Gestión de la Memoria

JOAQUÍN ROIZ PAGADOR[1]

[1] *Universidad Pablo de Olavide, Ctra. Utrera 1, 41013, Dos Hermanas, Sevilla, España.*
[*] *Contacto: quiniroiz@gmail.com*

Compiled June 1, 2018

Resumen del temario de la asignatura de Sistemas Operativos del Grado en Ingeniería Informática en Sistemas de Información. © 2018 Joaquín Roiz Pagador

OCIS codes: (130.6750) Systems.

quiniroiz@gmail.com

1. ADMINISTRACIÓN DE MEMORIA

A través de los años se ha elaborado el concepto de **jerarquía de memoria**, de acuerdo con el cual, las computadoras tienen unos cuantos megabytes de memoria caché, muy rápida, costosa y volátil, unos cuantos gigabytes de memoria principal, de mediana velocidad, a precio mediano y volátil, unos cuantos terabytes de almacenamiento en disco lento, económico y no volátil, y el almacenamiento removible, omo los DVDs y las memorias USB.

La parte del sistema operativo que administra la jerarquía de memoria se conoce como **administrador de memoria**.

A. Sin abstracción de memoria

Cuando el sistema se organiza de esta forma, se puede ejecutar sólo un proceso a la vez. Tan pronto como el usuario teclea un comando, el sistema operativo copia el programa solicitado del disco a la memoria y lo ejecuta. Cuando termina el proceso, el sistema operativo muestra un carácter indicador de comando y espera un nuevo comando. Cuando recibe el comando, carga un nuevo programa en memoria, sobrescribiendo el primero.

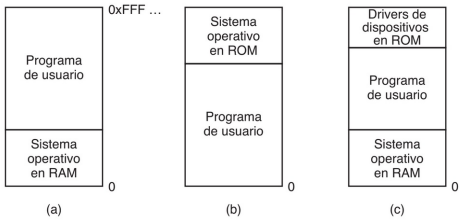


Fig. 1. Tres formas simples de organizar la memoria con un sistema operativo y un proceso de usuario. También existen otras posibilidades.

Ejecución de múltiple programas sin abstracción de memoria

Con la adición de cierto hardware especial es posible ejecutar múltiples programas concurrentemente, aun sin intercambio. Los primeros modelos de IBM 360 resolvieron el problema de la siguiente manera: la memoria estaba dividida en bloques de 2KB y a cada uno se le asignaba una llave de protección de 4 bits, guardad en registros especiales dentro de la CPU. Un equipo con una memoria de 1 MB sólo necesitaba 512 de estos registros de 4 bits para totalizar 256 bytes de almacenamiento de la llave. El registro **PSW**(*Program Status Word,* Palabra de estado del programa) también contenía una llave de 4 bits. El hardware de la 360 controlaba mediante un trap cualquier intento por parte de un proceso en ejecución de acceder a la memoria con un código de protección distinto del de la llave de. PSW. Como sólo el sistema operativo podía modificar las llaves de protección, los procesos de usuario fueron controlados para que no interfirieran unos con otros, ni con el mismo sistema operativo.

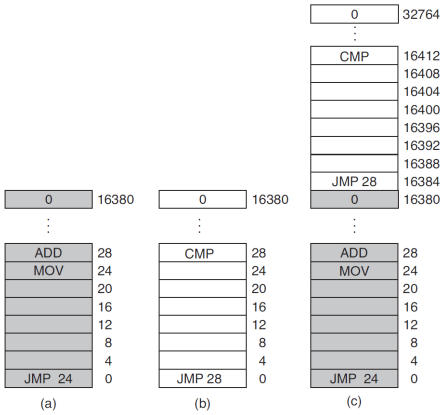


Fig. 2. Ilustración del problema de reubicación. (a) Un programa de 16 KB. (b) Otro programa de 16 KB. (c) Los dos programas cargados consecutivamente en la memoria.

Sin embargo,esta solución tendía una gran desventaja, que se ilustra en la figura 2. El problema central es que dos programas hacen referencia a la memoria física absoluta.

Lo que la IBM 360 hacía como solución para salir del paso era modificar el segundo programa al instante a medida que se cargaba en la memoria, usando una técnica conocida como **reubicación estática**. Cuando se cargaba un programa en la dirección 16.384, se sumaba el valor constante 16.384 a todas las direcciones del programa durante el proceso de carga. Aunque este mecanismo funciona si se lleva a cabo en la forma correcta,

no es una solución muy general y reduce la velocidad de carga.

2. UNA ABSTRACCIÓN DE MEMORIA: ESPACIOS DE DIRECCIONES

A. Intercambio

La estrategia más simple, conocida como **intercambio**, consiste en llevar cada proceso completo a memoria, ejecutarlo durante cierto tiempo y después regresarlo al disco. La otra estrategia, conocida como **memoria virtual**, permite que los programas se ejecuten incluso cuando sólo se encuentran en forma parcial en la memoria.

En la figura 3, al principio, sólo el proceso A está en la memoria.

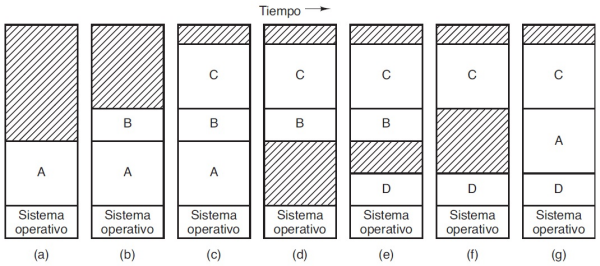


Fig. 3. La asignación de la memoria cambia a medida que llegan procesos a la memoria y salen de ésta. Las regiones sombreadas son la memoria sin usar.

Cuando el intercambio crea varios huecos en la memoria, es posible combinarlos todos en uno grande desplazando los procesos lo más hacia abajo que sea posible. Esta técnica se conoce como **compatación de memoria**.

AL intercambiar procesos al disco, se debe intercambiar sólo la memoria que se encuentre en uso; es un desperdicio intercambiar también la memoria adicional.

Si los procesos pueden tener dos segmentos en crecimiento, por ejemplo, cuando el segmento de datos se utiliza como heap para las variables que se asignan y liberan en forma dinámica y un segmento de pila para las variables locales normales y las direcciones de retorno, un arreglo alternativo se sugiere por sí mismo. Cada proceso ilustrado tiene una pila en la parte superior de su memoria asignada, la cual está creciendo hacia abajo, y un segmento de datos justo debajo del texto del programa, que está creciendo hacia arriba. La memoria entre estos segmentos se puede utilizar para cualquiera de los dos. Si se agota, el proceso tendrá que moverse a un hueco con suficiente espacio, intercambiarse fuera de la memoria hasta que se pueda crear un hueco lo suficientemente grande, o eliminarse.

3. ADMINISTRACIÓN DE MEMORIA LIBRE

Administración de memoria con mapas de bits

Con un mapa de bits, la memoria se divide en unidades de asignación tan pequeñas como unas cuantas palabras y ttan grandes como varios kilobytes. Para cada unidad de asignación hay un bit correspondiente en el mapa de bits, que es 0 si la unidad está libre y 1 si está ocupada.

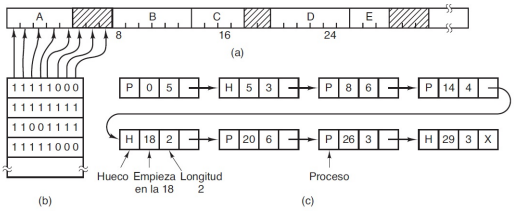


Fig. 4. (a) Una parte de la memoria son cinco procesos y tres huecos. Las marcas de graduación muestran las unidades de asignación de memoria. Las regiones sombreadas (0 en el mapa de bits) están libres. (b) El mapa de bits correspondiente. (c) La misma información en forma de lista.

El tamaño de la unidad de asignación es una importante cuestión de diseño. Entre más pequeña sea la unidad de asignación, mayor será el mapa de bits. El problema principal es que, cuando se ha decidido llevar un proceso de k unidades a la memoria, el administrador de la memoria debe buscar en el mapa para encontrar una serie de k bits consecutivos con el valor 0 en el mapa de bits. El proceso de buscar en un mapa de bits una serie de cierta longitud es una operación lenta; éste es un argumento contra los mapas de bits.

Administración de memoria con listas ligadas

La lista de segmentos se mantiene ordenada por dirección. Al ordenarla de esta manera, tenemos la ventaja de que cuando termina un proceso o se intercambia, el proceso de actualizar la lista es simple.

Como la ranura de la tabla de procesos para el proceso en terminación en general apuntará a la entrada en la lista para el mismo proceso, puede ser más conveniente tener la lista como una lista doblemente ligada. Esta estructura facilita encontrar la entrada anterior y ver si es posible una combinación.

Algoritmos usados

- Algoritmo del **primer ajuste**: el administrador de memoria explora la lista de segmentos hasta encontrar un hueco que sea lo bastante grande. Después el hueco se divide en dos partes, una para el proceso y otra para la memoria sin utilizar, excepto en el estadísticamente improbable caso de un ajuste exacto.
- Algoritmo del **siguiente ajuste**: Funciona de la misma manera que el primer ajuste, excepto porque lleva un registro de dónde se encuentra cada vez que descubre un hueco adecuado. La siguiente vez que es llamado para buscar un hueco, empieza a buscar en la lista desde el lugar en el que se quedó la última vez, en vez de empezar siempre desde el principio, como el algoritmo del primer ajuste.
- Algoritmo del **mejor ajuste**: Este algoritmo busca en toda la lista, de principio a fin y toma el hueco más pequeño que sea adecuado. En vez de dividir un gran hueco que podría necesitarse después, el algoritmo del mejor ajuste trata de buscar un hueco que esté cerca del tamaño actual necesario, que coincida mejor con la solicitud y los huecos disponibles. Es más lento que el del primer ajuste, provoca más desperdicio de memoria que los algoritmos del primer ajuste o del siguiente ajuste. Genera huecos más grandes en promedio.

- Algoritmo del **peor ajuste**: tomar siempre el hueco más grande disponible, de manera que el nuevo hueco sea lo bastante grande como para ser útil. El precio inevitable que se paga por esta aceleración en la asignación es la complejidad adicional y la lentitud al asignar la memoria, ya que un segmento liberado tiene que eliminarse de la lista de procesos e insertarse en la lista de huecos.

- Algoritmo de **ajuste rápido**: mantiene listas separadas para algunos de los tamaños más comunes solicitados. Buscar un hueco del tamaño requerido es extremadamente rápido, pero tiene la misma desventaja que todos los esquemas que se ordenan por el tamaño del hueco: cuando un proceso termina o es intercambiado, buscar en sus vecino para ver si es posible una fusión es un proceso costoso.

4. MEMORIA VIRTUAL

Existe la necesidad de ejecutar programas que son demasiado grandes como para caber en memoria. 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.

El problema de que los programas sean más grandes que la memoria ha estado presente desde los inicios de la computación. Una solución que se adoptó en la década de 1960 fue dividir los programas en pequeñas partes, conocidas como **sobrepuestos(*overlays*)**.

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. La memoria virtual es una generacización de la idea de los registros base y límite.

A. Paginación

En cualquier computadora, los programas hacen referencia a un conjunto de direcciones de memoria. Cuando un programa ejecuta una instrucción como

MOV REG, 100

lo hace para copiar el contenido de la dirección de memoria 100 a REG. Las direcciones se pueden generar usando indexado, registros base, registros de segmentos y otras formas más.

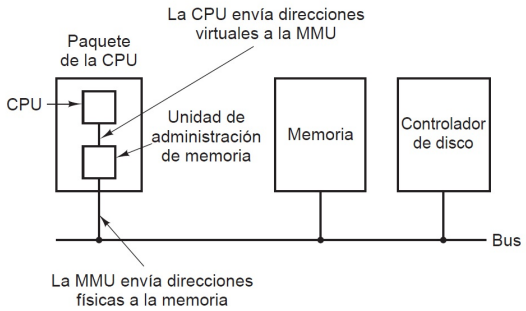


Fig. 5. La posición y función de la MMU. Aquí la MMU se muestra como parte del chip de CPU, debido a que es común esta configuración en la actualidad. Sin embargo, lógicamente podría ser un chip separado y lo era hace años.

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 se 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 Managemente Unit*, Unidad de administración de memoria) que asocia las direcciones virtuales a las direcciones de memoria física.

En la figura **??** tenemos una computadora que genera direcciones de 16 bits, desde 0 hasta 64k. 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. Las transferencias entre RAM y el disco siempre son en páginas completas.

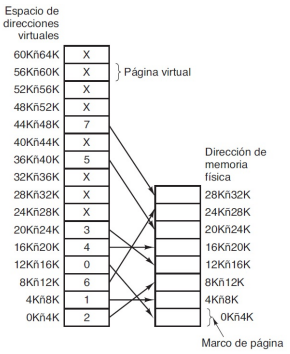


Fig. 6. La relación entre las direcciones vrituales y las direcciones de memoria física está dada por la tabla de páginas. Cada página empieza en un múltiplo de 4096 y termina 4095 direcciones más arriba, por lo que 4 K a 8 K en realidad significa de 4096 a 8191 y de 8 K a 12 K significa de 8192 a 12287.

El rango marcado de 0K a 4K significa que las direcciones virtuales o físicas en esa página son de 0 a 4095. El rango de 4K a 8K se refiere a direcciones de 4096 a 8191 y así en lo sucesivo. Cada página contiene exactamente 4096 direcciones que empiezan en un múltiplo de 4096 y terminan uno antes del múltiplo de 4096.

¿Qué ocurre si el programa hace referencia a direcciones no asiociadas, por ejemplo, mediante el uso de la instrucción

MOV REG, 32780

que es el byte 12 dentro de la página virtual 8? La MMU detecta que la página no está asociada y hace que la CPU haga un trap al sistema opeartivo. A este trap se le llama **fallo de página**. El sistema operativo selecciona un marco de página que se utilice poco y escribe su contenido de vuelta al disco. Después obtiene la página que se acaba de referenciar en el marco de página que se acaba de liberar, cambia la asociación y reinicia la instrucción que originó el trap. Si el sistema operativo decidiera desalojar el marco de página 1, cargaría la página virtual 8 en la dirección física 8192 y realizaría dos cambios en la asociación de la MMU. Primero, marcaría la entrada de la página virtual 1 como no asociada, para hacer un trap por cualquier acceso a las direcciones virtuales entre 4096 y 8191. Después reemplazaría la cruz en la entrada de la página virtual 8 con un 1, de manera que al ejecutar la instrucción que originó el trap, asocie la dirección virtual 32780 a la dirección física 4108.

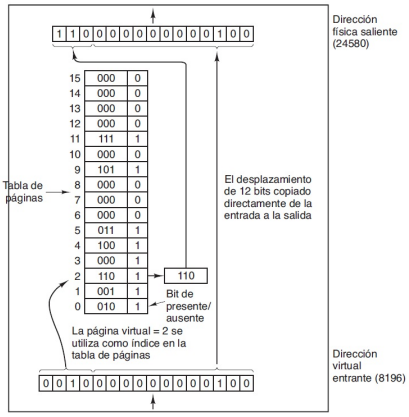


Fig. 7. La operación interna de la MMU con 16 páginas de 4 KB.

El número de página se utiliza como índice en la **tabla de páginas**, conduciendo al número del marco de página que corresponde a esa página virtual. Si el bit de *presente/ausente* es 0, se provoca un trap al sistema operativo. Si el bit es 1, el número del marco de página encontrado en la tabla de páginas se copia a los 3 bits de mayor orden del registro de salida, junto con el desplazamiento de 12 bits, que se copia sin modificación de la dirección virtual entrante. En conjunto forman una dirección física de 15 bits. Después, el registro de salida se coloca en el bus de memoria como la dirección de memoria fíica.

B. 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 y en un desplazamiento. 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 dentro de la página seleccionada. También es osible una división con 3, 5 u otro número de bits para la página. Las distintas divisiones implican diferentes tamaños de página.

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 puede enviar a la memoria.

El propósito de la tabla de páginas es asociar páginas virtuales a los marcos de página. 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 una entrada en tabla de páginas

La distribución exacta de una entrada depende en gran parte de la otra máquina, pero el tipo de información presente es aproximadamente el mismo de una máquina a otra.

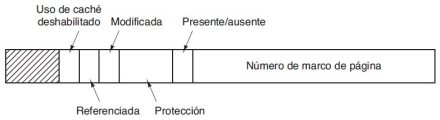


Fig. 8. Una típica entrada en la tabla de páginas.

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, debe escribirse de vuelta en el disco. Si no se ha modificado 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.

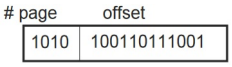
El bit de *referencia* se establece cada vez que una página es referenciada, ya sea para leer o escribir. Su función es ayudar al sistema operativo a elegir una página para desalojarla cuando ocurre un fallo de página. Las páginas que no se estén utilizando son mejores candidatos que las páginas que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

Bits Bits Bits

- Tamaño:
  - Memoria virtuales de 2^m.
  - Páginas de 2^n.
- Las direcciones lógicas estarán formadas por **m** bits:
  - Los primeros **m-n** bits representan el número de página.
  - Los **n** últimos representan el desplazamiento.

Ejemplo:

- Memoria virtual de 64 KB = 2^{16}
- Tamaño de página 4KB = 2^{12}
- m=16, n=12
- 4 bits de mayor tamaño son usados para el número de página.
- Los últimos 12 bits (n=12) representan el desplazamiento.



Tablas de páginas multinivel

Para solucionar el problema de tener que almacenar enormes tablas de páginas en la memoria todo el tiempo, muchas computadoras utilizan una tabla de páginas multinivel.

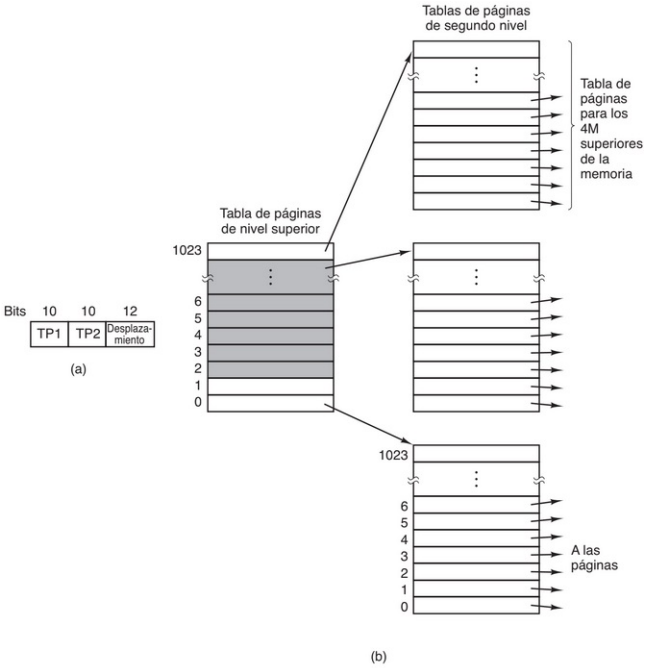


Fig. 9. a) Direccción de 32 bits con dos campos de tabla de páginas. b) Tablas de páginas de dos niveles.

El secreto del método de tablas de páginas multinivel radica en que no es necesario tener todas las tablas de páginas en la memoria todo el tiempo. En particular, las que no se necesiten no se deberán tenerse en ella.

C. Estructura de una entrada de tabla de páginas

La organiación exacta de una entrada depende mucho de la máquina, pero el tipo de información presente es casi el mismo en todas las computadoras. El campo más importante es el Número de marco de página. Después de todo, el objetivo de la correspondencia de páginas es averiguar este valor. Junto a él tenemos el bit presente/ausente. Si este bit es 1, la entrada es válida y puede usarse; si es 0, la página virtual a la que la entrada corresponde no está en memoria. Tener acceso a una entrada de tabla de página que tiene ese bit establecido a 0 causa un fallo de página.

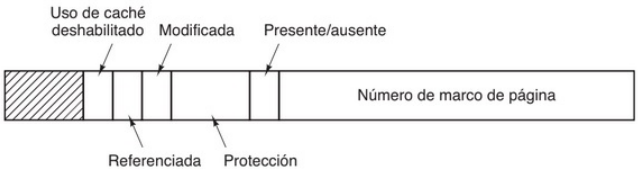


Fig. 10. Entrada típica de tabla de páginas.

Los bits Modificada y SOlicitada llevan el control del uso de la página. Cuando se escribe en una página, el hardware enciende de forma automática el bit Modificada. El bit también se conoce como **bit modificado**, pues refleja el estado de la página.

Búferes de traducción adelantada

El punto inicial de la mayor parte de las técnicas de optimización es que la tabla de pátinas está en la memoria. Este diseño tiene un enorme impacto sobre el rendimiento. Sólo se lee con mucha frecuencia una pequeña fracción de las entradas en la tabla de páginas; el resto se utiliza muy pocas veces.

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**. 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 establece cuando se modifica la página, el código de protección y el marco de página físico en el que se encuentra la página.

Los bits de Protección indican cuáles tipos de acceso están permitidos. En su forma más simple, este campo contiene un bit, que es 0 si se permite leer y escribir, y 1 si sólo se permite leer. Un esquema más avanzado usa 3 bits para habilitar la lectura, escritura y ejecución de la página, respectivamente.

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 W	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Fig. 11. Un TLB para acelerar la paginación

Tablas de páginas invertidas

Las tablas de páginas tradicionales del tipo que hemos descrito hasta ahora requieren una entrada por cada página virtual. En los sistemas es probable que sí se pueda manejar una tabla tan grande. Sin embargo, a medida que aumenta el número de computadoras de 64 bits, la situación cambia de manera drástica. Si cada entrada requiere 8 bytes, la tabla ocupará más de 30 millonnes de gigabytes.

Una solución es la **tabla de páginas invertida**. En este diseño, hay una entrada por cada marco de página en la memoria real, en lugar de una entrada por página del espacio de direcciones virtual. La entrada indica qué proceso y qué página virtual está en el marco correspondiente.

Con esto ahorramos un montón de espacio, pero el mapeo es mucho más complejo.

Cuando el proceso n referencia a la página virtual p, el hardware no puede encontrar la página física usando p como un índice en la tabla. En su lugar, deberá buscar en la tabla invertida una entrada (n,p). Además, esta búsqueda deberá haber acabado en cada referencia de memoria, no en fallos de página. La búsqueda relentizará el sistema. La solución a este dilema es usar el TLB (buffer de consulta para traducción).

5. ALGORITMOS DE REEMPLAZO DE PÁGINA

Cuando se presenta un fallo de página, el sistema operativo tiene que escoger la página que desalojará de la memoria para hacer espacio para colocar la página que traerá del disco. Si la página a desalojar ha sido modificada mientras estaba en la memoria, deberá reescribirse en el disco para actualizar la copia. Posteriormente se cargará la página solicitada.

Objetivos: Minimizar el número de fallos de página. Para ello será necesario utilizar algún algoritmo para el reemplazamiento de página.

A. El algoritmo óptimo de reemplazo de páginas

El mejor algoritmo de reemplazo de páginas posible es fácil de describir pero imposible de implementar. En el momento en el que se presenta un fallo de página, cierto conjunto de páginas está en memoria. En la siguiente instrucción se hará referencia a una de esas páginas. Puede ser que no se haa referencia a otras de las páginas sino hasta 10, 100 o quizá 000 instrucciones después.

El algoritmo de página óptima simplemente dice que debe desalojarse la página con el rótulo más grande. Si faltan ocho millones de instrucciones para que se use cierta página y faltan seis millones de iinstrucciones para que se use otra, el desalojo de la primera aplaza lo más posible el fallo de página que volverá a traer una página a la memoria. Las computadoras, igual que las personas, tratan de aplazar lo más posible los sucesos desagradables.

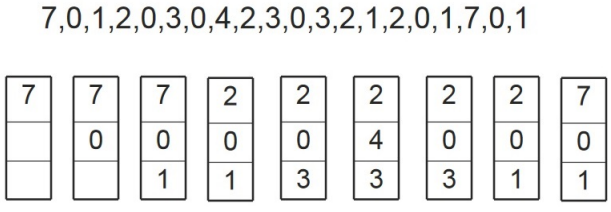


Fig. 12. Ejemplo de ejecución del algoritmo de página óptima.

B. El algoritmo de reemplazo de páginas no usadas recientemente (NRU)

Para que el sistema operativo pueda recabar estadísticas útiles acerca de cuáles páginas se están usando y cuáles no, casi todas las computadoras con memoria virtual asocial a cada página dos bits de estado.

- R se enciende (a 1) cuando se hace referencia a la página (lectura o escritura).
- M se enciende cada vez que se escribe en la página.

Los bits están incluídos en la entrada correspondiente de la tabla de páginas. Es importante tener en cuenta que dichos bits deben actualizarse en cada referencia da la memoria y, es indispensable que sea el hardware el que los encienda. Una vez que un bit haya sido encendido, conservará el valor 1 hasa que el sistema operativo lo restablezca a 0.

El valor inicial de R y M para cada página será 0. Periódicamente, el bit de R es reseteado, para distinguir páginas que no han sido referenciadas recientemente de otras que sí lo han sido.

Cuando un fallo de página ocurre, el sistem aoperativo inspecciona todas las páginas y las divide en 4 categorías basadas en los actuales valores de sus bits R y M:

- **Clase 0:** no solicitada, no modificada. (R=0,M=0)
- **Clase 1:** no referenciada, modificada. (R=0,M=1)
- **Clase 2:** referenciada, no modificada. (R=1,M=0)
- **Clase 3:** referenciada, modificada. (R=1,M=1)

El algoritmo **no usada recientemente (NRU**; not recently used) desaloja 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 desalojar una página modificada a la que no se ha hecho referencia en por lo menos un tic del 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 produce un desempeño que, si bien de ninguna manera es óptimo, podría ser aceptable.

C. Algoritmo FIFO

Primero en entrar, primero en salir. El sistema operativo mantiene una lista de todas las páginas actualmente en memoria. La más reciente en llegar en la cola y la menos reciente en llegar en la cabeza. En un fallo de página, la página en la cabeza es eliminada y la nueva página añadida a la cola de la lista. El problema de este algoritmo es que se pierden en el olvido posiblemente nuevas páginas que se usarán frecuentemente.

Fig. 13. Ejemplo de ejecución del algoritmo FIFO.

D. Algoritmo de la segunda oportunidad

La modificación de una FIFO evita el problema de tirar una página muy usada. Después de eliminar la página más vieja, inspecciona el bit R en la página. Si es 0, la página es suficientemente vieja y no usada, así que se remplaza inmediatamente. Pero, '?y si R es 1? Si el bit R es 1, el se resetea a 0. La página se pone al final de la lista de páginas. Su tiempo de carga se actualiza conforme van llegando en memoria. A partir de este punto continúa la búsqueda.

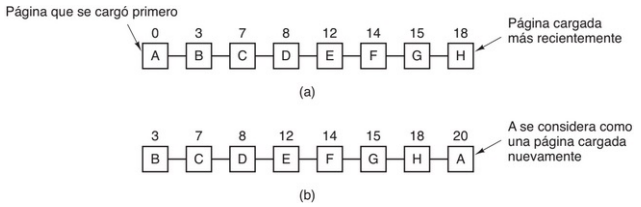


Fig. 14. Operación del algoritmo de segunda oportunidad. a) Páginas en orden FIFO. b) Lista de páginas si hay un fallo de página en tiempo 20 y el bit R de la página A está encendido. Los números arriba de las páginas son sus horas de carga.

E. El algoritmo de reemplazo de páginas tipo reloj

Aunque el algoritmo de segunda oportunidad es razonable, es menos eficiente de lo que se desearía porque en forma continua cambia páginas de lugar dentro de su lista. Una mejor estrategia sería mantener todas las páginas en una lista circular parecida a un reloj. Una manecilla apunta a la página más antigua.

Cuando se representa un fallo de página, se examina la página a la que apunta la manecilla. Si su bit R es 0, dicha se desaloja, la nueva se inserta en su lugar y la manecilla se adelanta una posición. 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 sorpresa que a este algoritmo se le llame reloj. La única diferencia respecto al de segunda oporunidad radica en la implementación.

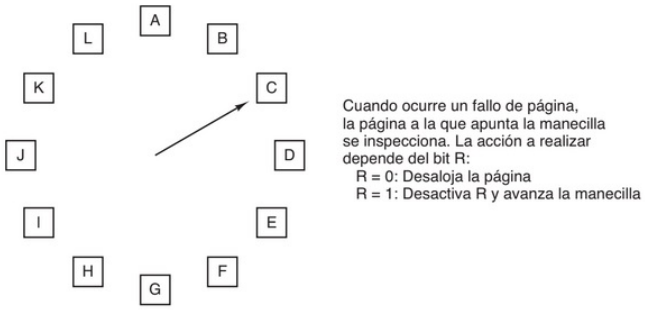


Fig. 15. El algoritmo de reemplazo de páginas tipo reloj.

F. Algoritmo de Reemplazo de Págiina menos recientemente usada (LRU)

Una buena aproximación al algoritmo óptimo se basa en la observación de que es probable que las páginas que se han usado mucho en las últimas instrucciones se usarán mucho otra vez en las siguientes. En cambio, es probable que las páginas que no se han usado en años seguirán si usarse mucho tiempo. Esta idea sugiere un algoritmo factible: cuando se presente un fallo de página, desalojar la que tiene más tiempo sin usarse. Tal estrategia se denomina paginación de **menos recientemente usada(LRU**;least recently used).

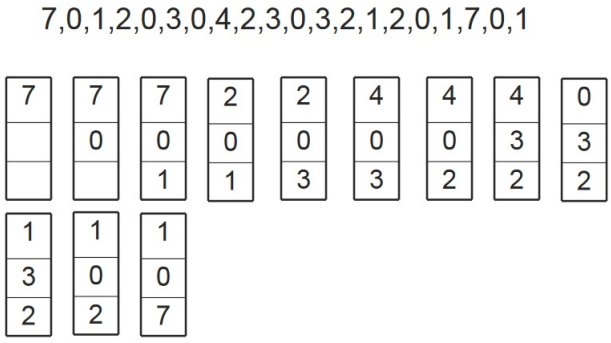


Fig. 16. Ejemplo de ejecución del algoritmoLRU.

REFERENCES

REFERENCES

1. Andrew S. Tanenbaum, Sistemas Operativos Modernos, Tercera Edición, Ed. Pearson (2009).
2. Silberschatz Galvin, Sistemas Operativos, Quinta Edición, Ed. Addison Wesley (1999).

SOBRE MI

Joaquín Roiz Pagador

Estudiante del Grado en Ingeniería Informática en Sistemas de Información en la universidad *Pablo de Olavide*, promoción de 2016.