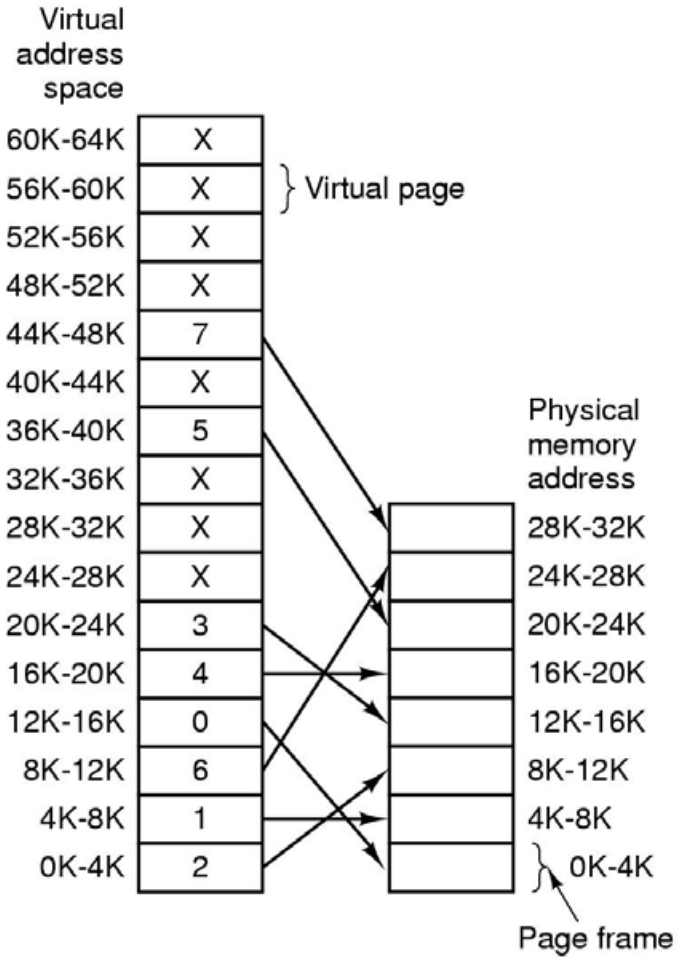
Paginación

Fallo de Página

El espacio de direcciones virtual se divide en unidades llamadas páginas. Las unidades correspondientes en la memoria física se denominan marcos de página. Las páginas y los marcos de página tienen siempre el mismo tamaño, que en este ejemplo es de 4 KB. Con un espacio de direcciones virtual de 64 KB, y con una memoria física 32 KB, tenemos 16 páginas virtuales y 8 marcos de página.



Cuando el programa intenta acceder a la dirección 0, por ejemplo, la dirección virtual 0 se envía a la MMU. La MMU ve que esa dirección virtual cae en la página 0 (0 a 4095), que de acuerdo a su correspondencia está en el marco de página 2 (8192 a 12287).

La MMU transforma todas las direcciones virtuales entre 0 y 4095 en las direcciones físicas entre 8192 y 12287.

¿Qué sucede si el programa intenta utilizar una página que no tiene correspondencia, como cuando el programa intenta acceder a la dirección 32780?

La MMU ve que la página no tiene correspondencia (lo que se indica con una cruz en la figura) y provoca una excepción que hace que la CPU ceda el control al sistema operativo. Esta excepción se denomina una falta de página. El sistema operativo escoge un marco de

página poco utilizado y escribe su contenido de vuelta al disco. A continuación el sistema operativo carga la página a la que se acaba de hacer referencia colocándola en el marco de página que acaba de quedar desocupado, modifica el mapa en la MMU y reinicia la instrucción interrumpida.

Por ejemplo, si el sistema operativo decidiera desalojar el marco de página 1, cargaría la página virtual 8 a partir de la dirección física 4K y haría dos cambios en el mapa de la MMU. Primero, marcaría la entrada de la página virtual 1 como sin correspondencia Luego sustituiría la cruz de la entrada correspondiente a la página virtual 8 por un 1.

Manejo de Fallos de Página

Aquí se describirá con detalle lo que ocurre en un fallo de página. La secuencia de eventos es la siguiente:

1. El hardware salta al *kernel*, guardando el contador de programa en la pila En la mayor parte de las máquinas, se guarda cierta información acerca del estado de la instrucción actual en registros especiales de la CPU.

2. Se inicia una rutina en código ensamblador para guardar los registros generales y demás información volátil, para evitar que el sistema operativo la destruya. Esta rutina llama al sistema operativo como un procedimiento.

3. El sistema operativo descubre que ha ocurrido un fallo de página y trata de descubrir cuál página virtual se necesita. A menudo, uno de los registros de hardware contiene esta información.

4. Una vez que se conoce la dirección virtual que produjo el fallo, el sistema comprueba si esta dirección es válida y si la protección es consistente con el acceso. De no ser así, el proceso recibe una señal o es eliminado. Si la dirección es válida y no ha ocurrido un fallo de página, el sistema comprueba si hay un marco de página disponible. Si no hay marcos disponibles, se ejecuta el algoritmo de reemplazo de páginas para seleccionar una víctima.

5. Si el marco de página seleccionado está sucio, la página se planifica para transferirla al disco y se realiza una conmutación de contexto, suspendiendo el proceso fallido y dejando que se ejecute otro hasta que se haya completado la transferencia al disco. En cualquier caso, el marco se marca como ocupado para evitar que se utilice para otro propósito.

6. Tan pronto como el marco de página esté limpio (ya sea de inmediato, o después de escribirlo en el disco), el sistema operativo busca la dirección de disco en donde se encuentra la página necesaria, y planifica una operación de disco para llevarla a memoria. Mientras se está cargando la página, el proceso fallido sigue suspendido y se ejecuta otro proceso de usuario, si hay uno disponible.

7. Cuando la interrupción de disco indica que la página ha llegado, las tablas de páginas se actualizan para reflejar su posición y el marco se marca como en estado normal.

8. La instrucción fallida se respalda al estado en que tenía cuando empezó, y el contador de programa se restablece para apuntar a esa instrucción.

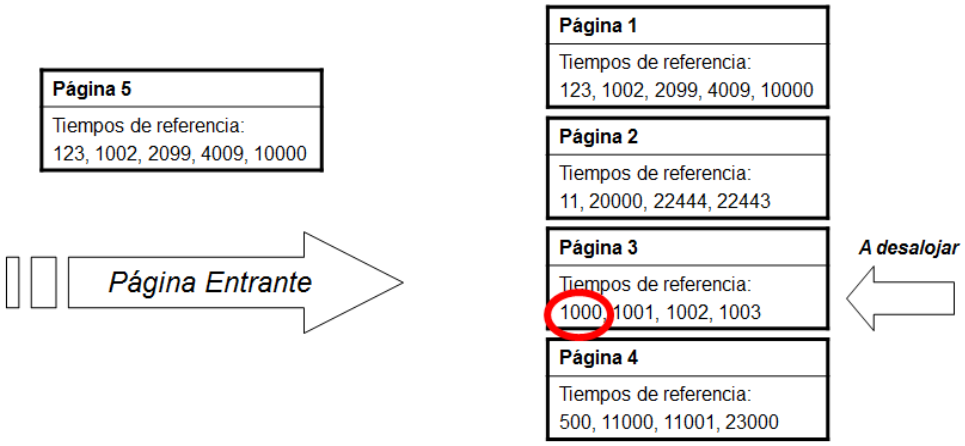
9. El proceso fallido se planifica y el sistema operativo regresa a la rutina (en lenguaje ensamblador) que lo llamó.

10. Esta rutina recarga los registros y demás información de estado, regresando al espacio de usuario para continuar la ejecución, como si no hubiera ocurrido el fallo.

Algoritmos de Reemplazo de Páginas

Algoritmo Óptimo

Un algoritmo óptimo debe generar el mínimo número de fallos de página. Por ello, la página que se debe reemplazar es aquella que tardará más tiempo en volverse a usar. Evidentemente, este algoritmo es irrealizable, ya que no se puede predecir cuáles serán las siguientes páginas accedidas. El interés de este algoritmo es que sirve para comparar el rendimiento de otros algoritmos realizables. Proceso de selección de la página a desalojar:



Ejemplo:

Cadena de Referencias	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
	F	F		F			F			F		

Algoritmo LRU (*Least Recently Used*)

Las páginas que no se han utilizado durante siglos probablemente seguirán sin utilizarse durante mucho tiempo. Esta idea sugiere un algoritmo realizable: cuando se produzca una falta de página, sustituir la página que lleve más tiempo sin utilizarse

Aunque LRU es teóricamente realizable, tiene un coste elevado. Para implementarlo fielmente es necesario mantener una lista enlazada de todas las páginas que están en la memoria, con la página que se utilizó más recientemente al principio y la página menos recientemente utilizada al final. La dificultad consiste en que la lista debe actualizarse en cada referencia a la memoria. Encontrar una página en la lista, sacarla de la lista y reinsertarla al frente es una operación muy lenta, incluso realizándose por hardware (suponiendo que pudiera construirse tal hardware).

Ejemplo:

Cadena de Referencias	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	3	2	1	5	2	4	5	3	2	5	2
		2	3	2	1	5	2	4	5	3	2	5
				3	2	1	5	2	4	5	3	3
	F	F		F	F		F		F	F		

Cada columna representa la lista enlazada de todas las páginas que están en la memoria actualmente. Siguiendo la mecánica de LRU, con cada referencia a una página esta es movida al principio de la lista y cada vez que ocurra un fallo de página se desaloja la última página.

Algoritmo FIFO (*First-In, First-Out*)

El sistema operativo mantiene una lista de todas las páginas que están actualmente en la memoria, con la más antigua al principio de la lista y la más nueva al final. Al producirse una falta de página, se sustituye la página que está al principio de la lista y la nueva se añade al final. FIFO en algunos casos podría sustituir una página que no tiene uso frecuente, pero también es posible que sustituya páginas que son muy usadas a lo largo de la vida de un programa, por lo tanto, estas páginas se cargarán y expulsarán repetidas veces. Por esta razón, raramente se utiliza FIFO en su forma pura. Ejemplo:

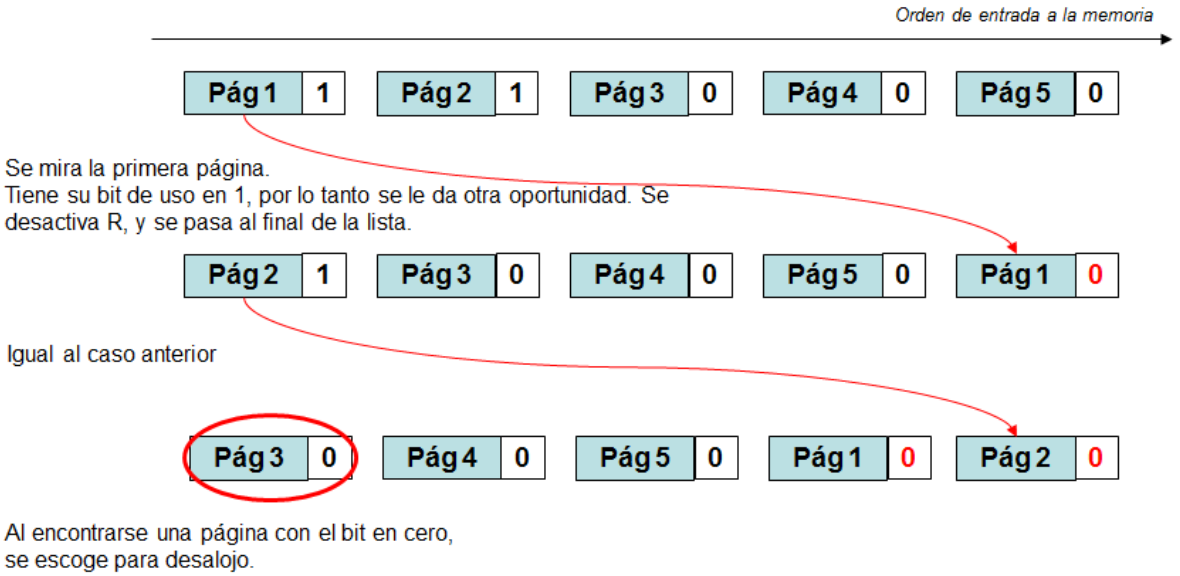
Cadena de Referencias	2	3	2	1	5	2	4	5	3	2	5	2
FIFO	2	3	3	1	5	2	4	4	3	3	5	2
		2	2	3	1	5	2	2	4	4	3	5
				2	3	1	5	5	2	2	4	3
	F	F		F	F	F	F		F		F	F

Cada columna representa la lista de todas las páginas que están actualmente en la memoria. A diferencia de LRU, FIFO sólo modifica la lista cuando ocurre un fallo de página.

Algoritmo de la Segunda Oportunidad

Una modificación sencilla del algoritmo FIFO que evita el problema de sustituir una página que se usa mucho consiste en examinar el bit *R* de la página más antigua. Si es 0, quiere decir que la página es antigua y además no se usa. Por lo tanto, se la sustituye de inmediato. Si el bit *R* es 1, se pone a 0, se coloca la página al final de lista de páginas y su instante de carga se actualiza como si acabara de cargarse en la memoria. Luego se continúa la búsqueda.

Nota: Para hacer posible que el sistema operativo pueda recoger estadísticas útiles sobre qué páginas se están usando y cuáles no, casi todos los ordenadores con memoria virtual asocian a cada página dos bits de estado *R* (*Referenced* o Referenciada) y *M* (*Modified* o Modificada). *R* se activa cada vez que se referencia la página (para leer o escribir) y *M* se activa cada vez que se escribe en la página (es decir, se modifica). Una vez activado un bit, conserva el valor 1 hasta que el sistema operativo lo desactiva a 0 por software.

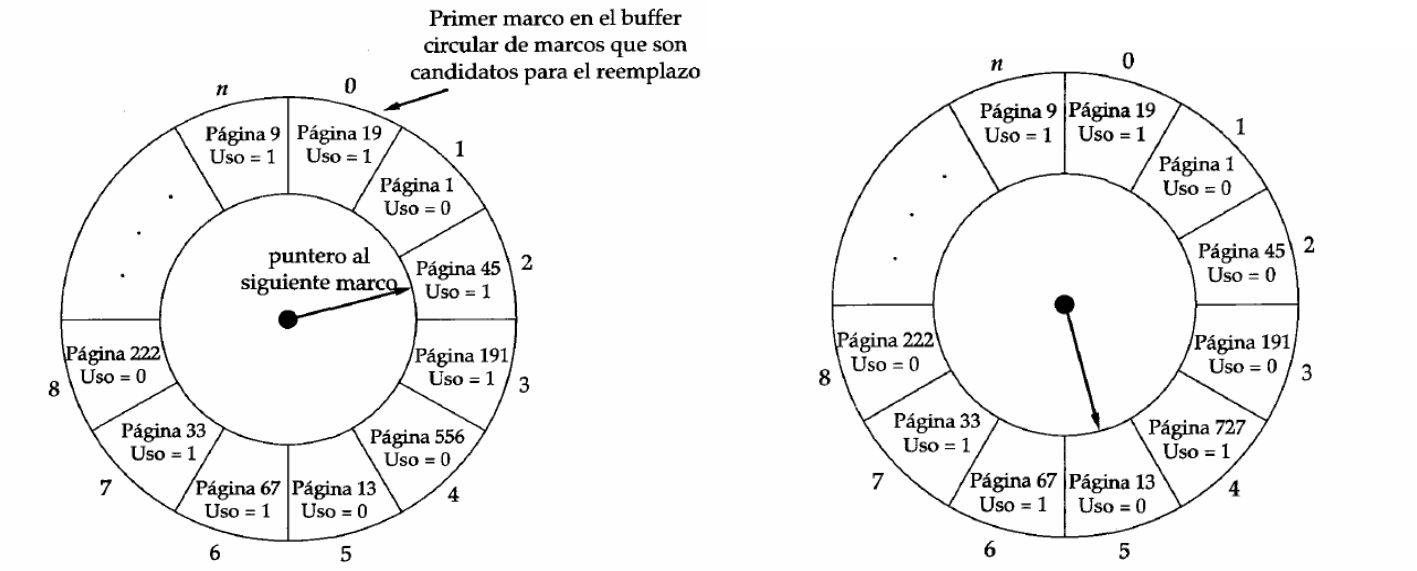


En el caso de que todas las páginas hayan tenido el bit de uso en cero, este algoritmo degrada en FIFO.

Algoritmo del Reloj (*Clock*)

Aunque el algoritmo de la segunda oportunidad es un algoritmo razonable, es innecesariamente ineficiente porque continuamente cambia las páginas de lugar dentro de la lista. Una estrategia mejor sería mantener todos los marcos de página sobre una lista circular en la forma de un reloj. Una manecilla apunta a la página más antigua.

Cuando se presenta una falta de página, se examina la página a la que apunta la manecilla. Si su bit *R* es 0, dicha página se sustituye, insertando la nueva en su lugar y adelantando una posición la manecilla. Si *R* es 1, se cambia a 0 y la manecilla se adelanta a la siguiente página. Este proceso se repite hasta hallar una página con *R* = 0. No es sorprendente que a este algoritmo se le llame el algoritmo del reloj. La única diferencia respecto al de la segunda oportunidad radica en su implementación.



(a) Estado del buffer justo antes del reemplazo de página

(b) Estado del buffer justo después del siguiente reemplazo de página

En este ejemplo cuando una página es traída de disco se activa el bit R, sin embargo es posible encontrar implementaciones donde el bit R no se activa inmediatamente sino hasta la siguiente referencia a esa página.

Algoritmo NRU (*Not Recently Used*)

Cuando se inicia un proceso, el sistema operativo pone a cero los bits R y M para todas sus páginas. De forma periódica (por ejemplo en cada interrupción de reloj), el bit *R* se pone a 0 para distinguir las páginas que no se han referenciado últimamente de aquellas que sí lo han sido.

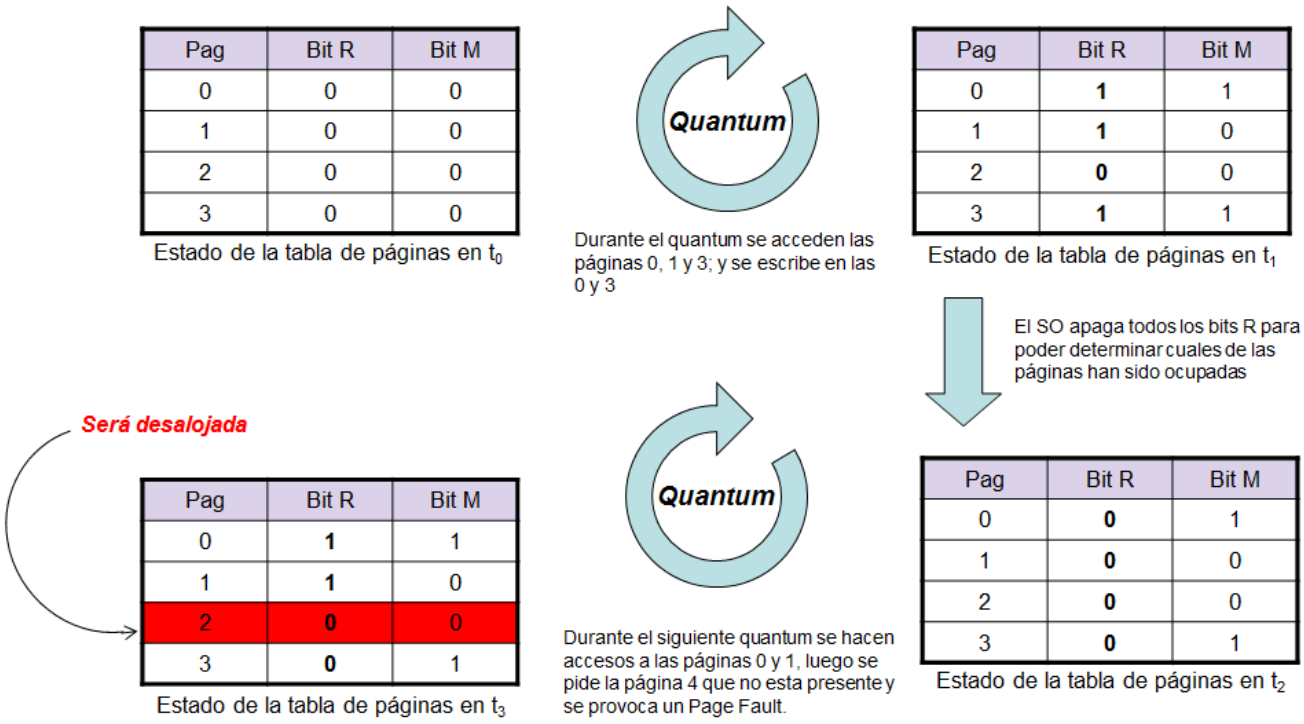
Cuando se presenta una falta de página, el sistema operativo inspecciona todas las páginas dividiéndolas en cuatro categorías según los valores actuales de sus bits *R* y *M*:

Clase 0: página ni referenciada, ni modificada.
Clase 1: página no referenciada y modificada.
Clase 2: página referenciada y no modificada.
Clase 3: página referenciada y modificada.

La clase 1 ocurre cuando una interrupción de reloj desactiva el bit *R* de una página de la clase 3. Las interrupciones de reloj no desactivan el bit *M* porque esa información es necesaria para saber si la página debe actualizarse en el disco o no.

El algoritmo NRU sustituye al azar una página de la clase de número más bajo que no esté vacía. Este algoritmo se basa en la suposición implícita de que es mejor sustituir una página modificada a la que no se ha hecho referencia en por lo menos un tic de reloj, en vez de una página limpia que se está usando mucho.

El principal atractivo de NRU es que es fácil de entender, tiene una implementación moderadamente eficiente y proporciona un rendimiento que, aunque ciertamente no es óptimo, puede ser aceptable.



Algoritmo de Envejecimiento

El algoritmo de la página no frecuentemente utilizada (NFU; *Not Frequently Used*) es una posible forma de implementar LRU por software. NFU requiere por cada página un contador software asociado con valor inicial cero. En cada interrupción de reloj, el sistema operativo explora todas las páginas que están en la memoria.

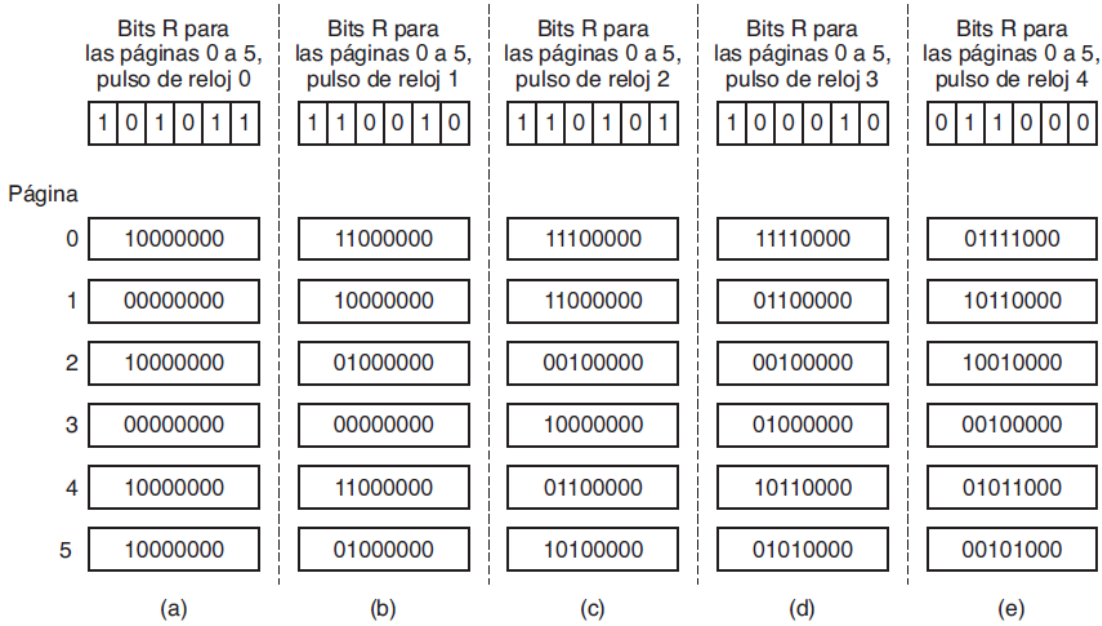
Para cada página, el bit *R*, que es 0 o 1, se suma al contador.

El problema principal con NFU es que nunca olvida nada. Por ejemplo, en un compilador de varias pasadas, las páginas que se utilizaron mucho durante la pasada 1 podrían seguir teniendo un valor alto del contador en las pasadas posteriores. De hecho, si la pasada 1 es la de mayor tiempo de ejecución de todas, las páginas que contienen el código de las pasadas subsiguientes podrían tener siempre contadores más bajos que las

páginas de la pasada 1. Eso haría que el sistema operativo sustituyese páginas útiles, en vez de páginas que ya no se usan.

Aging desciende del algoritmo "No usada frecuentemente", con algunas modificaciones necesarias para tener en cuenta en qué momento fue usada frecuentemente una página, y no solamente cuántas veces fue.

En vez de sólo incrementar el contador de la página cuando es referenciada, primero se desplaza a la derecha (se divide entre 2) y después el bit R se añade al bit más izquierdo.



Cuando sucede una falta de página, la página cuyo contador es el más bajo se reemplaza (página que menos ha sido referenciada en los últimos ticks de reloj).

Algoritmo del Conjunto de Trabajo

Los procesos al ejecutarse no hacen referencia a todas sus páginas constantemente. Sucede que en cualquier fase de la ejecución el proceso accede a un subconjunto pequeño de la totalidad de sus páginas

Se denomina Conjunto de Trabajo al conjunto de páginas que está utilizando en un momento dado, este cambia periódicamente a medida que el proceso se ejecuta.

Si no existe memoria disponible en el sistema para contener el conjunto de trabajo de un proceso, este generará muchos fallos de páginas en un número reducido de instrucciones. A este fenómeno se le denomina Thrasing o Hiperpaginación.

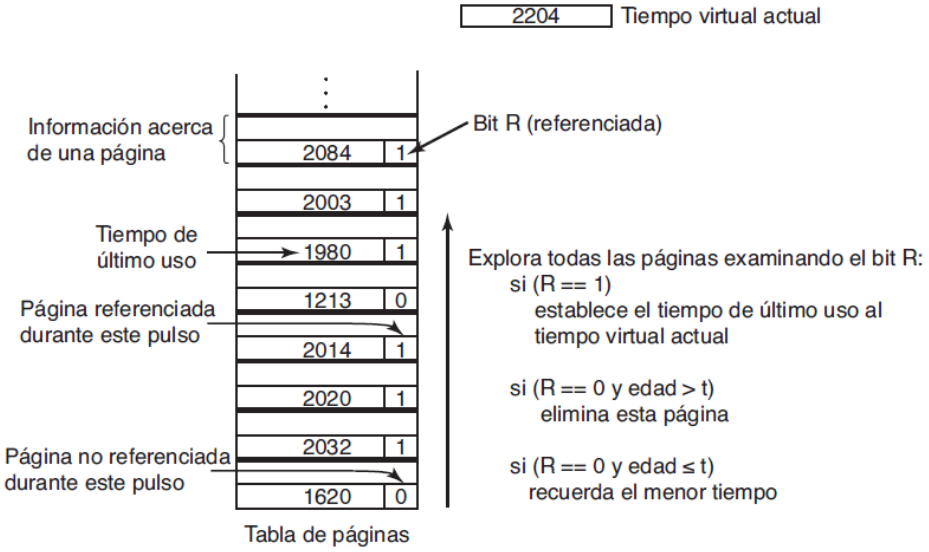
Desde luego, tener una definición operativa del conjunto de trabajo no implica que haya una forma eficiente de calcularlo en tiempo real, durante la ejecución del programa. Imaginemos un registro de desplazamiento de longitud *k*. Cada referencia a la memoria lo desplaza una posición a la izquierda e inserta a la derecha el número de la página a la que

se hizo referencia más recientemente. El conjunto correspondiente a los *k* números de página contenidos en el registro de desplazamiento sería el conjunto de trabajo. En teoría, al producirse una falta de página, el contenido del registro de desplazamiento podría leerse y ordenarse.

Después de eliminar las páginas repetidas, el resultado sería el conjunto de trabajo. Sin embargo, mantener el registro de desplazamiento y procesarlo en cada falta de página tendría un coste prohibitivo, por lo que esta técnica nunca se usa.

En vez de eso, se emplean diversas aproximaciones. Una muy común consiste en desechar la idea de contar hacia atrás *k* referencias a la memoria y utilizar en su lugar el tiempo de ejecución. Por ejemplo, en lugar de definir el conjunto de trabajo como las páginas que se usaron durante los 10 millones de referencias a la memoria anteriores, podemos definirlo como el conjunto de páginas utilizadas durante los últimos 100 milisegundos de tiempo de ejecución. Hay que señalar que cada proceso cuenta con su propio tiempo de ejecución. Por tanto, si un proceso comienza a ejecutarse en el instante *T* y ha tenido 40 milisegundos de tiempo de CPU en el instante real *T* + 100 milisegundos, para el cálculo del conjunto de trabajo su tiempo será de 40 milisegundos. El tiempo de CPU que ha consumido en realidad un proceso desde que se inició se denomina a menudo su tiempo virtual actual. Con esta aproximación, el conjunto de trabajo de un proceso es el conjunto de páginas a las que ha hecho referencia durante los últimos τ segundos de tiempo virtual. El algoritmo funciona como sigue. Se supone que el hardware se encarga de activar los bits R y M, como mencionamos anteriormente. También se supone que una interrupción de reloj periódica ejecuta el código necesario para desactivar el bit de Referenciada en cada tic de reloj.

En cada falta de página, la tabla de páginas se explora en busca de una página apropiada para sustituir.



Conforme se procesa cada entrada, se examina el bit R. Si es 1, se escribe el tiempo virtual actual en el campo Tiempo del último uso en la tabla de páginas, para indicar que la página se había utilizado cuando se produjo la falta de página. Puesto que se hizo referencia a la página durante el tic de reloj actual, es evidente que pertenece al conjunto de trabajo y no es candidata para su sustitución (se supone que τ abarca varios tics de reloj).

Si R es 0, quiere decir que no se ha hecho referencia a la página durante el tic de reloj actual y podría ser candidata para su sustitución. Para ver si se la debe desalojar o no, se calcula su edad, es decir, el tiempo virtual actual menos su Tiempo del último uso y se compara con τ.

Si la edad es mayor que τ quiere decir que la página ya no está en el conjunto de trabajo, así que se sustituye, cargándose la nueva página en el marco que ella estaba ocupando. No obstante, la exploración termina de actualizar las entradas restantes de la tabla de páginas.

Por otra parte, si R es 0 pero la edad es menor o igual que τ quiere decir que la página todavía pertenece al conjunto de trabajo. Se le perdona la vida por el momento, pero se toma nota de qué página tiene mayor edad (valor más pequeño de Tiempo del último uso). Si se explora toda la tabla de páginas sin encontrar una candidata para ser sustituida, significa que todas las páginas están en el conjunto de trabajo. En tal caso, si se encontró una o más páginas con R = 0, se sustituye la de mayor edad. En el peor caso, todas las páginas se habrán referenciado durante el tic de reloj actual (y, por lo tanto, todas tienen R = 1), así que se escoge una al azar para sustituirla, preferiblemente una que esté limpia.

El Algoritmo de Sustitución de Páginas WSClock

Es un algoritmo mejorado que se basa en el algoritmo del reloj pero que usa también la información del conjunto de trabajo. Se utiliza mucho en la práctica por su sencillez de implementación y buen rendimiento.

La estructura de datos en que se apoya WSClock es una lista circular de marcos de página. Cuando se carga la primera página, ésta se añade a la lista. A medida que se accede a nuevas páginas, éstas se incorporan a la lista formando un anillo. Cada entrada contiene el campo Tiempo del último uso del algoritmo del conjunto de trabajo básico, junto con el bit R y el bit *M*.

Al igual que en el algoritmo del reloj, cada vez que se produce una falta de página se examina primero la página a la que apunta la manecilla. Si el bit *R* es 1, quiere decir que la página se referenció durante el tic actual, así que no es una candidata ideal para sustituir. Por lo tanto, bit *R* se pone a 0, se adelanta la manecilla a la siguiente página y se repite el algoritmo con ella.

Consideremos ahora qué sucede si la página a la que apunta la manecilla tiene R = 0. Si su edad es mayor que τ y la página está limpia, quiere decir que no está en el conjunto de trabajo y que ya hay una copia válida en el disco. La nueva página simplemente se coloca en ese marco de página. Por otra parte, si la página está sucia, no podrá sustituirse de inmediato porque no hay una copia válida en el disco. Para evitar un cambio de proceso, se planificará su escritura en el disco, pero la manecilla se adelanta y el algoritmo continúa con la siguiente página. Pues podría haber una página limpia vieja más adelante que se podría utilizar inmediatamente.

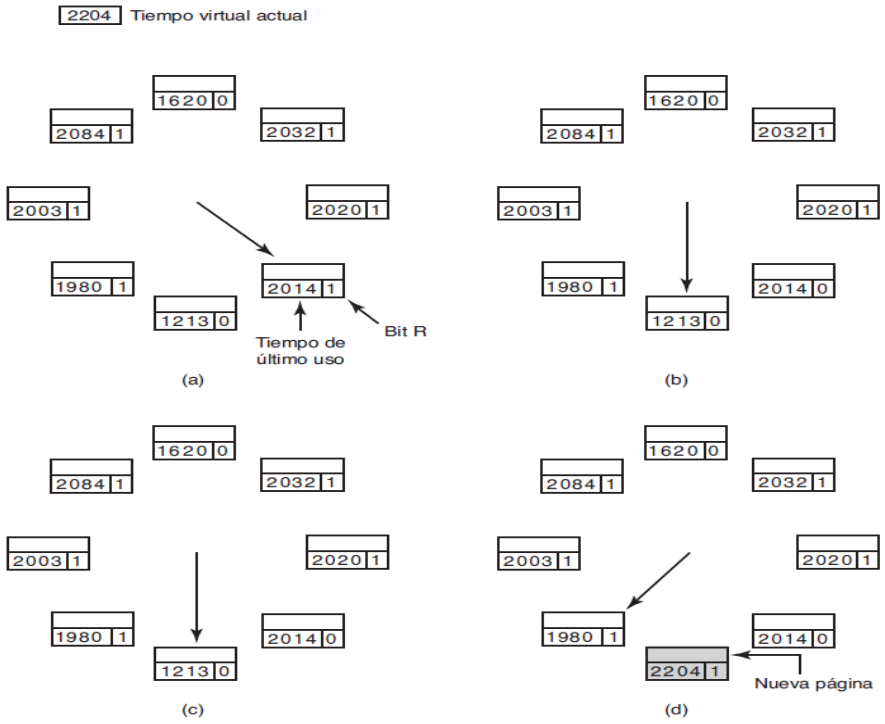
En principio, todas las páginas podrían estar planificadas para E/S de disco en una vuelta completa al reloj. Para reducir el tráfico de disco, podría fijarse un límite, y sólo permitir que se planifiquen para su escritura en el disco n páginas como máximo.

¿Qué sucede si la manecilla da toda la vuelta y regresa a su punto de partida? Debemos distinguir dos casos:

1. Se planificó al menos una escritura.

2. No se planificó ninguna escritura.

En el primer caso, la manecilla tan solo se sigue adelantando, en busca de una página limpia.



Operación del algoritmo WSClock. **(a)** y **(b)** dan un ejemplo de lo que ocurre cuando R = 1. **(c)** y **(d)** dan un ejemplo de cuando R = 0

Puesto que se han planificado una o más escrituras, tarde o temprano terminará alguna, marcándose su página como limpia. La primera página limpia que se encuentre

será la que se sustituya. Esta página no es necesariamente la correspondiente a la primera escritura planificada porque el controlador del disco podría reordenar las escrituras para optimizar el rendimiento del disco.

En el segundo caso, todas las páginas están en el conjunto de trabajo, pues de lo contrario se habría planificado por lo menos una escritura. A falta de información adicional, lo más sencillo es desalojar cualquier página limpia y usar su marco. Podría haberse tomado nota de la ubicación de una página limpia durante el movimiento de la manecilla. Si no hay páginas limpias, se escoge como víctima la página actual, se escribe en el disco y se desaloja.

Modelación de Algoritmos de Paginación

Anomalía de Belady

Intuitivamente, podríamos pensar que cuantos más marcos de página tenga la memoria, menos faltas de página experimentará un programa. Aunque parezca bastante sorprendente, no siempre sucede así. Belady y otros (1969) descubrieron un contraejemplo en el que FIFO provocaba más faltas de página con cuatro marcos de página que con tres. Esta situación se conoce como la **anomalía de Belady**, y se ilustra en la Figura 2.1, para un programa con cinco páginas virtuales, numeradas del 0 al 4. Las páginas se referencian en el orden:

0 1 2 3 0 1 4 0 1 2 3 4

En la Figura 2.1(a) vemos como con tres marcos de página se generan un total de nueve faltas de página. En la Figura 2.1 (b) obtenemos diez faltas de página con cuatro marcos de página.

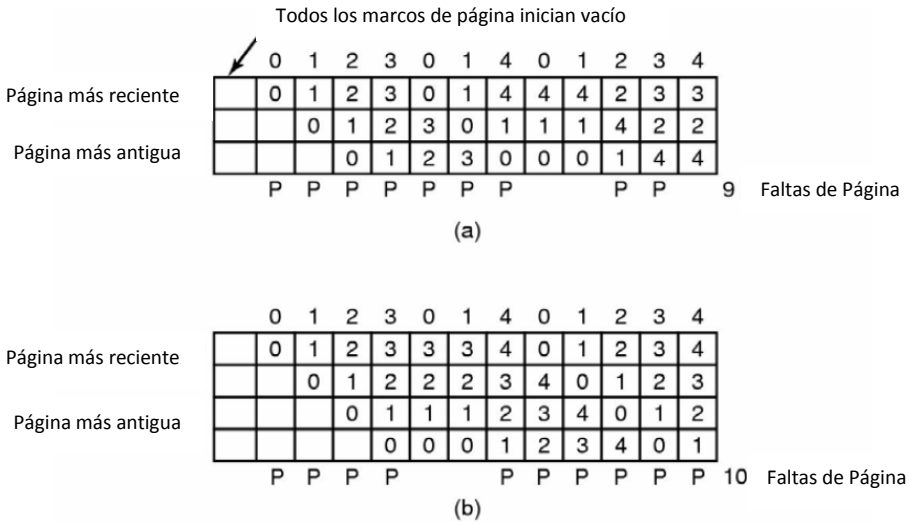


Figura 2.1 (a) FIFO con tres marcos de página. **Figura 2.1 (b)** FIFO con cuatro marcos de página. Las *P*s indican las referencias que provocan faltas de página.

Algoritmos de Pila

Los algoritmos de pila presentan la propiedad de que, dada una secuencia de referencias, una memoria con más marcos de página, tras un número determinado de referencias, mantiene el conjunto de páginas que contendría con un tamaño menor en la misma referencia. En otras palabras, la tasa de fallos nunca aumenta si se incrementa el tamaño de memoria. Los algoritmos de pila evitan la anomalía de Belady.

Para dejar más claro su funcionamiento, examinaremos un ejemplo utilizando el algoritmo de sustitución de páginas LRU. En este ejemplo supondremos dos sistemas con un espacio de direcciones virtual de 6 páginas y que la memoria física tiene 3 y 4 marcos de página, respectivamente. En la parte superior de la Figura tenemos una serie de referencias que consiste en las 9 páginas:

0 2 1 3 5 4 3 1 4

Debajo de la serie de referencias tenemos 10 columnas de 6 elementos cada una. La primera columna, que está vacía, refleja el estado de *M (M* es un array interno que sigue la pista del estado de la memoria), antes de iniciarse la ejecución. Cada columna sucesiva muestra *M* después de procesarse la siguiente referencia de la serie mediante la aplicación del algoritmo de sustitución de páginas utilizado por el sistema. La zona sombreada en azul corresponde a la parte superior de *M*, es decir, las primeras tres y cuatro entradas, que corresponden a marcos de página en la memoria. Las páginas que figuran dentro del rectángulo grueso están en la memoria, y las que están debajo se han intercambiado al disco.

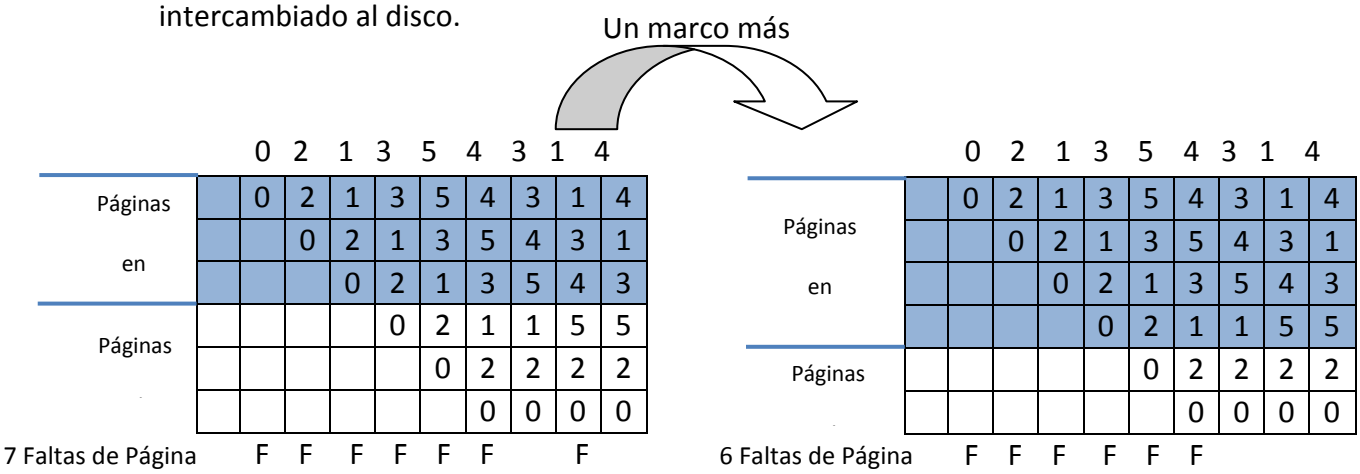


Figura 2.2 Estado del array *M*, después de procesarse cada elemento de la serie de referencias. Podemos apreciar que no ocurre la anomalía de Belady.

Cada vez que se hace referencia a una página que no está dentro del rectángulo con borde grueso, tiene lugar una falta de página, como se indica con las *F*s debajo de la matriz.

Algunas propiedades de este modelo son:

1. Cuando se hace referencia a una página, siempre se la coloca en lo alto de *M*.

2. Si la página solicitada ya estaba en *M*, todas las páginas que estaban por encima de ella bajan una posición. Una transición desde el interior del rectángulo grueso hacia fuera de él corresponde al desalojo de una página de la memoria.

3. Las páginas que estaban debajo de la página referenciada no se mueven. De esta manera, los contenidos sucesivos de *M* representan exactamente el comportamiento del algoritmo LRU.

La Cadena de Distancias

En el caso de los algoritmos de pila, la referencia a una página puede abstraerse como la distancia desde la parte superior de la pila donde se colocó la página.

La cadena de distancias lleva el conteo de la cantidad de veces que se ha accedido a páginas en cada una de las posibles distancias, considerando que:

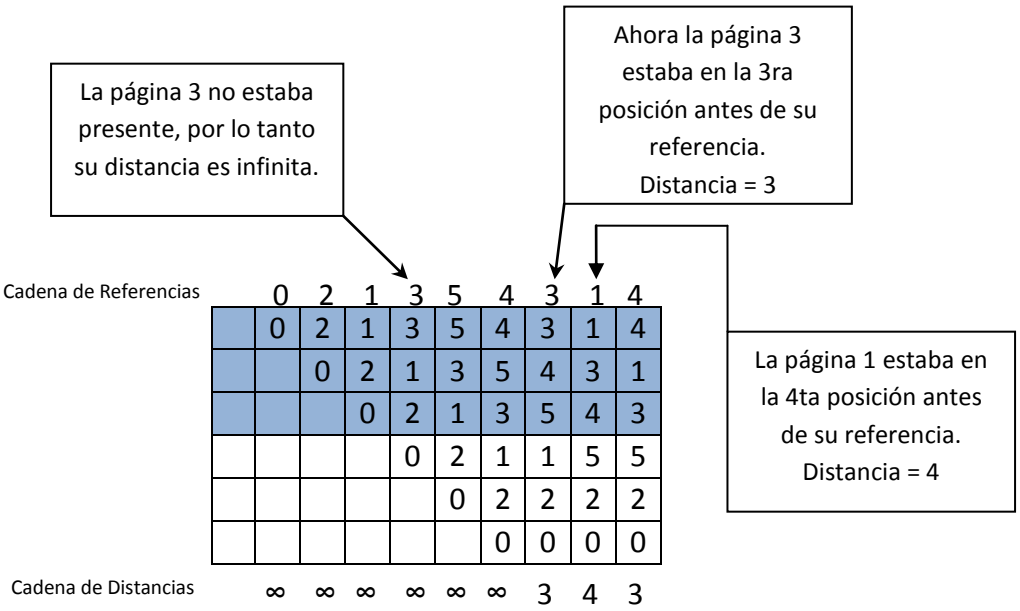


Figura 2.3 Estado del array *M* con la distancia de la serie o cadena de referencias.

Predicción de la Tasa de Faltas de Página

La serie de distancias puede servir para predecir el número de faltas de página que se producirán con memorias de diferentes tamaños. Este cálculo puede realizarse tomando en cuenta la Figura 2.3.

El algoritmo comienza explorando la serie de distancias, página por página, y llevando la cuenta de las veces que aparece la distancia 1, las veces que aparece la distancia 2, etc. C_i es el número de veces que aparece la distancia *i.* En la Figura 2.4 se ilustra el vector *C* para las distancias en la serie de referencias de la figura 2.3. En este ejemplo, sucede 4 veces que la página referenciada ya estaba en lo alto de la pila. En 3 ocasiones se referenció la página que estaba una posición más abajo, y así sucesivamente. C_∞ es el número de veces que aparece ∞ como distancia en la serie de referencias.

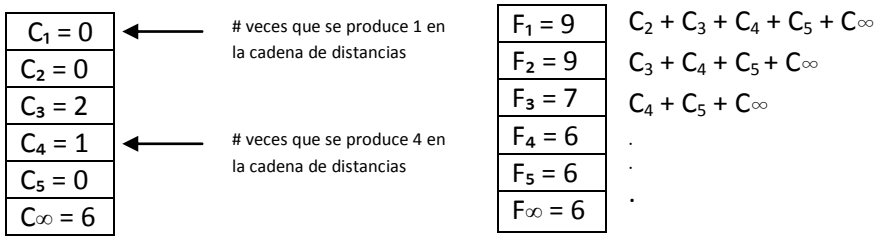


Figura 2.4 Cálculo de la tasa de faltas de página a partir de la cadena de distancias. **(a)** El vector *C*. **(b)** El vector *F*.

El vector *F* se calcula de acuerdo a la fórmula:

$$F_m = \sum_{k=m+1}^{n} C_k + C_\infty$$

m es el número de marcos de páginas.

n es el número de páginas virtuales del espacio de direcciones del proceso.

Fm número de faltas de páginas que se producirían con la serie de referencias.

El vector F nos da como resultado la cantidad de faltas de página que habrían con cierta cantidad de marcos de página, por ejemplo: F_1 es 9, esto quiere decir que con una memoria de un solo marco de página, de las 9 referencias de la cadena, todas generarán fallos de página.

Aspectos de Diseño para los Sistemas de Paginación

Tamaño de Página

Puede ser elegido por el diseñador del sistema operativo. La determinación del tamaño ideal dependerá de varios factores.

Al elegir páginas de pequeño tamaño, se evita la fragmentación interna, pero por otro lado, al tener páginas pequeñas es posible que la parte del programa presente en memoria principal esté en uso. Los inconvenientes de utilizar páginas de reducido tamaño son:

- Aumenta de forma considerable la tabla de páginas.
- Las transferencias a y desde el disco se incrementan de forma alarmante (existen pérdidas de tiempo, sobre todo por localización y demora rotacional).

En algunas máquinas, la tabla de páginas se carga en los registros del hardware cuando se pasa de un proceso a otro. Esto implica que el tiempo de carga de los registros será mayor cuanto menor sea el tamaño de la página. Además el espacio ocupado de la tabla de páginas crece.

Desde un punto de vista matemático, podemos analizar el problema:

Si *s* es el tamaño promedio de los procesos, *p* el tamaño de la página y *e* el gasto que ocasiona cada entrada de tabla, tenemos:

s/*p* = número de páginas que se necesitan por proceso.
se/*p* = espacio ocupado en la tabla de páginas (bytes).
p/*2* = memoria que se desperdicia en la última página.

Así que el costo general debido a ambos factores será:

$$\text{Costo general} = se/p + p/2$$

- Si la página es reducida, el primer término crece.
- Si la página tiene un tamaño grande, aumenta un segundo término. El tamaño adecuado debe quedar entre ambos. Para ello calculamos la derivada y la igualamos a cero.

$$-se/p + ½ = 0$$

Por lo tanto:

$$p = \sqrt{2se}$$

Espacios Separados de Instrucciones y de Datos

Casi todas las computadoras tienen un solo espacio de direcciones que contienen tanto programas como datos. Si este espacio de direcciones es lo bastante grande, todo funciona muy bien. Sin embargo, en muchas ocasiones es demasiado pequeño, y ello obliga a los programadores a hacer malabarismos para que todo quepa en el espacio de direcciones.

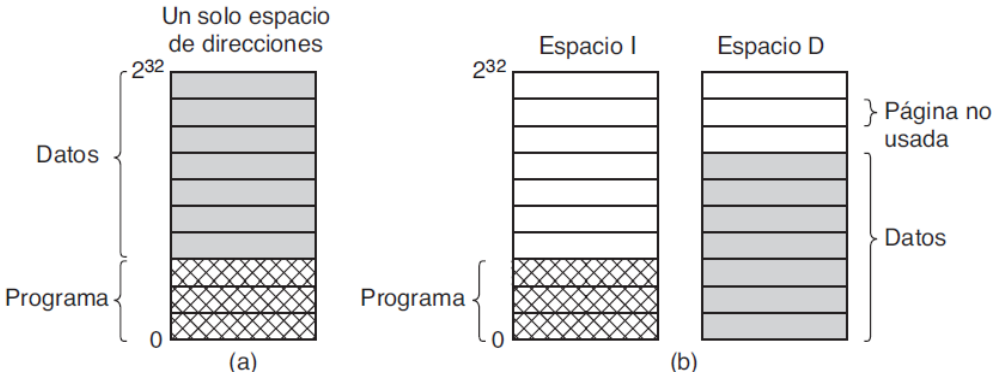


Figura 3.1. (a) Un espacio de direcciones. **(b)** Espacio I (Instrucciones) y D (Datos) separados.

Una solución es tener espacios de direcciones distintos para las instrucciones (texto de programa) y los datos, éstos se llaman espacio I y espacio D. Cada espacio de direcciones va desde 0 hasta algún máximo, por lo regular $2^{16} - 1$ o $2^{32} - 1$. El enlazador debe estar enterado del uso de espacios I y D separados, porque en tal caso los datos se reubican en la dirección virtual 0 en vez de comenzar después del programa.

En una computadora con este diseño, ambos espacios de direcciones pueden paginarse aparte. Cada uno tiene su propia tabla de páginas, con su propia correspondencia entre páginas virtuales y marcos de páginas físicos. Cuando el hardware quiere traer una instrucción, sabe que debe usar el espacio I y la tabla de páginas del espacio I. Asimismo, las referencias a los datos debe pasar por la tabla de páginas del espacio D. Fuera de esta distinción, tener espacios I y D separados no introduce complicaciones y sí duplica el espacio de direcciones disponible.

Control de Carga

Es el encargado de determinar el número de procesos que pueden estar en memoria principal, conocido como el grado de multiprogramación. La política de control de Carga es crítica para la efectividad de la gestión de memoria, ya que si en un momento dado hay pocos procesos residentes en memoria, habrán muchas ocasiones donde los procesos estarán bloqueados y se consumirá mucho tiempo en el intercambio. Por otro lado si en memoria principal residen muchos procesos, el tamaño medio del conjunto residente de procesos no será el adecuado y producirán frecuentes fallos de páginas, situación conocida como hiperpaginación.

- **Suspensión de Procesos**

Para que pueda ser reducido el grado de multiprogramación, deben suspenderse (descargarse) uno o más procesos actualmente residente. Se enumera las seis posibilidades siguientes:

- *Procesos con la prioridad más baja:* Esta alternativa toma una decisión de política de planificación y no tiene que ver con las cuestiones de rendimiento.

- *Procesos con fallos de página:* El razonamiento es que hay una probabilidad de que las tareas que provocan fallos no residente su conjunto de trabajo y, suspendiéndolas, el rendimiento pierda lo menos posible. Además, esta elección tiene resultados inmediatos, ya que bloquea un proceso que está a punto de ser bloqueado de cualquier modo y elimina así el coste de un reemplazo de página y de una operación de E/S.

- *Último proceso activado:* Este es el proceso con menos posibilidades de tener su conjunto de trabajo residente.

- *Proceso con el conjunto residente más pequeño:* Este es el proceso que necesita el menor esfuerzo futuro para volver a cargar el conjunto residente. Sin embargo, penaliza a los programas con ubicaciones pequeñas.

- *El proceso mayor:* Esta alternativa obtiene la mayor cantidad de marcos libres en una memoria muy ocupada, haciendo poco probables más desactivaciones en breve.

- *Procesos con la mayor ventana de ejecución restante:* En la mayoría de los esquemas de planificación del procesador, un proceso puede ejecutarse sólo durante cierto espacio de tiempo antes de ser interrumpido y puesto al final de la cola de Listos. Este es un enfoque parecido a la disciplina de planificación de "primero el proceso más corto".

Carga

Consiste en cargar un programa en memoria principal y crear una imagen del proceso. Este proceso es llevado a cabo por medio de un cargador que sitúa el módulo de carga en una posición en memoria.

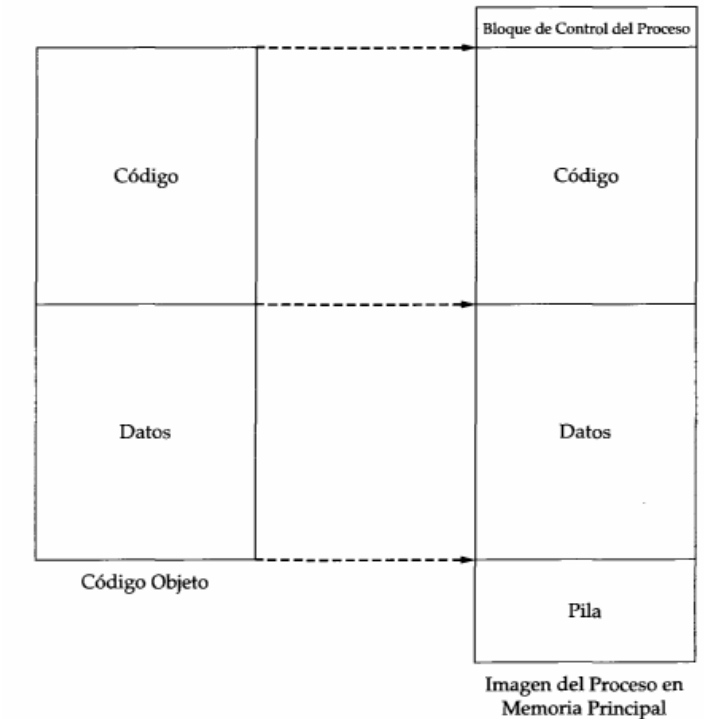


Figura 4.1 Módulo de Carga.

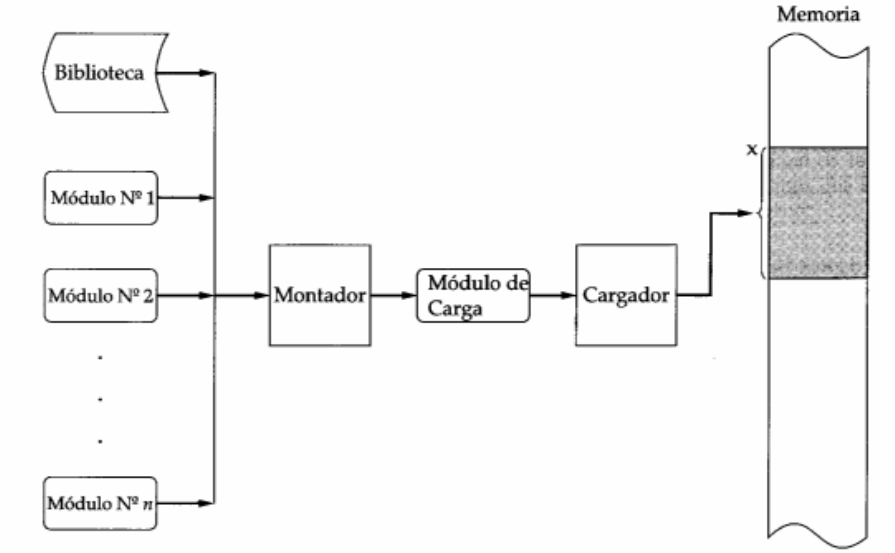


Figura 4.2 Esquema de Carga.

Carga Absoluta

La carga absoluta necesita que el módulo de carga ocupe siempre la misma posición de memoria principal. Así pues, todas las referencias del módulo de carga para el cargador deben ser direcciones específicas o absolutas en memoria principal.

La asignación de direcciones específicas a las referencias a memoria de un programa puede ser realizada tanto por el programador como en tiempo de compilación o ensamblaje.

- **Desventajas**

  ➢ Todos los programadores tendrán que conocer la estrategia de asignación deseada para situar los módulos en memoria principal.

  ➢ Si se hace alguna modificación en el programa que suponga inserciones o borrados en el cuerpo del módulo, tendrán que cambiarse todas las direcciones.

Carga Reubicable

En la carga reubicable, el ensamblador o el compilador no generará direcciones reales de memoria principal (direcciones absolutas) sino direcciones relativas a algún punto conocido, tal como el comienzo del programa.

Si el módulo va a ser cargado comenzando por la posición *x,* el cargador simplemente sumará *x* a cada referencia a memoria a medida que cargue el módulo en memoria.

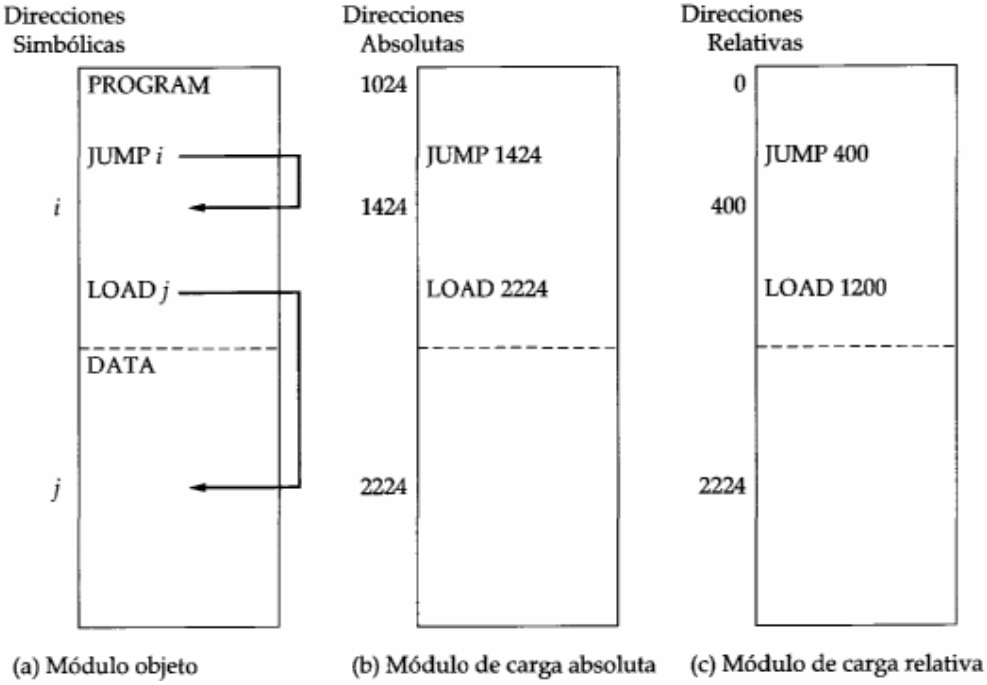


Figura 4.3 Módulos de Carga Absoluta y Reubicable

Aspectos de Implantación para los Sistemas de Paginación

Participación del Sistema Operativo en la Paginación

Hay cuatro ocasiones en las que el sistema operativo tiene que realizar trabajo relacionado con la paginación: al crear un proceso, al ejecutar un proceso, al ocurrir un fallo de página y al terminar un proceso. Ahora examinaremos brevemente cada una de estas ocasiones para ver qué se tiene que hacer.

Cuando se crea un proceso en un sistema de paginación, el sistema operativo tiene que determinar que tan grande serán el programa y los datos (al inicio), y crear una tabla de páginas para ellos.

Cuando un proceso se planifica para ejecución, la unidad de gestión de memoria se tiene que restablecer para el nuevo proceso y el TLB se vacía para deshacerse de los restos del proceso que se estaba ejecutando antes.

Cuando ocurre un fallo de pagina, el SO tiene que leer los registros de hardware para determinar cual dirección virtual produjo el fallo.

Cuando un proceso termina, el sistema operativo debe liberar su tabla de páginas, sus páginas y el espacio en disco que ocupan las páginas cuando están en disco.

Respaldo de Instrucción

Cuando un programa hace referencia a una página que no está en memoria, la instrucción que produjo el fallo se detiene antes de terminar de ejecutarse y se efectúa un salto al sistema operativo. Una vez que el sistema operativo obtiene la página necesaria, debe reiniciar la instrucción que produjo el salto.

Bloqueo de Páginas en Memoria

Si un dispositivo de E/S se encuentra en el proceso de realizar una transferencia por DMA a esa página, al eliminarla parte de los datos se escribirán el búfer al que pertenecen y parte sobre la pagina que se acaba de cargar. Una solución a este problema, es bloquear las páginas involucradas en operaciones de E/S en memoria, de manera que no se eliminen. Bloquear una página se conoce a menudo como fijada (pinning) en la memoria.

Protección

La protección de memoria evita que un proceso en un sistema operativo acceda a la memoria que no le ha sido asignada. Así pueden evitarse problemas durante la ejecución del software, y también se evita que software maligno acceda a los recursos del sistema. Estos son algunos aspectos que se deben considerar en la protección:

- Cada dirección física generada por la CPU es controlada para comprobar si es una dirección válida.

- En caso de un acceso inválido se genera una interrupción al sistema operativo.

- La unidad que convierte direcciones lógicas a físicas es la MMU (*Memory Management Unit),* y es la que controla el acceso a memoria. Esta es un dispositivo de hardware.

- La unidad MMU únicamente debe ser administrada en modo supervisor. Por ejemplo cargar los registros base y límite.

Una manera de evitar que un proceso acceda a un espacio de memoria que no le ha sido asignado es utilizando dos registros, un registro base y un registro límite. Aspectos que se consideran con registro base y registro límite son:

- Cada vez que un proceso genera una dirección se comprueba si es menor que el registro límite. En caso de serlo, la dirección física se obtiene sumando el valor del registro base. De no ser así se genera una excepción.

- El hardware ha de proporcionar dichos registros.
- El cambio de contexto actualizara el valor de dichos registros.
- El cambio de valores de los registros límite ha de ser una instrucción privilegiada.
- Este esquema proporciona además una manera de relocalización dinámica.

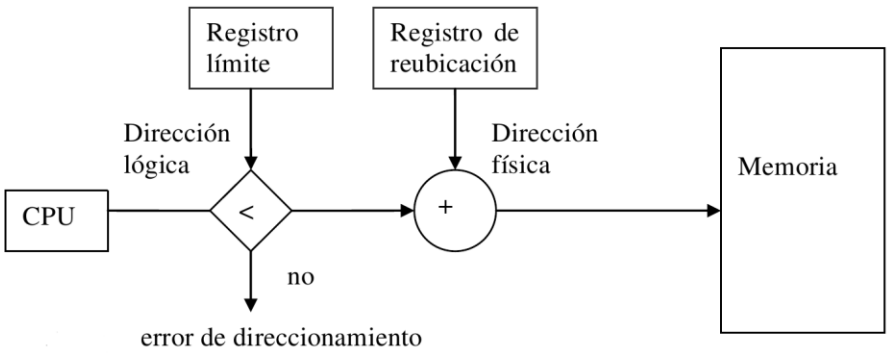


Figura 4.4 Registro Base y Límite.

Otro método es el de concepto de llave de protección, que divide la memoria física en bloques de un tamaño particular (ej. 2KB), de forma que cada uno de ellos tiene asociado un número denominado llave de protección. Cada proceso está asociado con una llave de protección. A la hora de acceder a la memoria el hardware comprueba que la llave de protección del proceso actual coincide con el del bloque de memoria al que se accede, en caso contrario se produce una excepción. Este mecanismo se usó en la arquitectura del System/360.

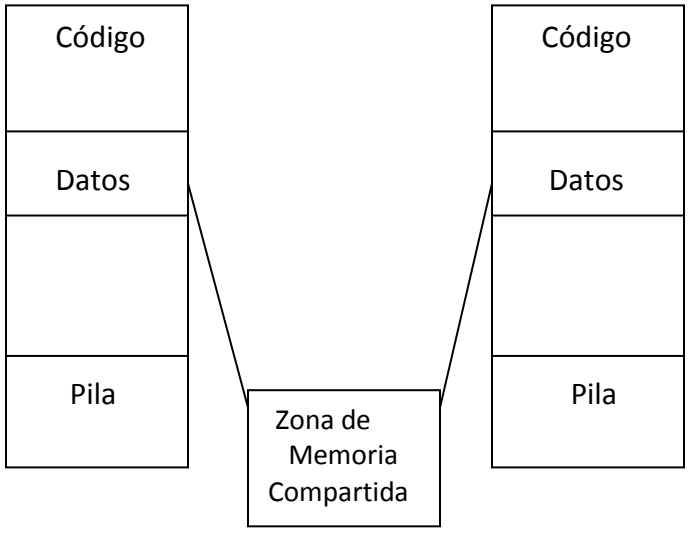
Protección de Memoria en Entorno con Paginación

- En la tabla de páginas pueden encontrarse unos bits de protección asociados a cada marco.
- Indican si la página es de sólo lectura o lectura y escritura.
- Cuando se consulta el número de marco, se consultan además los bits de protección.
- Se debe controlar que el número de página no supere el total de páginas usadas por el proceso (sería una dirección incorrecta).

Compartición

La compartición de la memoria permite comunicar procesos que se ejecutan. Esto llevaría a que las direcciones lógicas de dos o más procesos, posiblemente distintas entre sí, se correspondan con la misma dirección física. La posibilidad que dos o más procesos compartan una zona de memoria implica que el sistema de gestión de memoria debe permitir que la memoria asignada a un proceso no sea contigua.

- Debe haber flexibilidad para permitir que varios procesos accedan a una misma zona de memoria:
  - Dos procesos ejecutando el mismo programa comparten código.
  - Varios procesos pueden utilizar una misma estructura de datos.
- Se trata de permitir que direcciones lógicas de dos o más procesos posiblemente distintas entre sí, se correspondan con la misma dirección física.
- La compartición de memoria no debe comprometer la protección básica.
- Los mecanismos de paginación y segmentación resuelven adecuada y simultáneamente los problemas de reubicación, protección y compartición.



Procesos compartiendo una zona de memoria.

Páginas Compartidas:

- La paginación permite compartir código común entre varios procesos:
  - Sólo si el código es reentrante (no se modifica durante ejecución).
  - El área de datos de los procesos sería diferente.
  - Ejemplo: varios procesos ejecutan el mismo editor de textos.

Proceso 1

Memoria Lógica **Tabla de páginas**

Editor 1	
Editor 2	
Editor 3	
Datos 1	

0	3
1	4
2	6
3	1

Memoria Física

0	
1	Datos 1
2	
3	Editor 1
4	Editor 3
5	Editor 2
6	
7	Datos 2

Proceso 2

Memoria Lógica **Tabla de páginas**

Editor 1	
Editor 2	
Editor 3	
Datos 2	

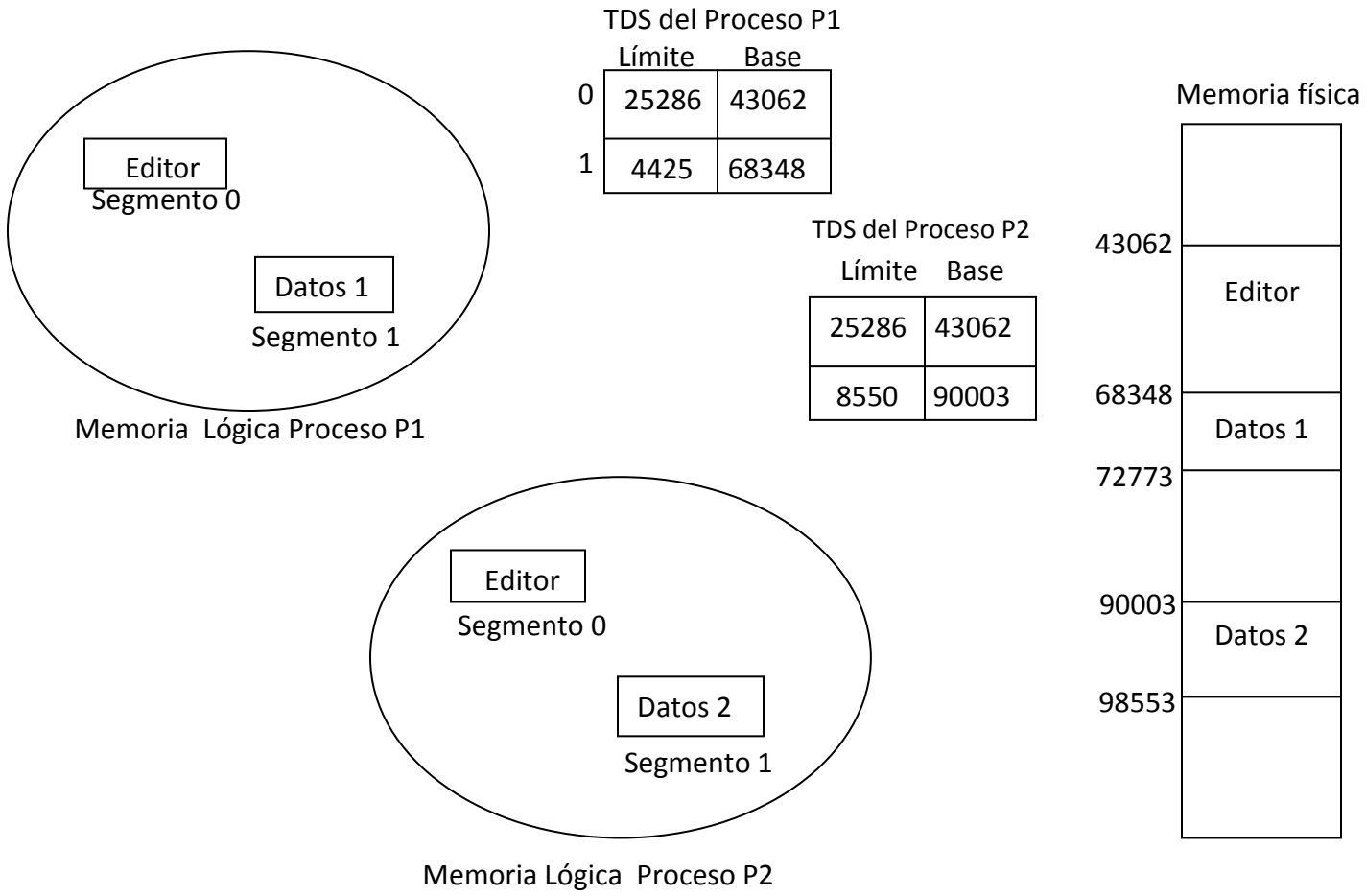
0	3
1	4
2	6
3	7

Una única copia del Editor en Memoria Física.

Segmentos Compartidos:

- La segmentación permite compartir código común entre varios procesos:
  - Puede realizarse a nivel de segmento (código o datos).
  - Cada proceso tendrá una tabla de segmento.
  - Compartir un segmento significa que una entrada de la tabla de segmentos coincide en varios procesos (igual posición física).

- Si compartimos un segmento todos los procesos que lo comparten deben definir dicho segmento con el mismo código.
- Ejemplo de Compartición Editor:



Políticas De Administración De Memoria

Políticas de Lectura

La política de lectura (fetch) está relacionada con la decisión de cuándo se debe cargar una página en memoria principal. Las dos alternativas más comunes son la paginación por demanda y la paginación previa.

- ***Paginación por demanda***: Se trae una página a memoria principal sólo cuando se hace referencia a una posición en dicha página. Si los otros elementos de la política de gestión de memoria funcionan adecuadamente, debe ocurrir lo siguiente. Cuando un proceso se ejecute por primera vez, se producirá un aluvión de fallos de página. A medida que se traigan a memoria más páginas, el principio de cercanía hará que la mayoría de las futuras referencias estén en páginas que se han cargado hace poco. Así pues, después de un tiempo, la situación se estabilizará y el número de fallos de página disminuirá hasta un nivel muy bajo.

- ***Paginación previa***: Se cargan otras páginas distintas a las demandadas debido a un fallo de página. El principal atractivo de esta estrategia está en las características de la mayoría de los dispositivos de memoria secundaria, como los discos, que tienen un tiempo de búsqueda y una latencia de giro. Si las páginas de un proceso se cargan secuencialmente en memoria secundaria, es más eficiente traer a memoria un número de páginas contiguas de una vez que ir trayéndolas de una en una durante un periodo largo de tiempo.

Esta política no es efectiva si la mayoría de las páginas extra que se traen no se referencian. La utilidad no ha sido demostrada.

Políticas de Ubicación

Tiene que ver con determinar dónde va a residir una parte de un proceso en memoria principal. En un sistema de segmentación puro, la política de ubicación es un aspecto importante del diseño; como posibles alternativas se tienen las políticas del mejor ajuste, el primer ajuste, peor ajuste y otras.

- ***Mejor Ajuste***: Colocar el proceso en el menor bloque en el que quepa.
- ***Peor Ajuste***: Colocar el proceso en el espacio más grande en el que quepa.
- ***Primer Ajuste***: Colocar el proceso en el primer espacio de la lista de almacenamiento libre en el que quepa.

Políticas de Reemplazo

Seleccionar la página a reemplazar en memoria principal cuando se debe cargar una nueva página.

- ***Reemplazo de Página Local***: Asignan a cada proceso una fracción fija de la memoria.
- ***Reemplazo de Página Global***: Asignan dinámicamente los marcos de páginas entre los procesos ejecutables. Así, el número de marcos de página asignados a cada proceso varía con el tiempo.

Gestión de Conjunto Residente

- **Tamaño del conjunto residente**

Con memoria virtual paginada no es necesario y, puede no ser posible, traer todas las páginas de un proceso a la memoria principal para preparar su ejecución. El sistema operativo debe decidir cuantas páginas traer, es decir, cuanta memoria principal asignar a un proceso. Aquí entran en juego varios factores:

- Cuanto menor es la cantidad de memoria asignada a un proceso, mayor es el número de procesos que pueden estar en la memoria principal. Esto aumenta la

probabilidad de que el sistema operativo encuentre al menos un proceso Listo y, por lo tanto, reduzca el tiempo perdido en el intercambio.

- Si en la memoria principal hay un número relativamente pequeño de páginas de un proceso entonces a pesar del principio de cercanía, el porcentaje de fallos de página será algo mayor.

- La asignación de memoria adicional a un proceso en particular no tendrá efectos notables en el porcentaje de fallos de página para ese proceso debido al principio de cercanía.

Con estos factores en los sistemas operativos actuales se pueden encontrar con dos tipos de políticas. La **política de asignación fija** otorga a cada proceso un número fijo de páginas en las que ejecutar. Con una política de asignación fija, cada vez que se produce un fallo de página en la ejecución de un proceso, se debe reemplazar una de las páginas de dicho procesador la página que se necesite.

La **política de asignación variable** permite que el núcleo de marcos asignados a un proceso cambie a lo largo de su vida.

La política de asignación variable parece ser la más potente. La dificultad de este método está en que requiere que el sistema operativo evalué el compartimiento de los procesos activos.

◦ Alcance del Reemplazo

	Reemplazo Local	**Reemplazo Global**
Asignación Fija	El número de marcos asignados a un proceso es fijo. La página a reemplazar se elige de entre los marcos asignados al proceso.	No es posible
Asignación Variable	El número de marcos asignados a un proceso puede cambiar de un momento a otro para mantener su conjunto de trabajo. La página a reemplazar se elige de entre las páginas asignadas al proceso	La página a reemplazar se elige de entre todos los marcos disponibles en memoria principal; esto hace que cambie el tamaño del conjunto residente de los procesos

Políticas de Vaciado

Son las encargadas de tomar la decisión de cuando una página que ha sido modificada, debe ser escrita en memoria secundaria. Entre las alternativas más utilizadas se encuentran:

- Vaciado por demanda.
- Vaciado Previo.

Vaciado por Demanda

Utilizando vaciado por demanda, las páginas modificadas no serán escritas en memoria secundaria hasta que estas no hayan sido escogidas para ser reemplazadas, y de esta manera ceder su marco a otra página.

Vaciado Previo

Utilizando vaciado previo, las páginas que han sido modificadas serán escritas en memoria secundaria antes que se requieran sus marcos, permitiendo de esta manera que las paginas puedan escribirse por lotes.

Utilizando cualquiera de las dos políticas, podrían llevar a un peligro. En el vaciado por demanda la escritura de una página modificada es anterior a la lectura de una nueva página. Esta técnica puede minimizar la escritura de páginas, pero hace que un proceso que sufra un fallo de página pueda que tenga que esperar dos transferencias de páginas antes de bloquearse. Mientras que con vaciado previo, una página se escribe pero reside en memoria principal hasta que el algoritmo de reemplazo de página diga que se suprime. Además esta técnica permite que se escriban las páginas por lotes, pero se perdería eficiencia cuando la mayoría de las páginas que se están escribiendo han sido nuevamente modificadas.

Alternativa:

Existe una mejor solución que sería la de incorporar un almacenamiento intermedio de páginas, que permitiría la adopción de la siguiente política: se cuenta dos listas, una de páginas modificadas y otras de no modificada. Las páginas de la lista de modificadas pueden escribirse periódicamente por lotes y trasladarse a la lista de no modificadas. Una página de la lista de no modificadas pueden reclamarse, si se le hace de nuevo referencia o perderse, cuando se asigna su marco a otra página.

Referencias Bibliográficas

Stallings, W. (1997). *Sistemas Operativos 2ed.* Madrid: PRENTICE HALL.

Tanenbaum A. (2003). *Sistemas Operativos Modernos 2ed*. México: PEARSON EDUCACIÓN.

Tanenbaum A. (2009). *Sistemas Operativos Modernos 3ed*. México: PRENTICE HALL.