

Sistemas Operativos

Administración de la memoria

Clase 9 - Unidad III

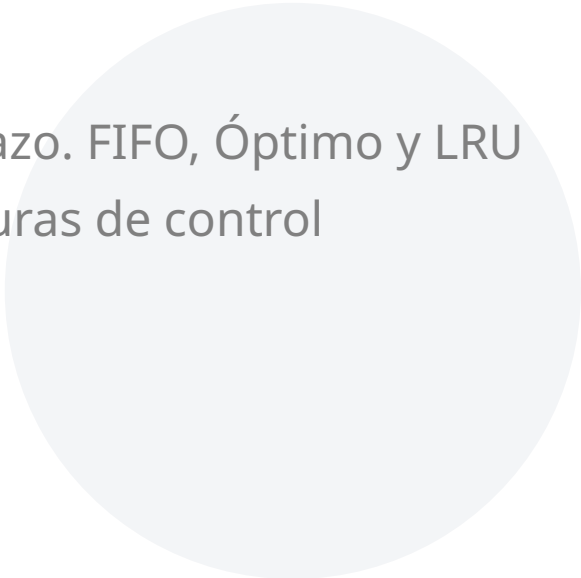
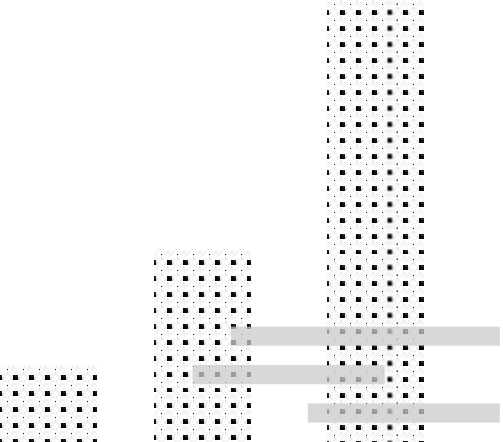
Lic. Alexis Sostersich
sostersich.alexis@uader.edu.ar



Teoría – Sistemas Operativos

Administración de la memoria

Unidad III – Clase 9

- Memoria virtual
 - Políticas de reemplazo. FIFO, Óptimo y LRU
 - Hardware y estructuras de control
- 
- 

Memoria virtual

Mientras que los registros base y límite se pueden utilizar para crear la abstracción de los espacios de direcciones, hay otro problema que se tiene que resolver: la administración del agrandamiento del software (bloatware).

Aunque el tamaño de las memorias se incrementa con cierta rapidez, el del software aumenta con una mucha mayor. En la década de 1980, muchas universidades operaban un sistema de tiempo compartido con docenas de usuarios (más o menos satisfechos) trabajando simultáneamente en una VAX de 4 MB.

Mas cerca en el tiempo, Microsoft recomendaba tener por lo menos 512 MB para que un sistema Vista de un solo usuario ejecute aplicaciones simples y 1 GB si el usuario va a realizar algún trabajo serio. La tendencia hacia la multimedia impone aún mayores exigencias sobre la memoria.

Memoria virtual

Como consecuencia de estos desarrollos, existe la necesidad de ejecutar programas que son demasiado grandes como para caber en la memoria y sin duda existe también la necesidad de tener sistemas que puedan soportar varios programas ejecutándose al mismo tiempo, cada uno de los cuales cabe en memoria, pero que en forma colectiva exceden el tamaño de la misma.

El intercambio no es una opción atractiva, ya que un disco SATA ordinario tiene una velocidad de transferencia pico de 100 MB/segundo a lo más, lo cual significa que requiere por lo menos 10 segundos para intercambiar un programa de 1 GB de memoria a disco y otros 10 segundos para intercambiar un programa de 1 GB del disco a memoria.

El problema de que los programas sean más grandes que la memoria ha estado presente desde los inicios de la computación, en áreas limitadas como la ciencia y la ingeniería (para simular la creación del universo o, incluso, para simular una nueva aeronave, se requiere mucha memoria).

Memoria virtual

Una solución que se adoptó en la década de 1960 fue dividir los programas en pequeñas partes, conocidas como sobrepuestos (overlays). Cuando empezaba un programa, todo lo que se cargaba en memoria era el administrador de sobrepuestos, que de inmediato cargaba y ejecutaba el sobrepuesto 0; cuando éste terminaba, indicaba al administrador de sobrepuestos que cargara el 1 encima del sobrepuesto 0 en la memoria (si había espacio) o encima del mismo (si no había espacio). Algunos sistemas de sobrepuestos eran muy complejos, ya que permitían varios sobrepuestos en memoria a la vez. Los sobrepuestos se mantenían en el disco, intercambiándolos primero hacia adentro de la memoria y después hacia afuera de la memoria mediante el administrador de sobrepuestos. Aunque el trabajo real de intercambiar sobrepuestos hacia adentro y hacia afuera de la memoria lo realizaba el sistema operativo, el de dividir el programa en partes tenía que realizarlo el programador en forma manual.

Memoria virtual

El proceso de dividir programas grandes en partes modulares más pequeñas consumía mucho tiempo, y era aburrido y propenso a errores. No pasó mucho tiempo antes de que alguien ideara una forma de pasar todo el trabajo a la computadora.

El método ideado (Fotheringham, 1961) se conoce actualmente como memoria virtual. La idea básica detrás de la memoria virtual es que cada programa tiene su propio espacio de direcciones, el cual se divide en trozos llamados páginas. Cada página es un rango contiguo de direcciones. Estas páginas se asocian a la memoria física, pero no todas tienen que estar en la memoria física para poder ejecutar el programa. Cuando el programa hace referencia a una parte de su espacio de direcciones que está en la memoria física, el hardware realiza la asociación necesaria al instante. Cuando el programa hace referencia a una parte de su espacio de direcciones que no está en la memoria física, el sistema operativo recibe una alerta para buscar la parte faltante y volver a ejecutar la instrucción que falló.

Memoria virtual

En cierto sentido, la memoria virtual es una generalización de la idea de los registros base y límite. El procesador 8088 tenía registros base separados (pero no tenía registros límite) para texto y datos. Con la memoria virtual, en vez de tener una reubicación por separado para los segmentos de texto y de datos, todo el espacio de direcciones se puede asociar a la memoria física en unidades pequeñas equitativas.

La memoria virtual funciona muy bien en un sistema de multiprogramación, con bits y partes de muchos programas en memoria a la vez. Mientras un programa espera a que una parte del mismo se lea y coloque en la memoria, la CPU puede otorgarse a otro proceso.

Paginación

La mayor parte de los sistemas de memoria virtual utilizan una técnica llamada paginación. En cualquier computadora, los programas hacen referencia a un conjunto de direcciones de memoria. Por ejemplo:

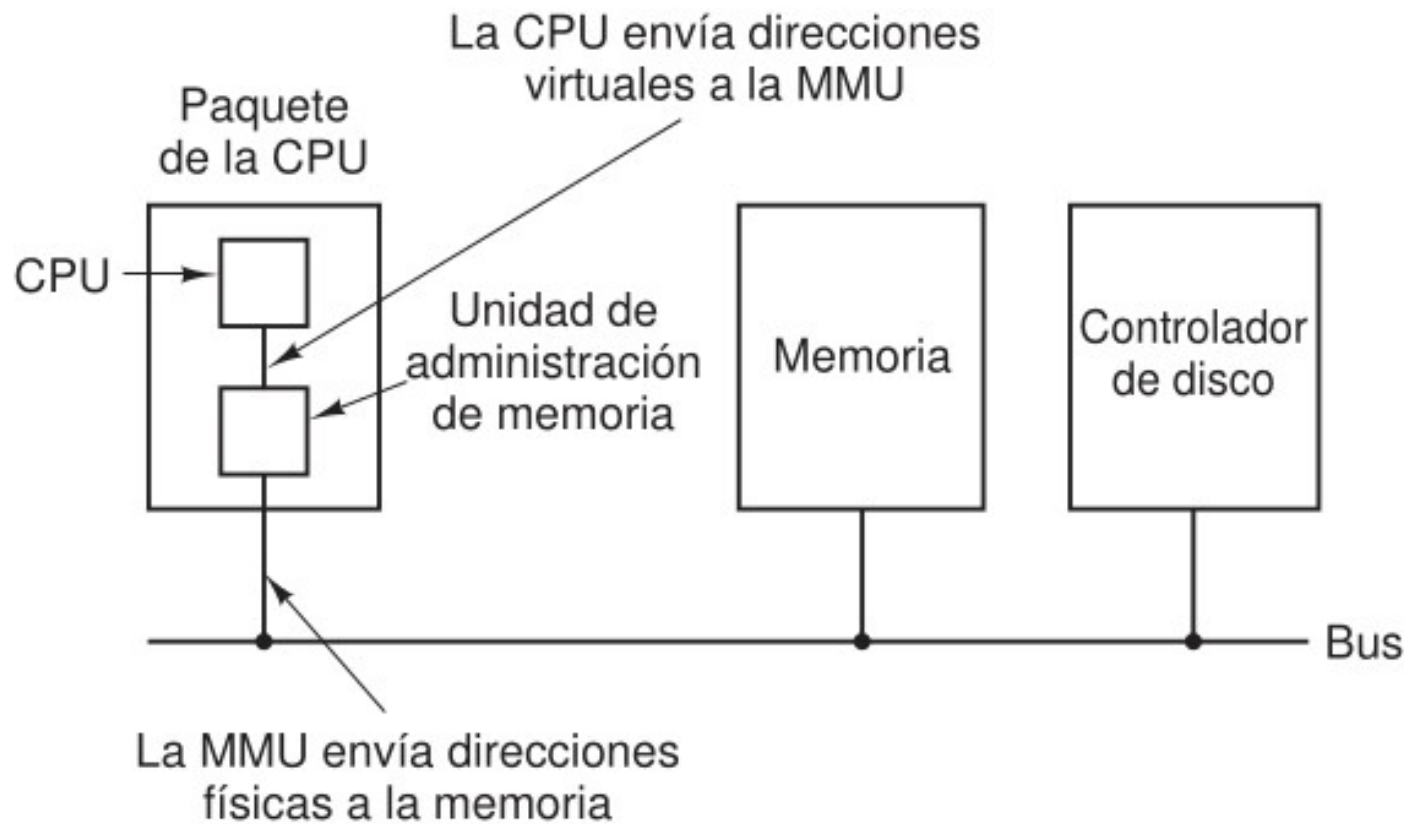
```
MOV REG, 1000
```

sirve para copiar el contenido de la dirección de memoria 1000 a REG (o viceversa, dependiendo de la PC).

Estas direcciones generadas por el programa se conocen como direcciones virtuales y forman el espacio de direcciones virtuales. En las computadoras sin memoria virtual, la dirección física se coloca directamente en el bus de memoria y hace que se lea o escriba la palabra de memoria física con la misma dirección. Cuando se utiliza memoria virtual, las direcciones virtuales no van directamente al bus de memoria. En vez de ello, van a una MMU (Memory Management Unit, Unidad de administración de memoria) que asocia las direcciones virtuales a las direcciones de memoria físicas

Paginación

La posición y función de la MMU. Aquí la MMU se ve como parte del chip de CPU, es común esta configuración en la actualidad. Sin embargo, lógicamente podría ser un chip separado y lo era hace años.



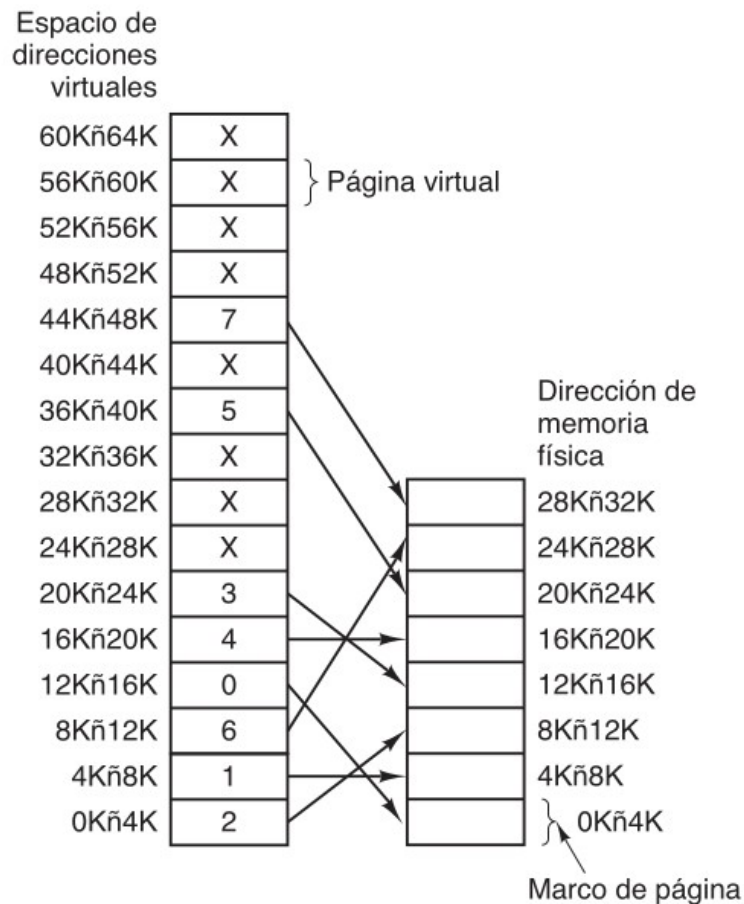
Paginación

En la figura vemos un ejemplo muy simple de cómo funciona esta asociación. Tenemos una computadora que genera direcciones de 16 bits, desde 0 hasta 64 K. Éstas son las direcciones virtuales. Sin embargo, esta computadora sólo tiene 32 KB de memoria física. Así, aunque se pueden escribir programas de 64 KB, no se pueden cargar completos en memoria y ejecutarse. No obstante, una copia completa de la imagen básica de un programa, de hasta 64 KB, debe estar presente en el disco para que las partes se puedan traer a la memoria según sea necesario.

El espacio de direcciones virtuales se divide en unidades de tamaño fijo llamadas páginas. Las unidades correspondientes en la memoria física se llaman marcos de página. Las páginas y los marcos de página por lo general son del mismo tamaño. En este ejemplo son de 4 KB, pero en sistemas reales se han utilizado tamaños de página desde 512 bytes hasta 64 KB. Con 64 KB de espacio de direcciones virtuales y 32 KB de memoria física obtenemos 16 páginas virtuales y 8 marcos de página. Las transferencias entre la RAM y el disco siempre son en páginas completas.

Paginación

La relación entre las direcciones virtuales y las direcciones de memoria física está dada por la tabla de páginas



Paginación

Formatos típicos de gestión de memoria

Dirección virtual

Número de página	Desplazamiento
------------------	----------------

Entrada de la tabla de páginas

P	M	Otros bits de control	Número de marco
---	---	-----------------------	-----------------

(a) Únicamente paginación

Dirección virtual

Segmento de página	Desplazamiento
--------------------	----------------

Entrada de la tabla de segmentos

P	M	Otros bits de control	Longitud	Comienzo de segmento
---	---	-----------------------	----------	----------------------

(b) Únicamente segmentación

Dirección virtual

Segmento de página	Número de página	Desplazamiento
--------------------	------------------	----------------

Entrada de la tabla de segmentos

Bits de control	Longitud	Comienzo de segmento
-----------------	----------	----------------------

Entrada de la tabla de páginas

P	M	Otros bits de control	Número de marco
---	---	-----------------------	-----------------

P = bit de presente
M = bit de modificado

Tablas de páginas

En una implementación simple, la asociación de direcciones virtuales a direcciones físicas se puede resumir de la siguiente manera: la dirección virtual se divide en un número de página virtual (bits de mayor orden) y en un desplazamiento (bits de menor orden). Por ejemplo, con una dirección de 16 bits y un tamaño de página de 4 KB, los 4 bits superiores podrían especificar una de las 16 páginas virtuales y los 12 bits inferiores podrían entonces especificar el desplazamiento de bytes (0 a 4095) dentro de la página seleccionada. Sin embargo, también es posible una división con 3, 5 u otro número de bits para la página. Las distintas divisiones implican diferentes tamaños de página.

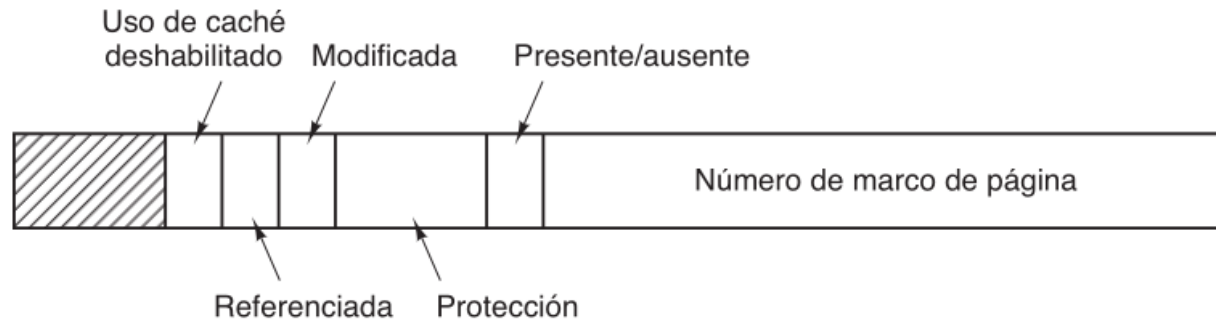
Tablas de páginas

El número de página virtual se utiliza como índice en la tabla de páginas para buscar la entrada para esa página virtual. En la entrada en la tabla de páginas, se encuentra el número de marco de página (si lo hay). El número del marco de página se adjunta al extremo de mayor orden del desplazamiento, reemplazando el número de página virtual, para formar una dirección física que se pueda enviar a la memoria.

Por ende, el propósito de la tabla de páginas es asociar páginas virtuales a los marcos de página. Hablando en sentido matemático, la tabla de páginas es una función donde el número de página virtual es un argumento y el número de marco físico es un resultado. Utilizando el resultado de esta función, el campo de la página virtual en una dirección virtual se puede reemplazar por un campo de marco de página, formando así una dirección de memoria física.

Estructura de la tabla de páginas

La distribución exacta de una entrada depende en gran parte de la máquina, pero el tipo de información presente es aproximadamente el mismo de una máquina a otra. El tamaño varía de una computadora a otra, pero 32 bits es un tamaño común. El campo más importante es el número de marco de página. Después de todo, el objetivo de la asociación de páginas es mostrar este valor. Enseguida de este campo tenemos el bit de presente/ausente. Si este bit es 1, la entrada es válida y se puede utilizar. Si es 0, la página virtual a la que pertenece la entrada no se encuentra actualmente en la memoria. Al acceder a una entrada en la tabla de página con este bit puesto en 0 se produce un fallo de página.



Estructura de la tabla de páginas

Los bits de protección indican qué tipo de acceso está permitido. En su forma más simple, este campo contiene 1 bit, con 0 para lectura/escritura y 1 para sólo lectura. Un arreglo más sofisticado es tener 3 bits: uno para habilitar la lectura, otro para la escritura y el tercero para ejecutar la página.

Los bits de modificada y referenciada llevan el registro del uso de páginas. Cuando se escribe en una página, el hardware establece de manera automática el bit de modificada. Este bit es valioso cuando el sistema operativo decide reclamar un marco de página. Si la página en él ha sido modificada ("sucia"), debe escribirse de vuelta en el disco. Si no se ha modificado ("limpia") sólo se puede abandonar, debido a que la copia del disco es aun válida. A este bit se le conoce algunas veces como bit sucio, ya que refleja el estado de la página.

Estructura de la tabla de páginas

El bit de referenciada se establece cada vez que una página es referenciada, ya sea para leer o escribir. Su función es ayudar al sistema operativo a elegir una página para desalojarla cuando ocurre un fallo de página. Las páginas que no se estén utilizando son mejores candidatos que las páginas que se están utilizando y este bit desempeña un importante papel en varios de los algoritmos de reemplazo de páginas.

Finalmente, el último bit permite deshabilitar el uso de caché para la página. Esta característica es importante para las páginas que se asocian con los registros de dispositivos en vez de la memoria. Si el sistema operativo está esperando en un ciclo de espera hermético a que cierto dispositivo de E/S responda a un comando que acaba de recibir, es esencial que el hardware siga obteniendo la palabra del dispositivo y no utilice una copia antigua en la caché. Con este bit, el uso de la caché se puede desactivar. Las máquinas que tienen un espacio de E/S separado y no utilizan E/S asociada a la memoria no necesitan este bit.

Aceleración de la paginación

En cualquier sistema de paginación hay que abordar dos cuestiones:

- 1) La asociación de una dirección virtual a una dirección física debe ser rápida.
- 2) Si el espacio de direcciones virtuales es grande, la tabla de páginas será grande.

El primer punto es una consecuencia del hecho de que la asociación virtual-a-física debe realizarse en cada referencia de memoria. Todas las instrucciones deben provenir finalmente de la memoria y muchas de ellas hacen referencias a operandos en memoria también.

El segundo punto se deriva del hecho de que todas las computadoras modernas utilizan direcciones virtuales de por lo menos 32 bits, donde 64 bits se vuelven cada vez más comunes. Por decir, con un tamaño de página de 4 KB, un espacio de direcciones de 32 bits tiene 1 millón de páginas y un espacio de direcciones de 64 bits tiene más de las que desearíamos contemplar.

Búferes de traducción adelantada

Ahora veamos esquemas implementados ampliamente para acelerar la paginación y manejar espacios de direcciones virtuales extensos, empezando con la aceleración de la paginación. El punto inicial de la mayor parte de las técnicas de optimización es que la tabla de páginas está en la memoria. Potencialmente, este diseño tiene un enorme impacto sobre el rendimiento.

Los diseñadores de computadoras han sabido acerca de este problema durante años y han ideado una solución que está basada en la observación de que la mayor parte de los programas tienden a hacer un gran número de referencias a un pequeño número de páginas y no viceversa. Por ende, sólo se lee con mucha frecuencia una pequeña fracción de las entradas en la tabla de páginas; el resto se utiliza muy pocas veces.

TLB (Búfer de traducción adelantada)

La solución que se ha ideado es equipar a las computadoras con un pequeño dispositivo de hardware para asociar direcciones virtuales a direcciones físicas sin pasar por la tabla de páginas. El dispositivo, llamado TLB (Translation Lookaside Buffer, Búfer de traducción adelantada) o algunas veces memoria asociativa.

Por lo general se encuentra dentro de la MMU y consiste en un pequeño número de entradas, ocho en este ejemplo, pero raras veces más de 64. Cada entrada contiene información acerca de una página, incluyendo el número de página virtual, un bit que se establece cuando se modifica la página, el código de protección (permisos de lectura/escritura/ejecución) y el marco de página físico en el que se encuentra la página.

TLB (Búfer de traducción adelantada)

Un TLB para acelerar la paginación

Válida	Página virtual	Modificada	Protección	Marco de página
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Gestión de TLB mediante software

Hemos supuesto que toda máquina con memoria virtual paginada tiene tablas por hardware, más un TLB. En este diseño, la administración y el manejo de fallas del TLB se realiza por completo mediante el hardware de la MMU. Las traps, o trampas, para el sistema operativo ocurren sólo cuando una página no se encuentra en memoria.

En el pasado, esta suposición era correcta. Sin embargo, muchas máquinas RISC modernas (incluyendo SPARC, MIPS y HP PA) hacen casi toda esta administración de páginas mediante software. En estas máquinas, las entradas del TLB se cargan de manera explícita mediante el sistema operativo. Cuando no se encuentra una coincidencia en el TLB, en vez de que la MMU vaya a las tablas de páginas para buscar y obtener la referencia a la página que se necesita, sólo genera un fallo del TLB y pasa el problema al sistema operativo.

El beneficio principal aquí es una MMU mucho más simple, lo cual libera una cantidad considerable de área en el chip de CPU para cachés y otras características que pueden mejorar el rendimiento.

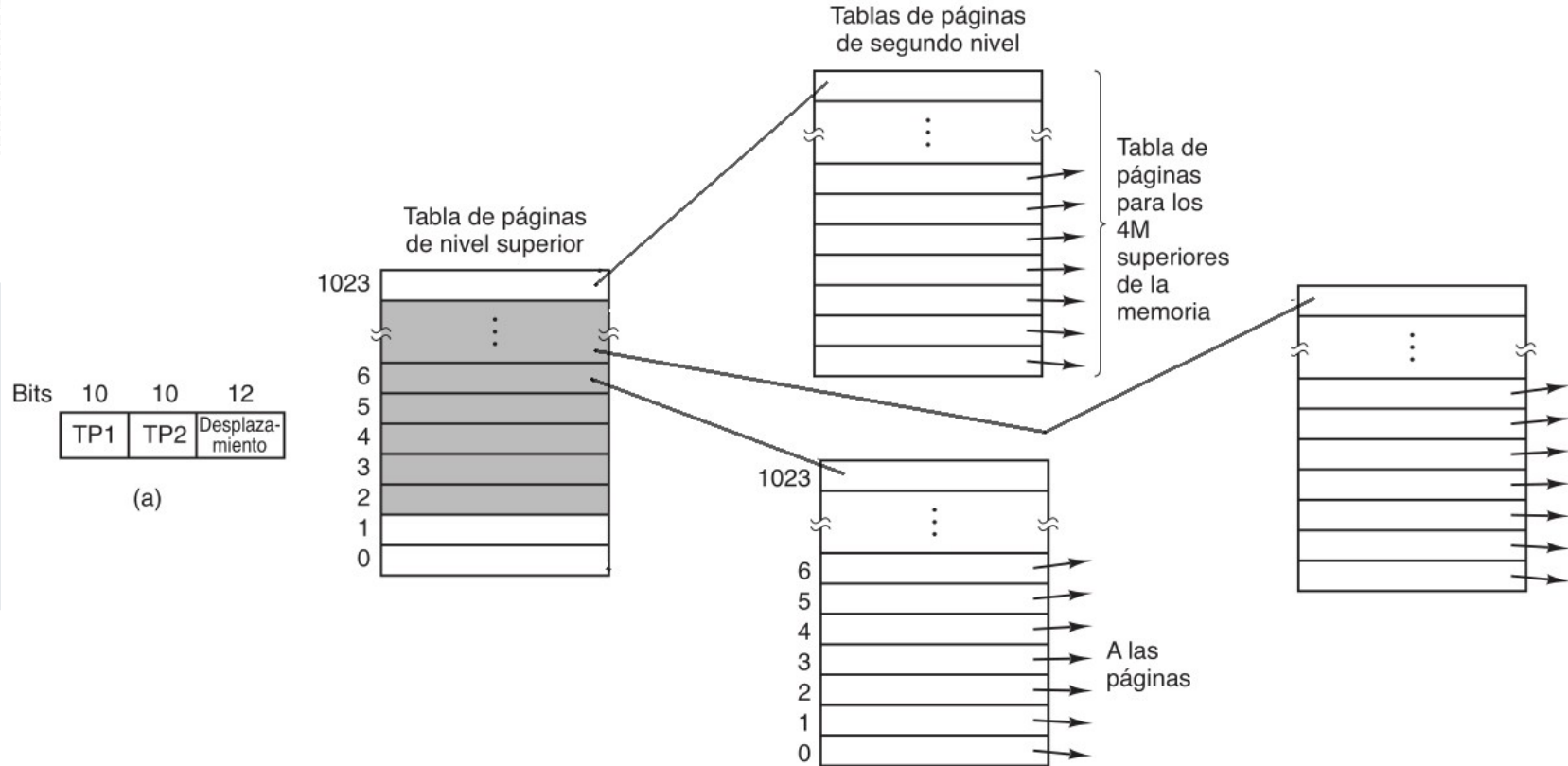
Tablas de páginas para memorias extensas

Los TLBs se pueden utilizar para acelerar las traducciones de direcciones virtuales a direcciones físicas sobre el esquema original de la tabla de páginas en memoria. Pero ése no es el único problema que debemos combatir. Otro problema es cómo lidiar con espacios de direcciones virtuales muy extensos.

Tablas de páginas multinivel

El secreto del método de la tabla de páginas multinivel es evitar mantenerlas en memoria todo el tiempo, y en especial, aquellas que no se necesitan. Suponga que un proceso necesita 12 Mb: los 4 Mb inferiores de memoria para el texto del programa, los siguientes 4 Mb para datos y los 4 Mb superiores para la pila. Entre la parte superior de los datos y la parte inferior de la pila hay un hueco gigantesco que no se utiliza.

Estructura de tabla de páginas multinivel



Tablas de páginas para memorias extensas

Tablas de páginas invertidas

Para los espacios de direcciones virtuales de 32 bits, la tabla de páginas multinivel funciona bastante bien. Sin embargo, a medida que las computadoras de 64 bits se hacen más comunes, la situación cambia de manera drástica. Si el espacio de direcciones es de 264 bytes, con páginas de 4 KB, necesitamos una tabla de páginas con 252 entradas. Si cada entrada es de 8 bytes, la tabla es de más de 30 millones de gigabytes (30 PB).

Una de esas soluciones es la tabla de páginas invertida. En este diseño hay una entrada por cada marco de página en la memoria real, en vez de tener una entrada por página de espacio de direcciones virtuales. Por ejemplo, con direcciones virtuales de 64 bits, una página de 4 KB y 1 GB de RAM, una tabla de páginas invertida sólo requiere 262,144 entradas. La entrada lleva el registro de quién (proceso, página virtual) se encuentra en el marco de página.

Tablas de páginas para memorias extensas

Tablas de páginas invertidas

Aunque las tablas de página invertidas ahorran grandes cantidades de espacio, al menos cuando el espacio de direcciones virtuales es mucho mayor que la memoria física, tienen una seria desventaja: la traducción de dirección virtual a dirección física se hace mucho más difícil. Cuando el proceso n hace referencia a la página virtual p , el hardware ya no puede buscar la página física usando p como índice en la tabla de páginas. En vez de ello, debe buscar una entrada (n, p) en toda la tabla de páginas invertida. Lo que es peor: esta búsqueda se debe realizar en cada referencia a memoria, no sólo en los fallos de página. Buscar en una tabla de 256 K en cada referencia a memoria no es la manera de hacer que la máquina sea deslumbrantemente rápida.

Tablas de páginas para memorias extensas

Tablas de páginas invertidas

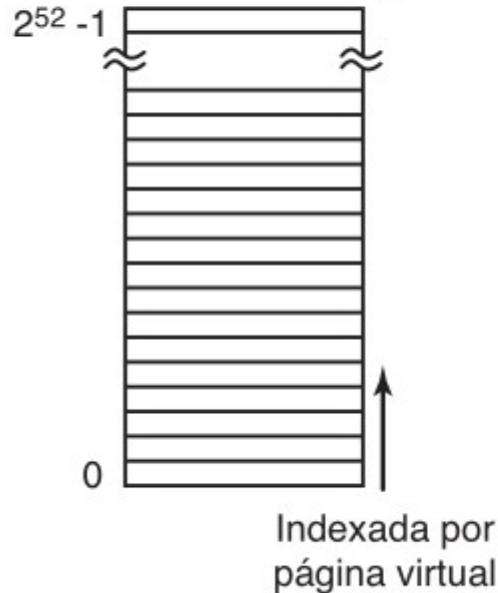
La forma de salir de este dilema es utilizar el TLB. Si el TLB puede contener todas las páginas de uso frecuente, la traducción puede ocurrir con igual rapidez que con las tablas de páginas regulares.

Sin embargo, en un fallo de TLB la tabla de páginas invertida tiene que buscarse mediante software. Una manera factible de realizar esta búsqueda es tener una tabla de hash arreglada según el hash de la dirección virtual.

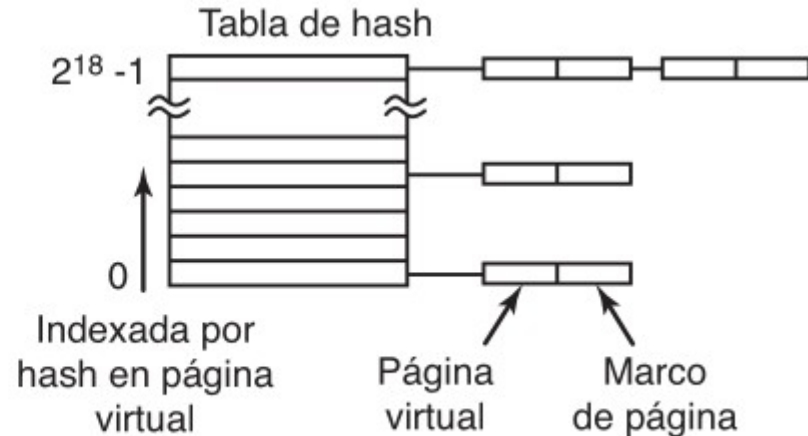
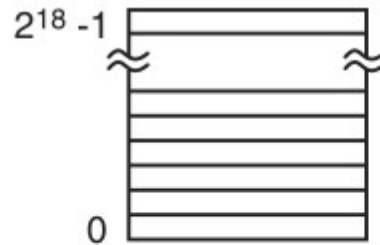
Todas las páginas virtuales que se encuentren en memoria y tengan el mismo valor de hash se encadenan en conjunto. Si la tabla de hash tiene tantas ranuras como las páginas físicas de la máquina, la cadena promedio tendrá solo una entrada, con lo cual se acelera de manera considerable la asociación. Una vez que se ha encontrado el número de marco de página, se introduce el nuevo par (virtual, física) en el TLB.

Tablas de páginas para memorias extensas

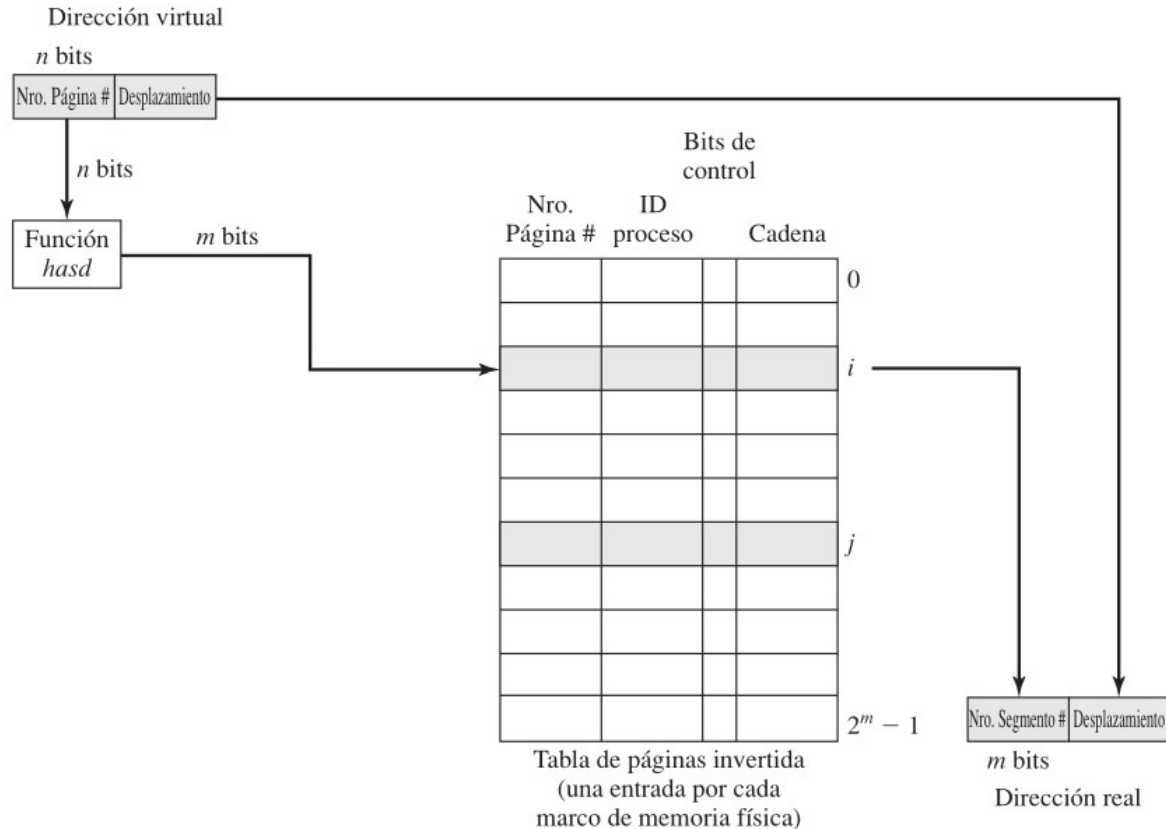
Tabla de páginas tradicional con una entrada para cada una de las 2^{52} páginas

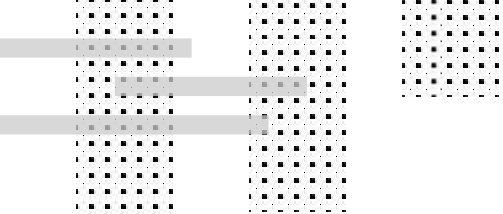


La memoria física de 1 GB tiene 2^{18} marcos de página de 4 KB

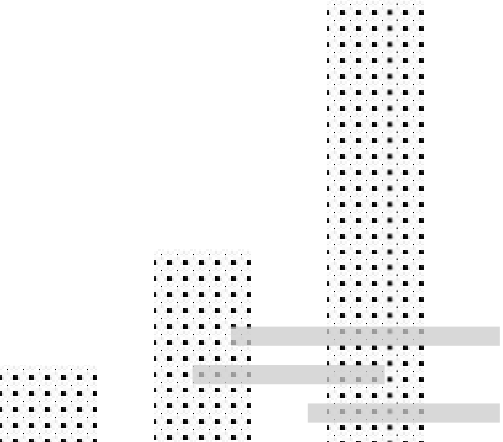
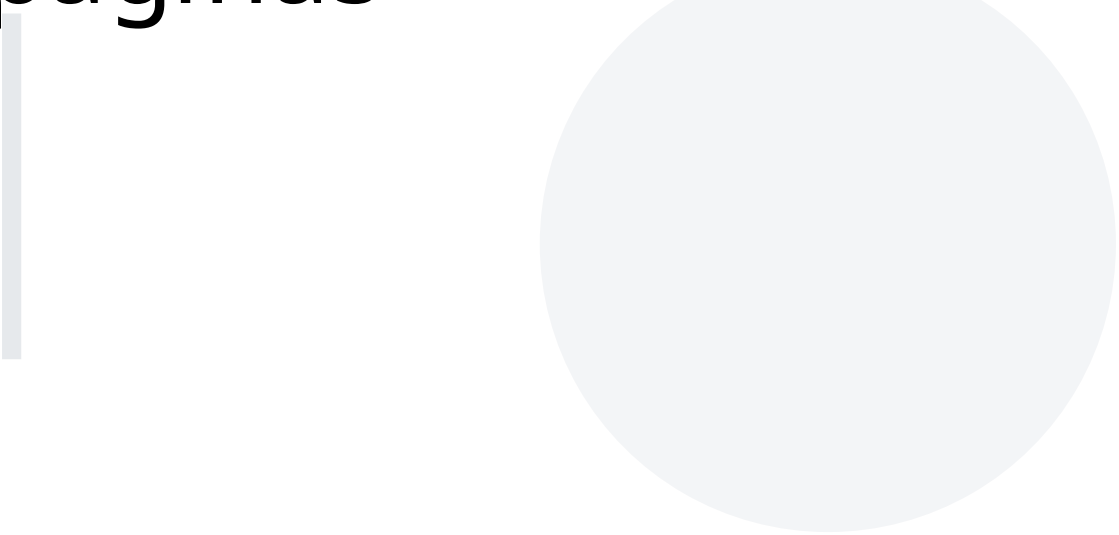


Estructura de tabla de páginas invertida





Algoritmos de reemplazo de páginas



Algoritmos de reemplazo de páginas

Cuando ocurre un fallo de página, el sistema operativo tiene que elegir una página para desalojarla (eliminarla de memoria) y hacer espacio para la página entrante. Si la página a eliminar se modificó mientras estaba en memoria, debe volver a escribirse en el disco para actualizar la copia del mismo. No obstante, si la página no se ha modificado (por ejemplo, si contiene el texto del programa), la copia ya está actualizada y no se necesita rescribir. La página que se va a leer sólo sobrescribe en la página que se va a desalojar.

Aunque sería posible elegir una página al azar para desalojarla en cada fallo de página, el rendimiento del sistema es mucho mayor si se selecciona una página que no sea de uso frecuente. Si se elimina una página de uso frecuente, tal vez tenga que traerse de vuelta rápidamente, lo cual produce una sobrecarga adicional. Se ha realizado mucho trabajo, tanto teórico como experimental en el tema de los algoritmos de reemplazo de páginas.

El algoritmo de reemplazo de páginas óptimo

El mejor algoritmo de reemplazo de páginas posible es fácil de describir, pero imposible de implementar: al momento en que ocurre un fallo de página, hay cierto conjunto de páginas en memoria y una de éstas se referenciará en la siguiente instrucción (la página que contiene la instrucción); otras páginas tal vez no se referencien sino hasta 10, 100 o tal vez 1000 instrucciones después. Cada página se puede etiquetar con el número de instrucciones que se ejecutarán antes de que se haga referencia por primera vez a esa página.

El algoritmo óptimo de reemplazo de páginas establece que la página con la etiqueta más alta debe eliminarse. Si una página no se va a utilizar durante 8 millones de instrucciones y otra no se va a utilizar durante 6 millones de instrucciones, al eliminar la primera se enviará el fallo de página que la obtendrá de vuelta lo más lejos posible en el futuro. Al igual que las personas, las computadoras tratan de posponer los eventos indeseables el mayor tiempo posible.

El algoritmo de reemplazo de páginas óptimo

El único problema con este algoritmo es que no se puede realizar. Al momento del fallo de página, el sistema operativo no tiene forma de saber cuándo será la próxima referencia a cada una de las páginas. Aún así, al ejecutar un programa en un simulador y llevar la cuenta de todas las referencias a páginas, es posible implementar un algoritmo óptimo de reemplazo de página en la segunda corrida, al utilizar la información de referencia de páginas recolectada durante la primera corrida.

De esta manera, se puede comparar el rendimiento de los algoritmos realizables con el mejor posible. Si un sistema operativo logra un rendimiento de, por ejemplo, sólo 1 por ciento peor que el algoritmo óptimo, el esfuerzo invertido en buscar un mejor algoritmo producirá cuando mucho una mejora del 1 por ciento.

El algoritmo de reemplazo de páginas: no usadas recientemente

Para permitir que el sistema operativo recolecte estadísticas útiles sobre el uso de páginas, la mayor parte de las computadoras con memoria virtual tienen dos bits de estado asociados a cada página. R se establece cada vez que se hace referencia a la página (lectura o escritura); M se establece cuando se escribe en la página (es decir, se modifica). Los bits están contenidos en cada entrada de la tabla de páginas. Es importante tener en cuenta que estos bits se deben actualizar en cada referencia a la memoria, por lo que es imprescindible que se establezcan mediante el hardware. Si el hardware no tiene estos bits, pueden simularse.

Cuando se inicia un proceso, ambos bits de página para todas sus páginas se establecen en 0 mediante el sistema operativo. El bit R se borra en forma periódica (en cada interrupción de reloj) para diferenciar las páginas a las que no se ha hecho referencia recientemente de las que si se han referenciado.

El algoritmo de reemplazo de páginas: no usadas recientemente

Cuando ocurre un fallo de página, el sistema operativo inspecciona todas las páginas y las divide en 4 categorías con base en los valores actuales de sus bits R y M:

- Clase 0: no ha sido referenciada, no ha sido modificada.
- Clase 1: no ha sido referenciada, ha sido modificada.
- Clase 2: ha sido referenciada, no ha sido modificada.
- Clase 3: ha sido referenciada, ha sido modificada.

Aunque las páginas de la clase 1 parecen a primera instancia imposibles, ocurren cuando una interrupción de reloj borra el bit R de una página de la clase 3. Las interrupciones de reloj no borran el bit M debido a que esta información se necesita para saber si la página se ha vuelto a escribir en el disco o no. Al borrar R pero no M se obtiene una página de clase 1.

El algoritmo de reemplazo de páginas: no usadas recientemente

El algoritmo NRU (Not Recently Used, No usada recientemente) elimina una página al azar de la clase de menor numeración que no esté vacía. En este algoritmo está implícita la idea de que es mejor eliminar una página modificada a la que no se haya hecho referencia en al menos un pulso de reloj (por lo general, unos 20 mseg) que una página limpia de uso frecuente. La principal atracción del NRU es que es fácil de comprender, moderadamente eficiente de implementar y proporciona un rendimiento que, aunque no es óptimo, puede ser adecuado.

El algoritmo de reemplazo de páginas: FIFO

Otro algoritmo de paginación con baja sobrecarga es el de Primera en entrar, primera en salir (First-In, First-Out, FIFO).

El sistema operativo mantiene una lista de todas las páginas actualmente en memoria, en donde la llegada más reciente está en la parte final y la menos reciente en la parte frontal. En un fallo de página, se elimina la página que está en la parte frontal y la nueva página se agrega a la parte final de la lista. Pero es raro que se utilice FIFO en su forma pura, ya que podría generar sobrecargas innecesarias.

El algoritmo de reemplazo de páginas: segunda oportunidad

Una modificación simple al algoritmo FIFO que evita el problema de descartar una página de uso frecuente es inspeccionar el bit R de la página más antigua. Si es 0, la página es antigua y no se ha utilizado, por lo que se sustituye de inmediato. Si el bit R es 1, el bit se borra, la página se pone al final de la lista de páginas y su tiempo de carga se actualiza, como si acabara de llegar a la memoria. Después la búsqueda continúa.

La operación de este algoritmo, conocido como segunda oportunidad, vemos que las páginas se mantienen en una lista ligada y se ordenan con base en el tiempo en que llegaron a la memoria.

Lo que el algoritmo de segunda oportunidad está buscando es una página antigua a la que no se haya hecho referencia en el intervalo de reloj más reciente. Si se ha hecho referencia a todas las páginas, el algoritmo segunda oportunidad se degenera y se convierte en FIFO puro.

El algoritmo de reemplazo de páginas: reloj

Aunque el algoritmo segunda oportunidad es razonable, también es innecesariamente ineficiente debido a que está moviendo constantemente páginas en su lista. Un mejor método sería mantener todos los marcos de página en una lista circular en forma de reloj. La manecilla apunta a la página más antigua.

Cuando ocurre un fallo de página, la página a la que apunta la manecilla se inspecciona. Si el bit R es 0, la página se desaloja, se inserta la nueva página en el reloj en su lugar y la manecilla se avanza una posición. Si R es 1, se borra y la manecilla se avanza a la siguiente página. Este proceso se repite hasta encontrar una página con $R = 0$. No es de sorprender que a este algoritmo se le llame en reloj.

El algoritmo de reemplazo de páginas: LRU

Una buena aproximación al algoritmo óptimo se basa en la observación de que las páginas que se hayan utilizado con frecuencia en las últimas instrucciones, tal vez se volverán a usar con frecuencia en las siguientes.

Por el contrario, las páginas que no se hayan utilizado por mucho tiempo probablemente seguirán sin utilizarse por mucho tiempo más. Esta idea sugiere un algoritmo factible: cuando ocurra un fallo de página, hay que descartar la página que no se haya utilizado durante la mayor longitud de tiempo. A esta estrategia se le conoce como paginación LRU (Least Recently Used, Menos usadas recientemente).

Aunque en teoría el algoritmo LRU es realizable, no es barato. Para implementar el LRU por completo, es necesario mantener una lista enlazada de todas las páginas en memoria, con la página de uso más reciente en la parte frontal y la de uso menos reciente en la parte final. La dificultad es que la lista debe actualizarse en cada referencia a memoria.

Simulación de LRU en software

Aunque los dos algoritmos LRU anteriores son (en principio) realizables, muy pocas máquinas (si acaso) tienen el hardware requerido. En vez de ello, se necesita una solución que puede implementarse en software. Una posibilidad es el algoritmo NFU (Not Frequently Used, No utilizadas frecuentemente). Este algoritmo requiere un contador de software asociado con cada página, que al principio es cero. En cada interrupción de reloj, el sistema operativo explora todas las páginas en memoria. Para cada página se agrega el bit R, que es 0 o 1, al contador. Los contadores llevan la cuenta aproximada de la frecuencia con que se hace referencia a cada página. Cuando ocurre un fallo de página, se selecciona la página con el contador que sea menor para sustituirla.

Simulación de LRU en software

El principal problema con el algoritmo NFU es que nunca olvida nada. Por ejemplo, en un compilador con varias pasadas, las páginas que se utilizaron con frecuencia durante la pasada 1 podrían tener todavía una cuenta alta en las siguientes pasadas. De hecho, si la pasada 1 tiene el tiempo de ejecución más largo de todas las pasadas, las páginas que contienen el código para las pasadas subsiguientes pueden tener siempre cuentas menores que las páginas de la pasada 1. En consecuencia, el sistema operativo eliminará páginas útiles, en vez de páginas que ya no se hayan utilizado.

Por fortuna, una pequeña modificación al algoritmo NFU le permite simular el algoritmo LRU muy bien. La modificación consta de dos partes. Primero, cada uno de los contadores se desplaza a la derecha 1 bit antes de agregar el bit R. Después, el bit R se agrega al bit de más a la izquierda, en lugar de sumarse al bit de más a la derecha. Este algoritmo es conocido como envejecimiento (aging).

El algoritmo de reemplazo de páginas: conjunto de trabajo

Muchos sistemas de paginación tratan de llevar la cuenta del conjunto de trabajo de cada proceso y se aseguran que esté en memoria antes de permitir que el proceso se ejecute. Este método se conoce como modelo del conjunto de trabajo (Denning, 1970). Está diseñado para reducir en gran parte la proporción de fallos de página. Al proceso de cargar las páginas antes de permitir que se ejecuten los procesos también se le conoce como prepaginación. Tenga en cuenta que el conjunto de trabajo cambia con el tiempo

El algoritmo de reemplazo de páginas WSClock

El algoritmo básico del conjunto de trabajo es incómodo ya que exige explorar toda la tabla de páginas en cada fallo de página hasta localizar un candidato adecuado. Un algoritmo mejorado, basado en el algoritmo de reloj pero que también utiliza la información del conjunto de trabajo, se conoce como WSClock (Carr y Hennessey, 1981). Debido a su simplicidad de implementación y buen rendimiento, es muy utilizado en la práctica.

La estructura de datos necesaria es una lista circular de marcos de página, como en el algoritmo de reloj. Al principio, esta lista está vacía. Cuando se carga la primera página, se agrega a la lista. A medida que se agregan más páginas, pasan a la lista para formar un anillo. Cada entrada contiene el campo Tiempo de último uso del algoritmo básico del conjunto de trabajo, así como el bit R (mostrado) y el bit M (no mostrado).

Resumen de los algoritmos de reemplazo de páginas

Algoritmo	Comentario
Óptimo	No se puede implementar, pero es útil como punto de comparación
NRU (No usadas recientemente)	Una aproximación muy burda del LRU
FIFO (primera en entrar, primera en salir)	Podría descartar páginas importantes
Segunda oportunidad	Gran mejora sobre FIFO
Reloj	Realista
LRU (menos usadas recientemente)	Excelente, pero difícil de implementar con exactitud
NFU (no utilizadas frecuentemente)	Aproximación a LRU bastante burda
Envejecimiento	Algoritmo eficiente que se aproxima bien a LRU
Conjunto de trabajo	Muy costoso de implementar
WSClock	Algoritmo eficientemente bueno

Bibliografía

- Tanenbaum, A. S. (2009). Sistemas operativos modernos (3a ed.) (pp. 1-18). México: Pearson Educación.
- Stallings, W. (2005). Sistemas operativos (5a ed.) (pp. 54-67). Madrid: Pearson Educación.

