el primero en entrar, primero en salir. Este algoritmo asocia a cada página el instante en que se trajo a memoria. Cuando hay que reemplazar una página, se elige la más antigua.

Para la cadena de referencias de ejemplo, y suponiendo un total de 3 marcos, aplicamos este algoritmo con el resultado de 15 fallos de página.

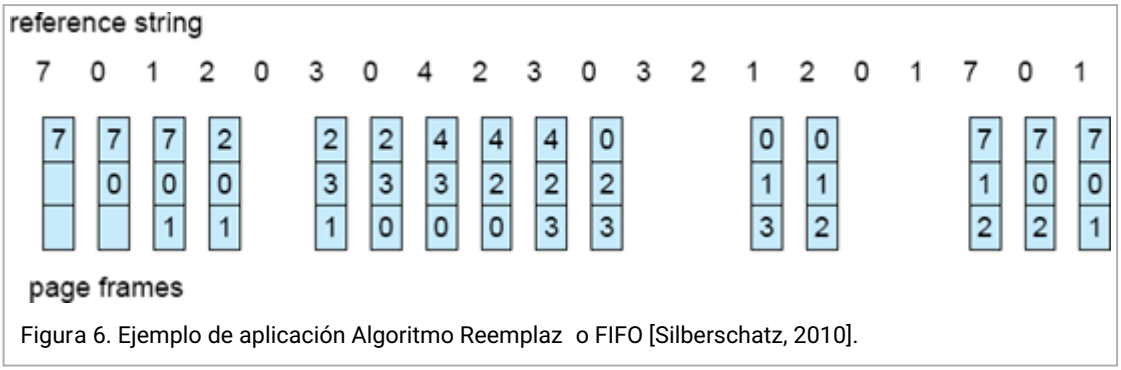


Figura 6. Ejemplo de aplicación Algoritmo Reemplazo FIFO [Silberschatz, 2010].

El algoritmo FIFO es fácil de comprender y programar, pero su rendimiento no siempre es bueno. La página reemplazada puede ser un módulo de asignación de valores iniciales que se utilizó hace mucho tiempo y que ya no se necesita. Pero también puede contener una variable cuyo valor inicial se asignó hace tiempo pero que se utiliza constantemente.

Además, este algoritmo presenta una irregularidad denominada **anomalía de Belady**. Esto significa que con el algoritmo FIFO la tasa de fallos pueden aumentar al incrementar el número de marcos asignados.

Para ilustrarlo suponga que ahora tenemos la siguiente serie de referencias 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. La Figura 7 muestra la curva de fallos de páginas frente al número de marcos disponibles.

Sorprendentemente el número de fallos para 4 marcos (10) es mayor que para 3 marcos (9).


Figura 7. Anomalía de Belady e el algoritmo FIFO [Silberschatz, 2010].

## 6.2 Reemplazo Óptimo

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Una consecuencia del descubrimiento de la anomalía de Belady fue la búsqueda de **un algoritmo de reemplazo de páginas óptimo**.

El algoritmo de reemplazo de páginas óptimo sería aquel que **eligiera la página de la memoria que vaya a ser referenciada más tarde**.

Por desgracia, **este algoritmo es irrealizable**, pues no hay forma de predecir qué páginas referenciará un proceso en el futuro. Como resultado de esto, el algoritmo óptimo **se utiliza sobre todo para estudios comparativos**. Puede ser útil para saber que un nuevo algoritmo está dentro del 12.3% de lo óptimo en el peor de los casos y dentro del 4.7% en promedio.

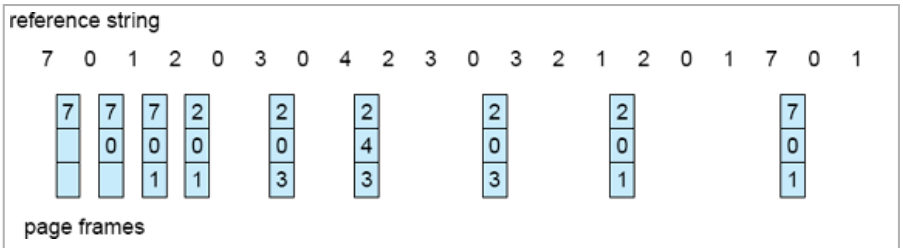



Figura 8. Ejemplo de aplicación Algoritmo Algoritmo Óptimo [Silberschatz, 2010].

## 6.3 Algoritmo Menos Recientemente Usado LRU (Least Recently Used)

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

La principal diferencia entre los algoritmos FIFO y OPT es que el primero utiliza el instante en que entró una página en memoria, y el segundo utiliza el tiempo en el que se usará la página.

Una buena **aproximación al algoritmo óptimo** se basa en el principio de localidad de referencia de los programas. Las páginas de uso frecuente en las últimas instrucciones se utilizan con cierta probabilidad en las siguientes instrucciones. De manera recíproca, es probable que las páginas que no hayan sido utilizadas durante mucho tiempo permanezcan sin ser utilizadas durante cierto tiempo. Es decir, estamos utilizando **el pasado reciente como aproximación de lo que sucederá en el futuro**. El algoritmo LRU explota esta idea: al ocurrir un fallo de página se utiliza la página que no haya sido utilizada hace más tiempo.

> El resultado de aplicar este algoritmo a nuestro ejemplo produce 12 fallos.

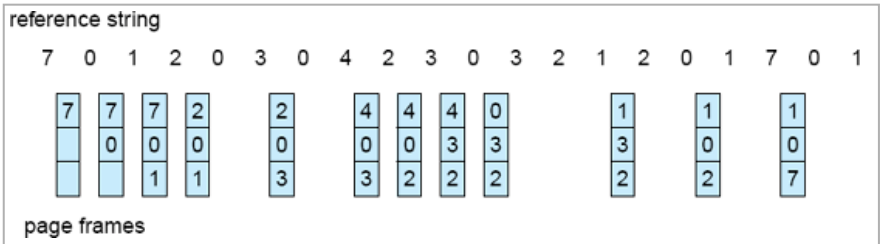



Figura 9. Ejemplo de aplicación Algoritmo Reemplaz o LRU [Silberschatz, 2010].

Ni el reemplazo óptimo ni el reemplazo LRU padecen la anomalía de Belady. Los algoritmos de reemplazo denominados **algoritmos de pila** nunca presentan esta anomalía.

Un **algoritmo de pila** es aquel que para el cual se puede demostrar que **el conjunto de páginas en memoria para $n$ marcos es siempre un subconjunto del conjunto de las páginas que estarían en memoria con $n+1$ marcos**.

Para el reemplazo LRU, el conjunto de páginas en memoria sería las $n$ páginas más recientemente usadas. Si el numero de marcos aumentara, estas $n$ páginas seguirían siendo las de más reciente referencia y, por tanto, seguirían en memoria.

## 6.3.1 Implementación de LRU



Aunque LRU es realizable teóricamente, **su implantación presenta problemas**. Sería necesario mantener **una lista enlazada** de todas las páginas en la memoria, en donde la página de uso más reciente esté al principio de la lista y la de uso menos reciente al final. La dificultad estriba en que la lista debe actualizarse en cada referencia a la memoria. La búsqueda de una página en la lista, su eliminación y posterior traslado al frente de la misma puede ser una operación que consuma mucho tiempo. Es preciso un *hardware* especial (caro), o bien determinar una solución aproximada mediante software.

La búsqueda y manipulación de una lista enlazada en cada instrucción es prohibitiva por su lentitud, aún en *hardware*. Sin embargo, existen otras formas de implantar LRU con un hardware especial.

Consideremos primero el caso más sencillo. Este método requiere un **contador especial** de 64 bits, C, en *hardware*, que se incrementa de forma automática después de cada instrucción. Además, cada entrada de la tabla de páginas debe tener un campo de tamaño adecuado como para contener al contador. Después de cada referencia a la memoria, el valor actual de C se almacena en la entrada de la tabla de páginas correspondiente a la página a la que se hizo referencia. Al ocurrir un fallo de página, el sistema operativo examina todos los contadores de la tabla de páginas y elige el mínimo. Esa página es la utilizada menos recientemente.

Analicemos ahora un segundo algoritmo LRU en *hardware*. En una máquina con **n** marcos, el *hardware* LRU puede utilizar una matriz de **nxn** bits, cuyos datos iniciales son todos cero. En una referencia al marco **k**, el *hardware* primero activa todos los bits del renglón **k** y desactiva después todos los bits de la columna **k**. En cualquier instante, la fila cuyo valor en binario es mínimo es el marco utilizado menos recientemente, la fila con el siguiente valor más pequeño es el segundo marco utilizado menos recientemente, etc.

> El funcionamiento de este algoritmo aparece en la Figura 9 para cuatro marcos con referencias a los marcos en el orden 0 1 2 3 2 1 0 3 2 3. Después de hacer referencia al marco 0 tenemos la situación de la Figura 9 (a), etc.

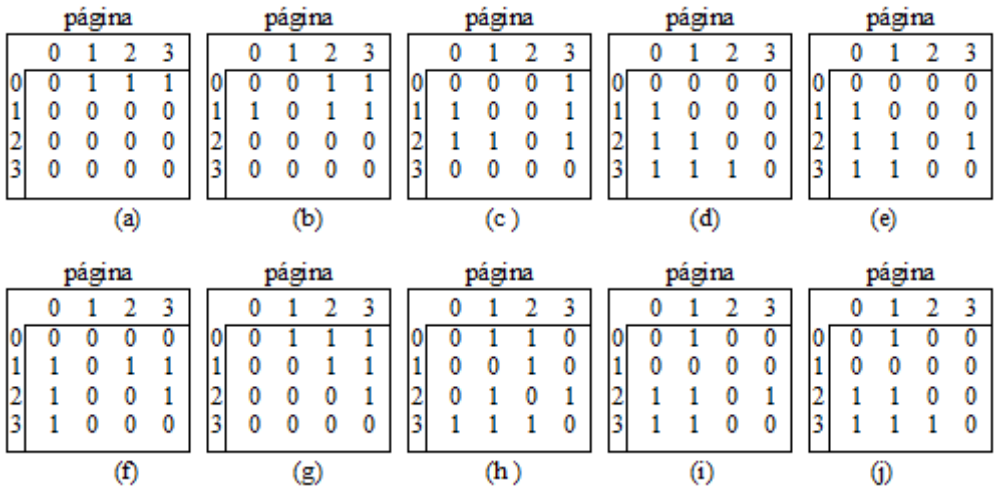


Figura 10. Reemplaz o LRU implementado con una matriz.

## 6.4 Aproximaciones a LRU

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Pocos sistemas proporcionan suficiente ayuda del *hardware* para un verdadero reemplazo de páginas LRU. Sin embargo, muchos sistemas ofrecen cierta ayuda, en la forma de un bit de referencia. El *hardware* coloca a uno el bit de referencia para una página cada vez que se hace una referencia a ella. Los bits de referencia están asociados a cada entrada de la tabla de páginas.

En principio, el sistema operativo pone a cero todos los bits. Al ejecutarse un proceso de usuario, el *hardware* activa (asignando 1) el bit asociado a cada página referenciada. Después de cierto tiempo, examinando los bits de referencia podemos **determinar qué páginas se han utilizado y cuáles no, aunque no sabemos el orden de uso**. Esta información parcial de la ordenación nos lleva a varios algoritmos de reemplazo de páginas que se aproximan al LRU.

### 6.4.1 Algoritmos de bits Adicionales de Referencia

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Podemos obtener **información adicional de la ordenación anotando los bits de referencia a intervalos regulares**.

> En una tabla en memoria podemos conservar 8 bits para cada página. A intervalos regulares (cada 100 milisegundos), una interrupción del cronómetro transfiere el control al sistema operativo. Éste desplaza el bit de referencia de cada página al bit de orden superior de sus 8 bits, desplazando los otros bits una posición a la derecha y descartando el bit de orden inferior. Estos registros contienen la **historia de la utilización de la página durante los últimos 8 periodos**.
>
> - Si el registro de desplazamiento contiene 00000000, entonces la página no ha sido utilizado en 8 periodos;
> - una página que se utiliza por lo menos una vez cada periodo tendría un valor 11111111 en su registro de desplazamiento.
>
> Una página con valor 11000100 ha sido utilizada más recientemente que una con valor 01110111. **Interpretando estos bits como enteros sin signo, la página con el menor número es la que se debe reemplazar**.

El número de bits históricos puede variar. En el caso extremo, el número puede reducirse a cero, dejando únicamente el bit de referencia. A esta versión se le llama **algoritmo de reemplazo de la segunda oportunidad**.

## 6.4.2 El algoritmo de la Segunda Oportunidad

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
Una modificación simple de FIFO, que evita deshacerse de una página de uso frecuente, inspecciona el bit R de la página más antigua. Si es 0, la página es antigua y no utilizada, por lo que se reemplaza de manera inmediata. Si el bit es 1, R se pone a cero, la página se coloca al final de la lista de páginas, como si hubiera llegado en ese momento a la memoria. Después continúa la búsqueda siguiendo la lista.

> En la Figura 11 (a), vemos las páginas de la A a la H, en una lista enlazada ordenada según el tiempo de llegada a la memoria.
>
> Supongamos que ocurre un fallo de página. La página más antigua es A, si A tiene el bit R a cero, se saca de la memoria. Por contra, si R vale 1, A se coloca al final de la lista, poniéndose a cero el bit R. La búsqueda de una página adecuada prosigue con B.
>
> Lo que hace el **algoritmo de la segunda oportunidad es buscar una página antigua sin referencias**, si todas las páginas tienen alguna referencia deviene en un simple FIFO. En efecto, supongamos que todas las páginas de la Figura 11 (a) tienen R a 1. Una a una, el sistema operativo traslada las páginas al final de la lista y pone su bit R a cero. Al final, se vuelve a la página A, la cual tiene R a cero. En este momento, A se retira de la memoria. Por lo tanto, el algoritmo siempre termina.

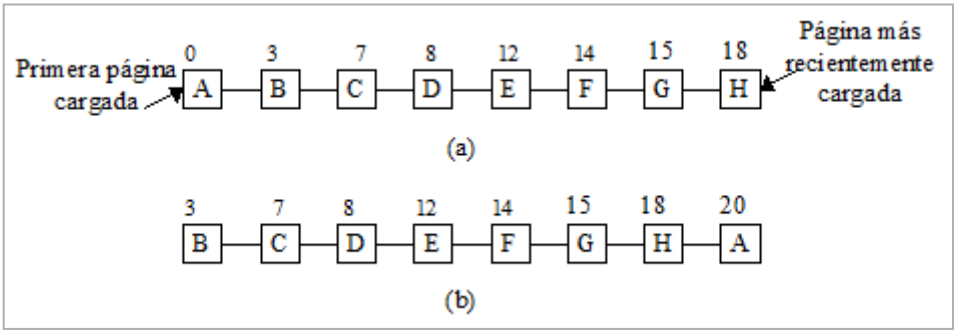



Figura 11. Segunda Oportunidad. (a) Páginas con orden FIFO. (b) Lista de páginas cuando ocurre un fallo de página en el instante 20 y A tiene el bit R a uno.

## 6.4.3 El Algoritmo del Reloj

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Aunque la segunda oportunidad es un algoritmo razonable, es ineficiente e innecesario, puesto que desplaza las páginas en una lista. Un mejor diseño consiste en mantener las páginas en una lista circular, con la forma de un reloj, como se muestra en la Figura 12. Una manecilla apunta hacia la página más antigua.

Al ocurrir un fallo de página, se inspecciona la página a la que apunta la manecilla. Si su bit R vale 0, la página se retira de la memoria, se inserta la nueva página en su lugar en el reloj, y la manecilla avanza una posición. Si R vale 1, este bit se pone a 0 y la manecilla avanza a la página siguiente. Este proceso continúa hasta encontrar una página con R a cero. Este algoritmo sólo difiere del anterior en su implementación.

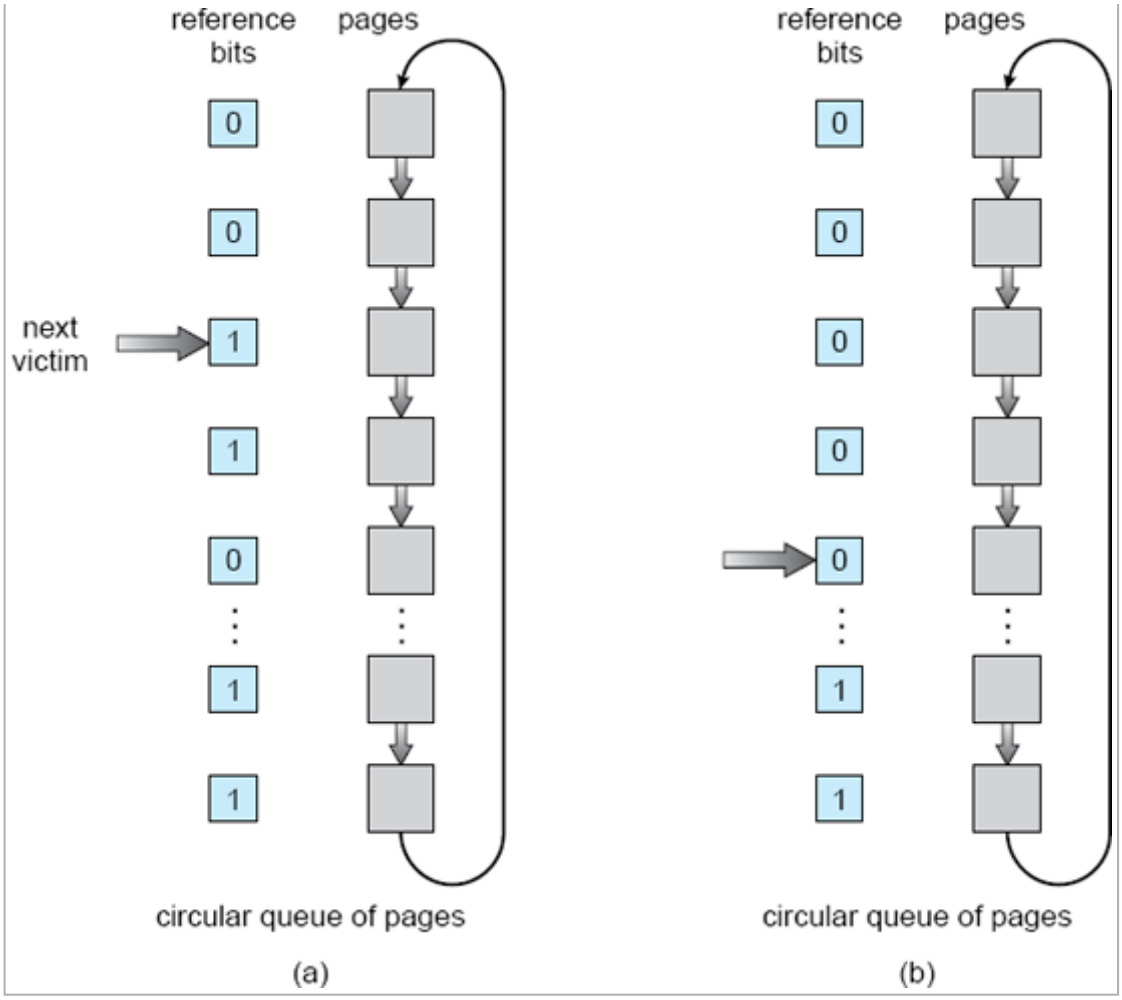



Figura 12. Algoritmo de reemplazo Segunda Oportunidad (Reloj) [Silberschatz, 2010].

## 6.5 Algoritmo LFU (Least Frequently Used)

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

El algoritmo de reemplazo de página **menos frecuentemente usada** (*Least Frequently Used*) mantiene un ***contador del número de referencias*** que se han hecho para cada página. Se reemplaza la página con el menor recuento. La razón para esta selección es que una página que se usa activamente debe tener un alto número de referencias.

Este algoritmo tiene problemas cuando una página se usa mucho en la fase inicial de un proceso, pero después ya no se utiliza. Como se usó bastantes veces, tiene un recuento alto y permanece en memoria aunque ya no se necesite. Una solución consiste en **desplazar los recuentos un bit a la derecha a intervalos regulares**, formando un recuento promedio de utilización que disminuye exponencialmente.

## 6.6 El algoritmo MFU (Most Frequently Used)

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
Otro algoritmo de reemplazo de páginas es el reemplazo más frecuentemente usado, que **se basa en el argumento de que la página con el menor recuento probablemente acaba de llegar** y aún tiene que usarse.

Como se podría esperar, no es común ni el reemplazo MFU ni el LFU porque son costosos y se alejan mucho del reemplazo OPT.

## 6.7 El algoritmo la de uso no reciente o NRU (Not Recently Used)

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

### El algoritmo de uso no reciente o NRU (Not Recently Used)

La mayoría de los ordenadores **presentan los bits R (de referencia) y M (de modificación)** en las **entradas de la tabla de páginas**, siendo estos bits actualizados vía *hardware*. Si el hardware no proporciona dichos bits, éstos pueden ser simulados mediante el software.

Cuando se inicia un proceso se señalan todas sus entradas en la tabla de páginas como si no estuvieran dentro de la memoria.

- Si se hace referencia a cualquier página, ocurre un fallo de página. El sistema operativo activa entonces el bit R (en sus propias tablas) y cambia la entrada de la tabla de páginas para que apunte hacia la página correcta, poniendo dicha entrada en modo sólo lectura (recordar los bits de permisos). El proceso retoma su ejecución;
- si se escribe en la página, ocurre otra interrupción por violación de permisos, lo que permite al sistema operativo activar el bit M en sus tablas, cambiando los permisos de la página a lectura y escritura.

Disponiendo de los citados bits R y M se puede construir el siguiente algoritmo: al iniciar un proceso, el sistema operativo asigna un valor 0 a los bits R y M de todas sus páginas. De manera periódica (cada interrupción del reloj) se pone a cero el bit R, para distinguir las páginas que no tienen referencias recientes de las que sí.

Ante un fallo de página, el sistema operativo inspecciona todas las páginas y las divide en cuatro categorías, según los valores actuales de los bits R y M:

- Clase 0:no referenciada, ni modificada (0,0).
- Clase 1:no referenciada, pero modificada (0,1).
- Clase 2:referenciada, pero no modificada (1,0).
- Clase 3:referenciada y modificada (1,1).

Aunque parece imposible la existencia de las páginas de la clase 1, este caso aparece cuando en una página de clase 3, una interrupción del reloj pone a cero su bit R. Las interrupciones del reloj no provocan el poner a cero el bit M, puesto que esta información es necesaria para saber si hay que volver a escribir la página en el disco o no.

El algoritmo NRU elimina una página de manera aleatoria de la primera clase no vacía con el número más pequeño. Una hipótesis implícita de este algoritmo es que es mejor eliminar una página modificada sin referencias en al menos un intervalo del reloj (por lo general, de 20 mseg) que una página sin modificar de uso frecuente. Este algoritmo es fácil de comprender, tiene una implementación eficiente y un rendimiento aceptable.

## 7 Asignación de Marcos

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
**¿Cómo se asigna la cantidad fija de memoria libre a los distintos procesos?**

> Si disponemos de 93 marcos libres y de dos procesos, ¿cuántos marcos obtiene cada proceso?

## 7.1 Número mínimo de Marcos

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

**No** podemos **asignar más del total de marcos libres**. También hay un **número mínimo de marcos** que pueden asignarse. Este número mínimo está definido por la arquitectura del conjunto de instrucciones. Recuerde que cuando ocurre un fallo de página antes de terminar la ejecución de una instrucción, ésta debe reiniciarse. Por tanto, debemos **contar con marcos suficientes para todas las páginas a las que pueda hacer referencia una instrucción**.

> Por ejemplo, considere una máquina donde todas las instrucciones de referencia a memoria permiten direccionamiento indirecto de un nivel (por ejemplo, en la página 16 hay una instrucción de carga que puede referirse a una dirección de la página 0, la cual es una referencia indirecta a la página 23), entonces la paginación necesita al menos tres marcos por proceso.

El **número mínimo de marcos está definido por la arquitectura del computador**, mientras que el **número máximo está definido por la cantidad de memoria física disponible**. Entre estos dos extremos podemos adoptar distintas opciones de asignación de marcos.

## 7.2 Algoritmos de Asignación de Marcos

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

La manera más fácil de dividir **m** marcos entre **n** procesos consiste en otorgar a cada uno una parte igual, **m/n** marcos. Una **asignación equitativa**.

> Por ejemplo, si hay 93 marcos y 5 procesos, cada proceso recibirá 18 marcos. Los tres marcos sobrantes pueden usarse como depósito de marcos libres.

Una alternativa es reconocer que los diversos procesos necesitarán cantidades distintas de memoria. Se debe hacer una **asignación proporcional**. Asignamos la memoria disponible a cada proceso de acuerdo con el tamaño de éste.

> Si un pequeño proceso de 10K y una base de datos interactiva de 127K son los únicos procesos que se ejecutan en un sistema con 62 marcos libres, no tiene mucho sentido asignar 31 marcos a cada proceso. El proceso pequeño no necesita más de 10 marcos, por lo que los 21 restantes se desperdician.

Sea $v_i$ el tamaño de la memoria virtual para el proceso $p_i$ y definamos $V = \sum_{i=1}^{n} v_i$. Entonces, si el total de marcos disponibles es **m**, asignamos $a_i$ marcos al proceso $p_i$, donde $a_i$ es aproximadamente $a_i = v_i/Vxm$. Por supuesto, debemos ajustar $a_i$ para que sean enteros, mayores al número mínimo de marcos requerido por el conjunto de instrucciones, y que sumados no excedan de m.

> Para la asignación proporcional dividiríamos los 62 marcos entre dos procesos, uno de 10 páginas y el otro de 127, asignando 4 y 57 marcos respectivamente, ya que 10/137 x 62 = 4, 127/137 x 62 = 57.

Por supuesto, **tanto en la asignación equitativa como proporcional, la asignación para cada proceso puede variar de acuerdo con el nivel de multiprogramación**. Si aumenta el nivel de multiprogramación, cada proceso perderá algunos marcos para proporcionar al nuevo proceso la memoria necesaria. Por el contrario, si disminuye, los marcos liberados se distribuyen entre los restantes.

El esquema de asignación proporcional se ha hecho dependiente del tamaño, pero nos podríamos basar en sus prioridades o en una combinación entre tamaño y prioridad. De igual modo, se podría permitir que un proceso de alta prioridad pueda reemplazar páginas de otro proceso de menor prioridad. Esto haría aumentar el número de marcos del proceso de alta prioridad en perjuicio de los de prioridad menor.

## 7.3 Reemplazo global frente a reemplazo local

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Cuando varios procesos compiten por marcos podemos clasificar los algoritmos de reemplazo de páginas en dos categorías: **reemplazo global** y **reemplazo local**.

El **reemplazo global** permite que el sistema operativo seleccione un marco para reemplazar de entre todo el conjunto de marcos, incluso si ese marco está asignado a otro proceso; un proceso puede quitar un marco a otro.

El **reemplazo local** requiere que el sistema operativo seleccione un marco de los asignados al proceso.

Con una **estrategia de reemplazo local, el número de marcos asignados a un proceso no cambia o podría aumentar si hay marcos libres**. Con el reemplazo global es posible que un proceso sólo seleccione marcos que pertenezcan a otros procesos, incrementando su número de marcos asignados (suponiendo que otros procesos no elijan sus marcos para reemplazo).

Un problema del **algoritmo de reemplazo global es que un proceso no puede controlar su propia tasa de fallos de página**. El conjunto de páginas en memoria para un proceso no sólo depende del comportamiento de paginación de ese proceso, sino también del de los otros. Por tanto, puede variar la forma en que se ejecuta un proceso debido a circunstancias totalmente ajenas.

Esto no ocurre con un **algoritmo de reemplazo local**. Con este esquema, **el conjunto de páginas en memoria para ese proceso sólo depende de su comportamiento** en cuanto a paginación. Ahora bien, el reemplazo local impide a un proceso tener acceso a otras páginas de memoria menos usadas. Por esta razón, el **reemplazo global generalmente brinda una mayor productividad**.

# 8 Hiperpaginación

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

Si el número de marcos asignados a un proceso de baja prioridad desciende por debajo del número mínimo requerido por la arquitectura del computador, debemos suspender la ejecución de ese proceso. Luego debemos descargar sus páginas restantes, liberando los marcos asignados.

Cualquier **proceso que no cuente con marcos suficientes provocará fallos de página muy frecuentemente**. Si se reemplazan páginas que a su vez están activas, estaremos sustituyendo una página que casi de inmediato se volverá a necesitar. Por tanto, pronto vuelve a generarse otro fallo de página, ocurriendo esto una y otra vez.

A una altísima actividad de paginación se le llama **hiperpaginación** (***thrashing***). Un sistema está en hiperpaginación si emplea más tiempo paginando que ejecutando.

## 8.1 Causas de la hiperpaginación

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

La hiperpaginación ocasiona severos problemas de rendimiento. Considere la siguiente situación, basada en el comportamiento real de los primeros sistemas de paginación.

El sistema operativo supervisa la utilización de la CPU. Si ésta es demasiado baja, aumentamos el nivel de multiprogramación introduciendo un nuevo proceso en el sistema. Se emplea un algoritmo de reemplazo

de páginas global, que reemplaza páginas sin importar a qué procesos pertenezcan. Suponga ahora que un proceso entra en una nueva fase de su ejecución y necesita más marcos. Comienza a generar fallos de página y a tomar páginas de otros procesos. Sin embargo, estos procesos necesitan esas páginas, por lo que también fallan, tomando páginas de otros procesos. Todos estos procesos generando fallos de página deben usar el dispositivo de paginación para intercambiar las páginas. A medida que se colocan en la cola del dispositivo de paginación, la cola de procesos listos se vacía. Mientras los procesos están bloqueados en espera del dispositivo de paginación, la utilización de la CPU decrece.

1. El planificador de la CPU ve disminuir la utilización de ésta y, por tanto, incrementa el nivel de multiprogramación.
2. El nuevo proceso trata de comenzar tomando páginas de los procesos en ejecución, lo que ocasiona más fallos de página y una cola más larga para el dispositivo de paginación.
3. Y se vuelven a repetir los pasos descritos. Ha comenzado la hiperpaginación y se desploma la productividad del sistema.
4. La tasa de fallos de página aumenta tremendamente y, como resultado, se incrementa el tiempo efectivo de acceso a memoria.
5. No se efectúa ningún trabajo porque el sistema operativo está continuamente paginando.

Los efectos de la hiperpaginación se pueden limitar utilizando un algoritmo de reemplazo local (o por prioridades). Con el reemplazo local, si un proceso comienza a hiperpaginar, no puede robar marcos de otro proceso y provocar que este también entre en hiperpaginación. Sin embargo, si los procesos están en hiperpaginación, la mayor parte del tiempo pueden encontrarse en la cola del dispositivo de paginación. El tiempo promedio de servicio de un fallo de página aumenta y, por tanto, el tiempo efectivo de acceso aumenta incluso para un proceso que no esté en hiperpaginación.

Para evitar la hiperpaginación debemos ofrecer a un proceso todos los marcos que necesita; pero, ¿cómo sabemos cuántos necesitan? Para saberlo se utilizan técnicas como el **modelo del conjunto de trabajo**, que se basa en el **concepto de localidad** de la ejecución de procesos.

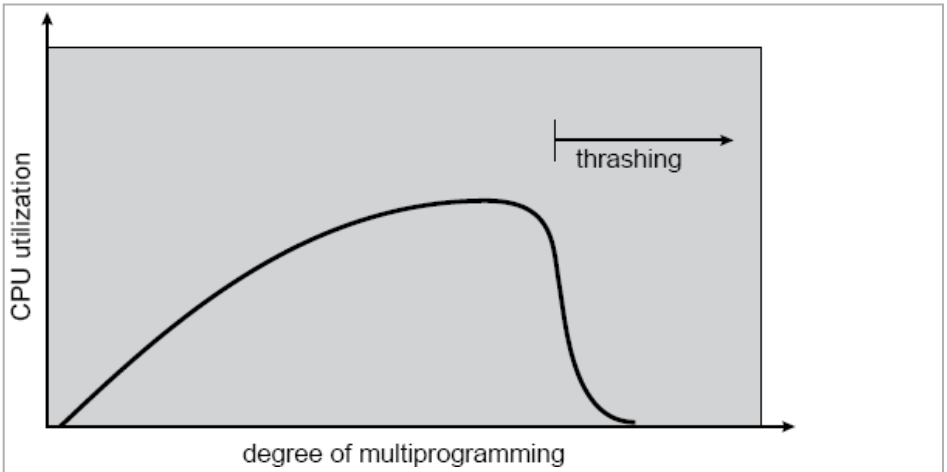


Figura 13. Curva de fallos de página fr ente al núm ero de marcos [Silberschatz, 2010].

## 8.2 Localidad

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

El **concepto de localidad** establece que un proceso, durante su ejecución, pasa de una localidad a otra. Una localidad es un conjunto de páginas que se utilizan conjuntamente. Un programa generalmente está compuesto por varias localidades distintas, las cuales pueden superponerse. Las localidades están definidas por la estructura del programa y sus estructuras de datos.

> Por ejemplo, al llamar a una subrutina se define una nueva localidad. En ésta se efectúan referencias a memoria de las instrucciones de la subrutina, sus variables locales y un subconjunto de las variables globales. Al salir de la subrutina, el proceso abandona esta localidad.

Suponga que a un proceso le asignamos suficientes marcos para albergar su localidad actual. Generará fallos de página hasta completar su localidad pero, luego, no habrá más fallos hasta que cambie de localidad. Si asignamos menos marcos que el tamaño de la localidad, el proceso entrará en hiperpaginación, ya que no puede mantener en memoria todas las páginas que está usando en ese momento.

## 8.3 Modelo del Conjunto de Trabajo

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**

El modelo del **conjunto de trabajo** (***Working Set Model***) se basa en el concepto de localidad. El modelo utiliza un parámetro, $\Delta$, para definir la ventana del conjunto activo. La idea es examinar las $\Delta$ referencias más recientes a páginas. El conjunto de páginas en las $\Delta$ referencias constituye el conjunto de trabajo. Si una página está en uso pertenecerá al conjunto de trabajo, y si ya no se usa se descartará tras $\Delta$ referencias después de ser referenciada por última vez. De esta manera, el conjunto de trabajo es una aproximación a la localidad del programa.

Por ejemplo, dada la serie de referencias a memoria de la Figura 14, si $\Delta = 10$ referencias de memoria, entonces el conjunto de trabajo en el instante t1 es {1, 2, 5, 6, 7}. En el instante t2„ habrá cambiado a {3, 4}.

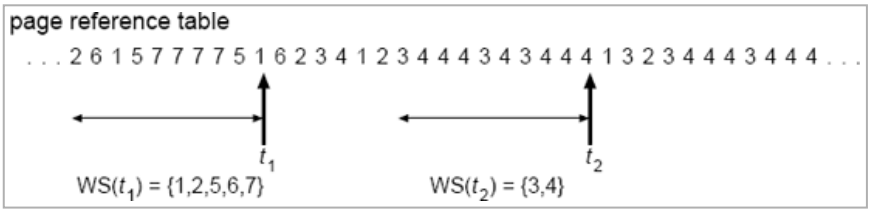



Figura 14. Modelo del Conjunto de Trabajo [Silberschatz, 2010].
La exactitud del conjunto de trabajo depende de la selección de $\Delta$. Si $\Delta$ es demasiado pequeño, no abarcará todo el conjunto del trabajo; si es demasiado grande, puede solapar varias localidades. En el caso extremo, si $\Delta$ es infinito, el conjunto de trabajo es todo el programa.
La propiedad más importante de los conjuntos del trabajo es su tamaño. Si calculamos el tamaño del conjunto de trabajo $WSS_i$, para cada proceso del sistema, podemos considerar $D = \sum_{i=1}^{n} WSS_i$, donde D es la demanda total de marcos. Cada proceso utiliza las páginas de su conjunto de trabajo, por lo que i necesita $WSS_i$ marcos. Si la demanda total es mayor que el número total de marcos disponibles (D > m) se producirá la hiperpaginación, ya que algunos procesos no tendrán marcos suficientes.

> El uso del **modelo del conjunto de trabajo** resulta sencillo. El sistema operativo supervisa el conjunto de trabajo de cada proceso y le asigna marcos suficientes para proporcionarle el tamaño del conjunto activo. Si hay suficientes marcos iniciales, se puede iniciar otro proceso. Si aumenta la suma de los tamaños de los conjuntos activos, excediendo el número total de marcos disponibles, el sistema operativo selecciona un proceso y lo suspende.
> Cuando existe un fallo de página y no hay página libres, se reemplaza una página que no pertenezca a ninguno de los conjuntos de trabajo de los procesos activos en memoria principal.

Sin embargo, tenemos el **problema del seguimiento del conjunto de trabajo**. La ventana del conjunto de trabajo es cambiante; cada vez que se hace referencia a la memoria, la nueva referencia aparece en un extremo y del otro extremo desaparece la referencia más antigua. Una página se encuentra en un conjunto activo si existe una referencia a ella en cualquier lugar de la ventana. Podemos aproximarnos al modelo del conjunto de trabajo mediante interrupciones a intervalos fijos de un cronómetro y un bit de referencia.
Por ejemplo, suponga que $\Delta$ es 10000 referencias y podemos producir una interrupción del cronómetro cada 5000 referencias. Cuando recibimos una interrupción del cronómetro, copiamos y borramos los valores de los bits de referencia para cada página, de manera que si ocurre un fallo de página podemos examinar el bit de referencia actual y los dos bits en memoria para determinar si la página ha sido utilizada en los últimos 10000 a 15000 referencias. Si se usó, por lo menos uno de éstos bits estará a uno; de no ser así, todos estarán a cero. Se considerará que todas las páginas que poseen por lo menos un bit a uno están en el conjunto de trabajo. Esta manera de actuar no es totalmente precisa, ya que no podemos decir cuándo ocurrió una referencia dentro de un intervalo de 5000.

Es posible reducir la incertidumbre aumentando el número de bits históricos y el número de interrupciones; sin embargo, el servicio de esas interrupciones será proporcionalmente más costoso.

## 8.4 Frecuencia de Fallos de Página

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
El **modelo del conjunto de trabajo** tiene bastante éxito, y el conocimiento del conjunto puede resultar útil para la **prepaginación**. Sin embargo, existen otros métodos de controlar la hiperpaginación, como la estrategia de **frecuencia de fallos de página**, que sigue un camino más directo.

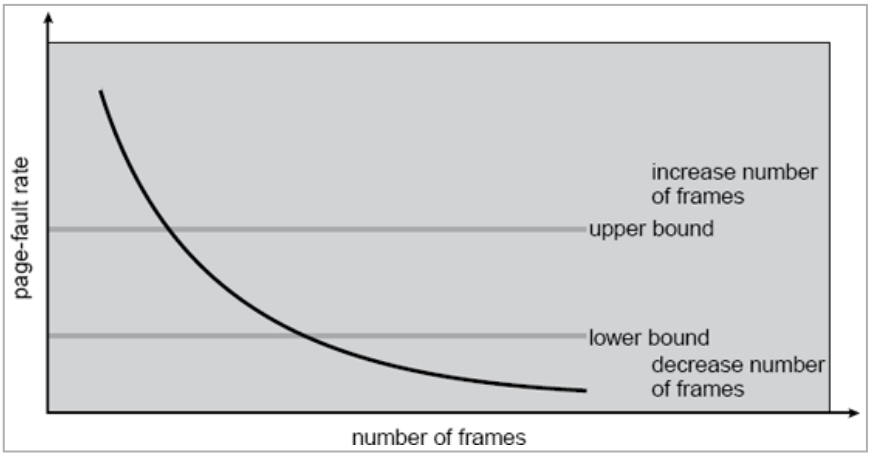



Figura 15. Frecuencia de fallos de página [Silberschatz, 2010].
Queremos evitar la hiperpaginación, o lo que es lo mismo, la elevada tasa de fallos de página. Cuando es demasiado alta sabemos que un proceso necesita más marcos, y si es demasiado baja, puede ser que el proceso tenga demasiados marcos. Podemos establecer límites superior e inferior para la tasa deseada de fallos de página. Si la tasa de fallos de página excede el límite superior, asignamos otro marco a ese proceso; si la tasa cae por debajo del límite inferior, quitamos un marco al proceso. Así podemos medir y controlar directamente la tasa de fallos de página para evitar la hiperpaginación.
Al igual que ocurre con la estrategia del conjunto de trabajo, posiblemente se tenga que suspender a un proceso. Si aumenta mucho la tasa de fallos y no hay marcos disponibles, debemos seleccionar un proceso y suspenderlo. Los marcos liberados se distribuyen entre los procesos con tasa elevada de fallos de página.

## 8.5 Otras Consideraciones

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
La **selección de un algoritmo de reemplazo** y una **política de asignación de marcos** son las principales decisiones que hay que tomar para un sistema de paginación, existen otros factores que deben considerarse.

### 8.5.1 Prepaginación

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
Una propiedad evidente de un sistema de paginación por demanda pura es el **gran número de fallos de página que ocurren al iniciar un proceso**. Esta situación surge al tratar de introducir en memoria la localidad inicial. También ocurrirá **cuando se reanuda un proceso intercambiado**.

La **prepaginación** es una estrategia que intenta evitar este alto nivel de paginación inicial. La estrategia consiste en traer a memoria al mismo tiempo todas las páginas que se necesitarán.
En un sistema que utiliza el **modelo del conjunto de trabajo**, por ejemplo, con cada proceso debemos conservar una lista de las páginas que forman en un instante el conjunto de trabajo. Por tanto, cuando un proceso se reanuda (porque a finalizado su E/S o se dispone de suficientes marcos libres), automáticamente **se incorpora todo el conjunto de trabajo antes de reanudar al proceso**.
La prepaginación puede ser una ventaja en algunos casos. La cuestión es sencillamente si **el costo de la prepaginación es menor al costo del servicio de los fallos de página correspondientes**. Puede ser que ya no se usen muchas de las páginas devueltas a memoria por la prepaginación. Suponga que las páginas están prepaginadas y una fracción $\alpha$ de estas **s** páginas realmente se usan ($0 \leq \alpha \leq 1$). La pregunta es si el costo de las $\alpha$ fallos de página que se evitan es mayor o menor al costo de la prepaginación de $(1 - \alpha)s$ páginas innecesarias. Si $\alpha$ es cercano a 0, pierde la prepaginación; si $\alpha$ es cercano a 1, la prepaginación gana.

## 8.5.2 Fijación de Páginas para E/S

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
Cuando se utiliza la paginación por demanda, en ocasiones tenemos que permitir que **algunas páginas se fijen en memoria**.
Debemos evitar que se produzca la siguiente secuencia de sucesos. Un proceso emite una solicitud de E/S y se coloca en la cola para ese dispositivo de E/S. Mientras tanto, se asigna la CPU a otros procesos. Estos generan fallos de página y, usando un algoritmo de reemplazo global, uno de ellos reemplaza la página que contiene el buffer de memoria del proceso en espera. Más tarde, cuando la solicitud de E/S avanza hasta el inicio de la cola de dispositivo, se efectúa la E/S para la dirección especificada. Sin embargo, este marco se está utilizando ahora para una página que pertenece a otro proceso.
Hay dos soluciones a este problema:

- No ejecutar nunca la E/S en memoria de usuario, sino copiar los datos entre la memoria del sistema y la de usuario. La E/S sólo se realiza entre la memoria del sistema y el dispositivo de E/S. Esta copia adicional puede generar un tiempo de procesamiento añadido inaceptable.
- Permitir que las páginas se fijen en memoria. A cada marco se asocia un bit de fijación; si se ha fijado el marco, no se puede seleccionar para reemplazo. Cuando termina la E/S, se desactiva la fijación de las páginas.

Otra aplicación del **bit de fijación se refiere al reemplazo normal de páginas**.
Considere la siguiente secuencia de sucesos. Un proceso de baja prioridad genera un fallo. Al seleccionar un marco para reemplazarlo, el sistema de paginación lee en memoria la página requerida. El proceso se encuentra listo para continuar su ejecución cuando reciba la CPU. Pero mientras un proceso de alta prioridad genera un fallo y el sistema de paginación encuentra una página que está en memoria pero que no ha sido ni modificada ni referenciada: la página que acaba de incorporar el proceso de baja prioridad. Esta página parece perfecta para el reemplazo; está limpia y no hay que reescribirla.
Podríamos impedir que un página recién incorporada sea reemplazada para no desperdiciar el costo que ha supuesto el llevar un página a memoria para luego no hacer uso de ella. Sin embargo, el uso del bit de fijación puede ser peligroso si se activa y nunca se desactiva. Se iría poco a poco inutilizando el espacio de memoria.

## 8.5.3 Estructura de los Programas

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
La paginación por demanda, está diseñada para ser transparente para el programa de usuario. En muchos casos, el usuario no se percata de que la memoria está paginada. Sin embargo, en muchos casos el rendimiento del sistema puede mejorar si se es consciente de la paginación subyacente.

> Un ejemplo en el que se pone de manifiesto esto sería el siguiente. Considere un programa en Pascal cuya función es poner a 0 cada elemento de una matriz de 128 por 128 (suponga que las páginas son de 128 palabras). El código sería:

**var A: array [1..128]of array [1..128] of integer;**
**for j:= 1 to 128 do**
**for i:=1 to 128 do**
**A[i,j] := 0;**

Supongamos que la matriz está almacenada por filas. Es decir, la matriz se almacena como A[1,1ฦ, A[1,2ฦ, ..., A[2,1ฦ, A[2,2ฦ, ...., A[128,128ฦ. Para páginas de 128 palabras, cada fila ocupa una página. De esta manera, el código anterior asigna ceros a una palabra de cada página, luego otra palabra en cada página, etc. Si el sistema operativo asigna 128 marcos al programa, entonces ocasiona 128 x 128 = 16384 fallos de página. Si cambiamos el código a
**var A: array [1..128] of array [1..128] of integer;**
**for i:= 1 to 128 do**
**for j:=1 to 128 do**
**A[i,j] := 0;**
se asignan ceros a todas las palabras de una página antes de comenzar con la siguiente, reduciendo a 128 el número de fallos de página.

Una cuidadosa **selección de las estructuras de datos** y de la programación puede **mejorar la localidad** y, por tanto, **disminuir la tasa de fallos de página** y el número de páginas para un programa. El **compilador y el cargador pueden tener un efecto considerable sobre la paginación**. Al separar el código y datos y generar código reentrante, las páginas de código pueden ser de sólo lectura y, por tanto, nunca se modificarán. El cargador puede evitar la colocación de rutinas entre páginas, o agrupar las rutinas que se llaman entre sí en una página.

# 9 Tamaño de página

2º Grado en Ingeniería en Informática
teoría de Sistemas Operativos
**Bloque Memoria: Memoria Virtual**
Al diseñar una máquina hay que tomar una decisión sobre el mejor tamaño de página. Como podrá suponer, no hay un tamaño único que sea el mejor, pero existen varios factores que apoyan tamaños distintos. Los tamaños son invariablemente potencias de dos, que suelen ir de 512 ($2^9$) a 16 MB. **¿Cómo seleccionamos el tamaño de página?**

## Tamaño de página versus tamaño de tabla de página

Un factor es el tamaño de la **tabla de páginas**. Un sistema con páginas pequeñas, usa más páginas y sus tablas de páginas utilizan más espacio de memoria. Como cada proceso activo debe tener su propia tabla de páginas, sería deseable un tamaño de página grande.
Para una memoria virtual de 4 MB habría 4096 páginas de 1K bytes, pero sólo 512 páginas de 8192 bytes.

## Tamaño de página versus fragmentación interna

Por otra parte, la memoria se utiliza mejor con páginas pequeñas. Una parte de la última página estará asignada pero no totalmente ocupada (fragmentación interna). Suponiendo que los tamaños de los procesos y de la página son independientes, podemos esperar que, en promedio, se desperdiciará la mitad de la última página de cada proceso. Para minimizar la fragmentación interna necesitamos un tamaño de página pequeño.
Esta pérdida representaría sólo 256 bytes en una página de 512 bytes, pero serían 4096 bytes en una página de 8192.

## Tamaño de página versus tiempo acceso a disco

Otro problema es el tiempo necesario para leer o escribir una página. El tiempo de E/S está compuesto por tiempo de búsqueda, latencia y transferencia. El tiempo de transferencia es proporcional a la cantidad transferida (o sea, al tamaño de la página), hecho que aparentemente apunta en favor de un tamaño de página pequeño. Sin embargo, **los tiempos de búsqueda y de latencia son mucho mayores al tiempo de transferencia**. Minimizar el tiempo de E/S implica elegir un tamaño de página grande.
A una velocidad de transferencia de 2MB por segundo, la transferencia de 512 bytes cuesta sólo 0.2 milisegundos. La latencia puede que sea de 8 milisegundos y el tiempo de búsqueda de 20 milisegundos. Por tanto, del tiempo total de E/S (28.2 milisegundos) sólo el 1% puede atribuirse a la transferencia. Si duplicamos el tamaño de la página, el tiempo aumenta a sólo 28.4 milisegundos. Esto significa que se