ADMINISTRACIÓN DE MEMORIA

La parte del sistema operativo que administra (parte de) la jerarquía de memoria se conoce como administrador de memoria. Su trabajo es administrar la memoria con eficiencia: llevar el registro de cuáles partes de la memoria están en uso, asignar memoria a los procesos cuando la necesiten y desasignarla cuando terminen.

- SIN ABSTRACCIÓN DE MEMORIA

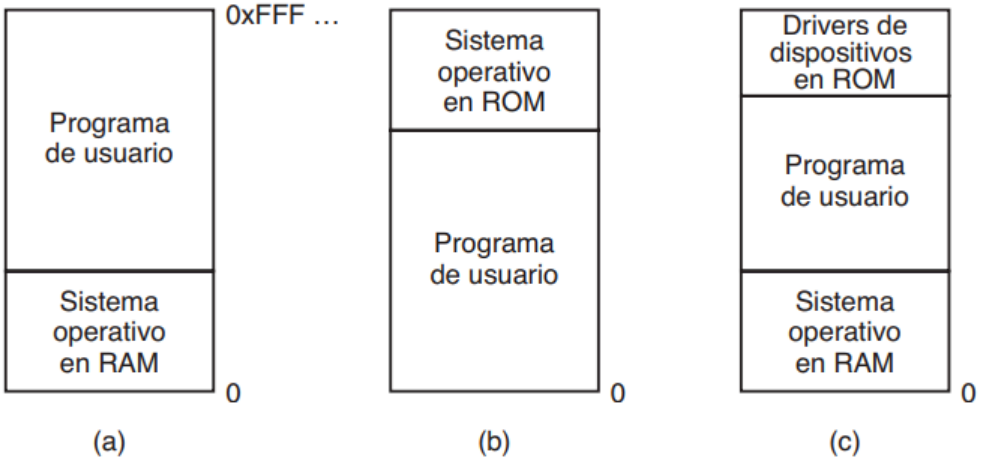


Figura 3-1. Tres formas simples de organizar la memoria con un sistema operativo y un proceso de usuario. También existen otras posibilidades.

El primer modelo se utilizó antes en las mainframes y minicomputadoras, pero actualmente casi no se emplea. El segundo modelo se utiliza en algunas computadoras de bolsillo y sistemas integrados. El tercer modelo fue utilizado por las primeras computadoras personales (por ejemplo, las que ejecutaban MS-DOS), donde la porción del sistema en la ROM se conoce como BIOS (Basic Input Output System, Sistema básico de entrada y salida). Los modelos (a) y (c) tienen la desventaja de que un error en el programa de usuario puede borrar el sistema operativo, posiblemente con resultados desastrosos (la información del disco podría quedar ininteligible).

1. Ejecución de múltiple programas sin una abstracción de memoria

    1.1. No obstante, aun sin abstracción de memoria es posible ejecutar varios programas al mismo tiempo. Lo que el sistema operativo debe hacer es guardar todo el contenido de la memoria en un archivo en disco, para después traer y ejecutar el siguiente programa. Mientras sólo haya un programa a la vez en la memoria no hay conflictos.

    1.2. El registro PSW (Program Status Word, Palabra de estado del programa) también contenía una llave de 4 bits.

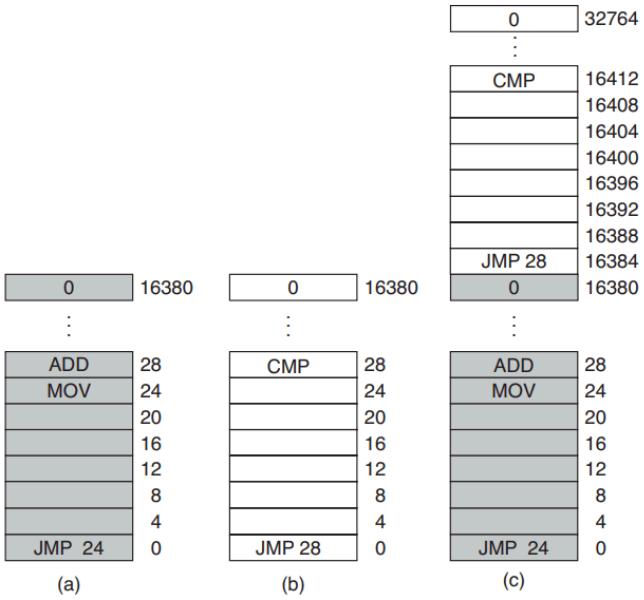


Figura 3-2. Ilustración del problema de reubicación. (a) Un programa de 16 KB. (b) Otro programa de 16 KB. (c) Los dos programas cargados consecutivamente en la memoria.

2. 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.

- UNA ABSTRACCIÓN DE MEMORIA: ESPACIOS DE DIRECCIONES

1. La noción de un espacio de direcciones

   Hay que resolver dos problemas para permitir que haya varias aplicaciones en memoria al mismo tiempo sin que interfieran entre sí: protección y reubicación.

   1.1. Una solución primitiva al primer problema en la IBM 360: etiquetar trozos de memoria con una llave de protección y comparar la llave del proceso en ejecución con la de cada palabra de memoria obtenida por la CPU. Sin embargo, este método por sí solo no resuelve el segundo problema, aunque se puede resolver mediante la reubicación de los programas al momento de cargarlos, pero ésta es una solución lenta y complicada.

   1.2. Un espacio de direcciones (address space) es el conjunto de direcciones que puede utilizar un proceso para direccionar la memoria. Cada proceso tiene su propio espacio de direcciones, independiente de los que pertenecen a otros procesos

   1.3. Registros base y límite

      1.3.1. La solución sencilla utiliza una versión muy simple de la reubicación dinámica. Lo que hace es asociar el espacio de direcciones de cada proceso sobre una parte distinta de la memoria física, de una manera

simple. Es equipar cada CPU con dos registros de hardware especiales, conocidos comúnmente como los registros base y límite.

1.3.2. Cuando se ejecuta un proceso, el registro base se carga con la dirección física donde empieza el programa en memoria y el registro límite se carga con la longitud del programa.

1.3.3. Una desventaja de la reubicación usando los registros base y límite es la necesidad de realizar una suma y una comparación en cada referencia a memoria. Las comparaciones se pueden hacer con rapidez, pero las sumas son lentas debido al tiempo de propagación del acarreo, a menos que se utilicen circuitos especiales de suma.

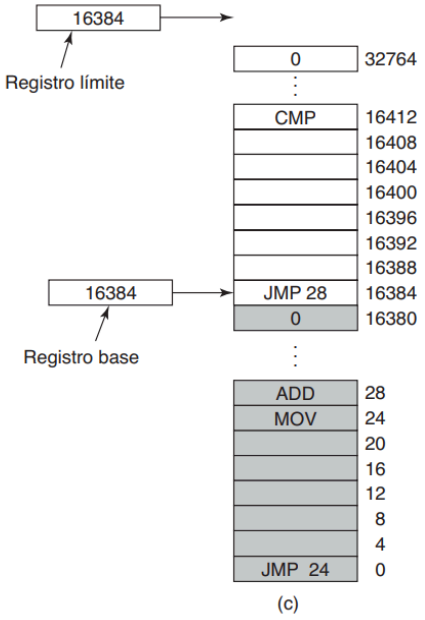


Figura 3-3. Se pueden utilizar registros base y límite para dar a cada proceso un espacio de direcciones separado.

2. Intercambio

2.1. En la práctica, la cantidad total de RAM que requieren todos los procesos es a menudo mucho mayor de lo que puede acomodarse en memoria.

2.2. La estrategia más simple, conocida como intercambio, consiste en llevar cada proceso completo a memoria, ejecutarlo durante cierto tiempo y después regresarlo al disco.

2.3. La otra estrategia, conocida como memoria virtual, permite que los programas se ejecuten incluso cuando sólo se encuentran en forma parcial en la memoria.

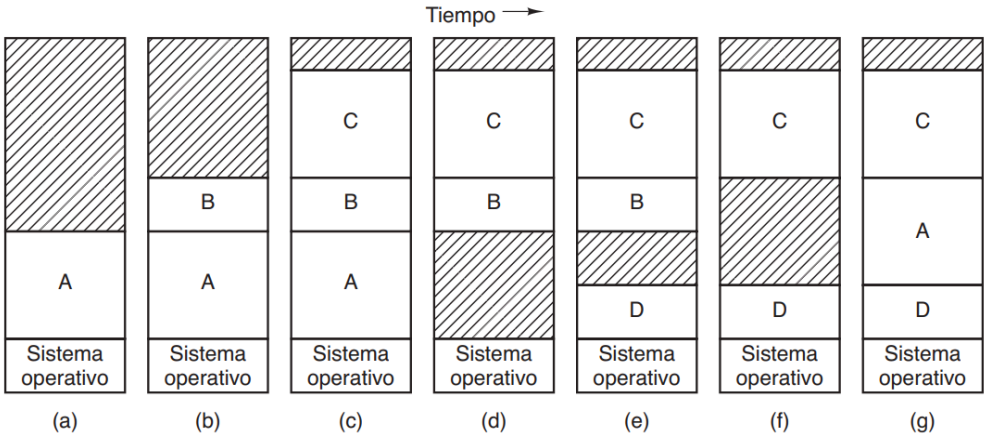


Figura 3-4. La asignación de la memoria cambia a medida que llegan procesos a la memoria y salen de ésta. Las regiones sombreadas son la memoria sin usar.

2.4. 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 compactación de memoria.

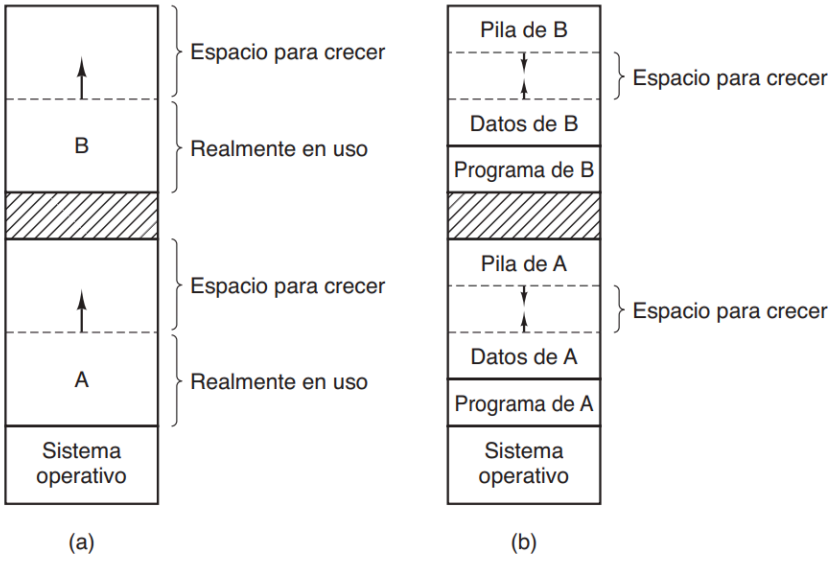


Figura 3-5. (a) Asignación de espacio para un segmento de datos en crecimiento. (b) Asignación de espacio para una pila en crecimiento y un segmento de datos en crecimiento.

3. Administración de memoria libre

3.1. Administración de memoria con mapas de bits

3.1.1. Con un mapa de bits, la memoria se divide en unidades de asignación tan pequeñas como unas cuantas palabras y tan 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

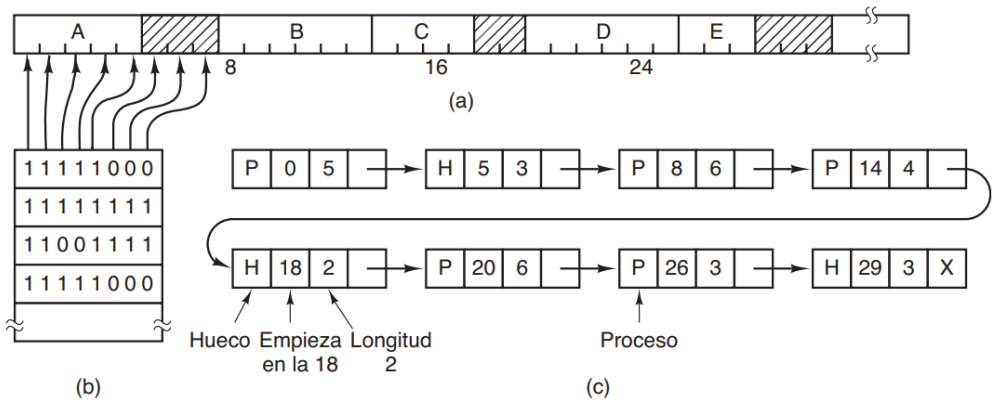


Figura 3-6. (a) Una parte de la memoria con cinco procesos y tres huecos. Las marcas de graduación muestran las unidades de asignación de memoria. Las regiones sombreadas (0 en el mapa de bits) están libres. (b) El mapa de bits correspondiente. (c) La misma información en forma de lista.

3.1.2. Un mapa de bits proporciona una manera simple de llevar el registro de las palabras de memoria en una cantidad fija de memoria, debido a que el tamaño del mapa de bits sólo depende del tamaño de la memoria y el tamaño de la unidad de asignación.

3.1.3. El problema principal es que, cuando se ha decidido llevar un proceso de k unidades a la memoria, el administrador de 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

3.2. Administración de memoria con listas ligadas

3.2.1. Otra manera de llevar el registro de la memoria es mantener una lista ligada de segmentos de memoria asignados y libres, en donde un segmento contiene un proceso o es un hueco vacío entre dos procesos

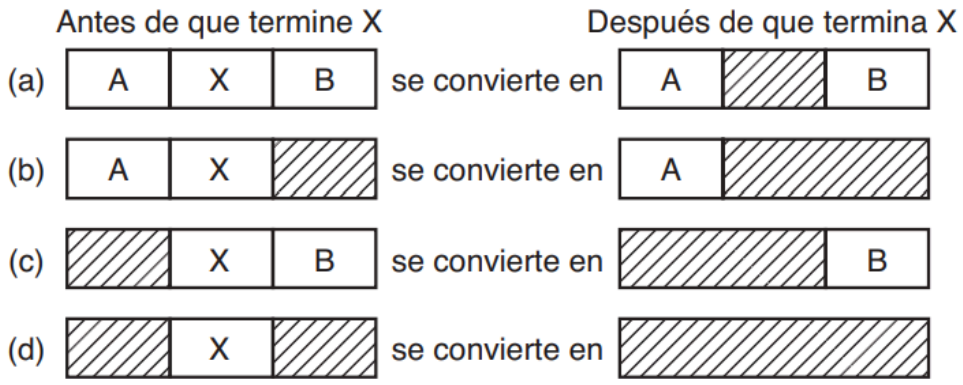


Figura 3-7. Cuatro combinaciones de los vecinos para el proceso en terminación, *X*.

3.2.2. El algoritmo más simple es el de **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. El algoritmo del primer ajuste es rápido debido a que busca lo menos posible.

3.2.3. **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. Las simulaciones realizadas por Bays (1977) muestran que el algoritmo del siguiente ajuste tiene un rendimiento ligeramente peor que el del primer ajuste.

3.2.4. **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 necesitar 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. El algoritmo del mejor ajuste es más lento que el del primer ajuste, ya que debe buscar en toda la lista cada vez que se le llama. De manera sorprendente, también provoca más desperdicio de memoria que los algoritmos del primer ajuste o del siguiente ajuste, debido a que tiende a llenar la memoria con huecos pequeños e inutilizables.

3.2.5. **Peor ajuste**: es decir, tomar siempre el hueco más grande disponible, de manera que el nuevo hueco sea lo bastante grande como para ser útil. La simulación ha demostrado que el algoritmo del peor ajuste no es muy buena idea tampoco.

3.2.6. Con una lista de huecos ordenada por tamaño, los algoritmos del primer ajuste y del mejor ajuste son igual de rápidos, y hace innecesario usar el del siguiente ajuste. **Ajuste rápido**, el cual 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 vecinos para ver si es posible una fusión es un proceso costoso.

- 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).

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.

1. Paginación

    1.1. . Las direcciones se pueden generar usando indexado, registros base, registros de segmentos y otras formas más.Estas direcciones generadas por el programa se conocen como direcciones virtuales y forman el espacio de direcciones virtuales. 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 físicas

    1.2. 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.

    1.3. La MMU detecta que la página no está asociada (lo cual se indica mediante una cruz en la figura) y hace que la CPU haga un trap al sistema operativo. 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 (si no es que ya está ahí). 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.

    1.4. 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 física.

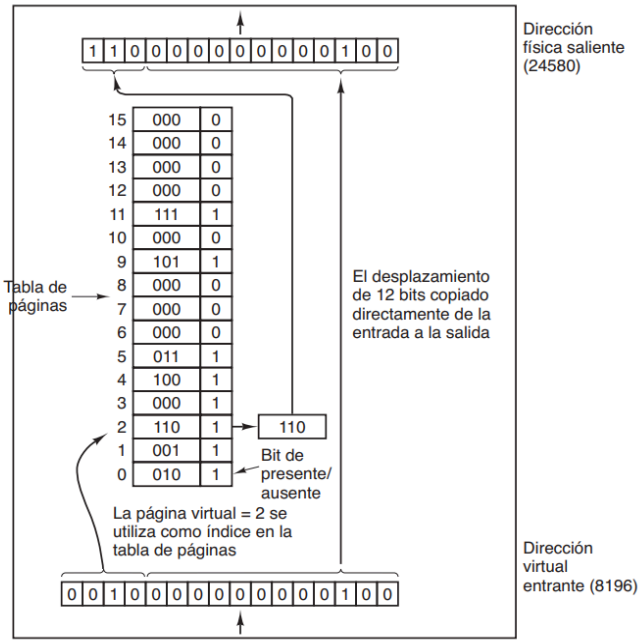


Figura 3-10. La operación interna de la MMU con 16 páginas de 4 KB.

2. Tablas de páginas

2.1. 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.

2.2. 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.

2.3. Estructura de una entrada en la tabla de páginas

2.3.1. 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.

2.3.2. 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.

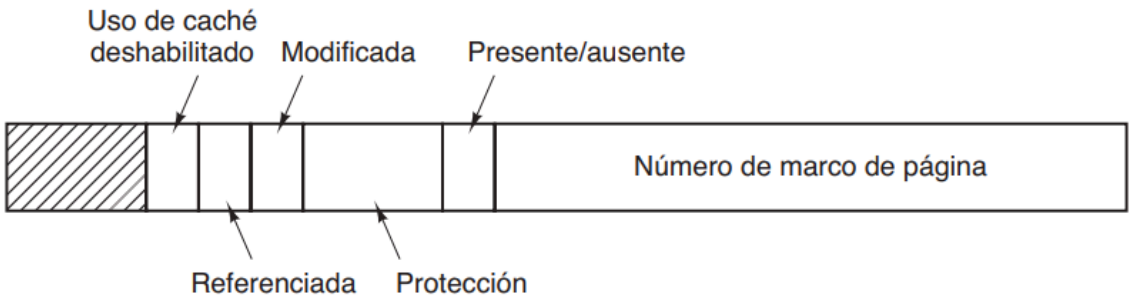


Figura 3-11. Una típica entrada en la tabla de páginas.

2.3.3. 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.

2.3.4. 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 (es decir, está “sucia”), debe escribirse de vuelta en el disco. Si no se ha modificado (es decir, está “limpia”) sólo se puede abandonar, debido a que la copia del disco es aún válida. A este bit se le conoce algunas veces como bit sucio, ya que refleja el estado de la página

2.3.5. 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.

2.3.6. 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. 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.

3. Aceleración de la paginación

3.1. La asociación de una dirección virtual a una dirección física debe ser rápida.

3.1.1. Si la ejecución de una instrucción tarda, por ejemplo 1 nseg, la búsqueda en la tabla de páginas debe realizarse en menos de 0.2 nseg para evitar que la asociación se convierta en un cuello de botella importante.

3.2. Si el espacio de direcciones virtuales es grande, la tabla de páginas será grande

3.2.1. Con 1 millón de páginas en el espacio de direcciones virtual, la tabla de páginas debe tener 1 millón de entradas. Y recuerde que cada proceso necesita su propia tabla de páginas (debido a que tiene su propio espacio de direcciones virtuales).

3.3. Búferes de traducción adelantada

3.3.1. 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. Por ejemplo, considere una instrucción de 1 byte que copia un registro a otro. A falta de paginación, esta instrucción hace sólo una referencia a memoria para obtener la instrucción. Con la paginación se requiere al menos una referencia adicional a memoria para acceder a la tabla de páginas.

3.3.2. 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

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

Figura 3-12. Un TLB para acelerar la paginación.

3.3.3. Cuando se presenta una dirección virtual a la MMU para que la traduzca, el hardware primero comprueba si su número de página virtual está presente en el TLB al compararla con todas las entradas en forma simultánea (es decir, en paralelo). Si se encuentra una coincidencia válida y el acceso no viola los bits de protección, el marco de página se toma directamente del TLB, sin pasar por la tabla de páginas. Si el número de página virtual está presente en el TLB, pero la instrucción está tratando de escribir en una página de sólo lectura, se genera un fallo por protección.

3.3.4. Un caso interesante es lo que ocurre cuando el número de página virtual no está en el TLB. La MMU detecta que no está y realiza una búsqueda ordinaria en la tabla de páginas. Después desaloja una de las entradas del TLB y la reemplaza con la entrada en la tabla de páginas que acaba de buscar.

3.4. Administración del TLB mediante software

3.4.1. 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 sistema debe buscar la página, eliminar una entrada del TLB, introducir la nueva página y reiniciar la instrucción que originó el fallo. Y, desde luego, todo esto se debe realizar en unas cuantas instrucciones, ya que los fallos del TLB ocurren con mucha mayor frecuencia que los fallos de página.

3.4.2. Para reducir los fallos del TLB, algunas veces el sistema operativo averigua de modo “intuitivo” cuáles páginas tienen más probabilidad de ser utilizadas a continuación y precarga entradas para ellas en el TLB.

3.4.3. El problema al realizar esta búsqueda en software es que las páginas que contienen la tabla de páginas tal vez no estén en el TLB, lo cual producirá fallos adicionales en el TLB durante el procesamiento. Estos fallos se pueden reducir al mantener una caché grande en software (por ejemplo, de 4 KB) de entradas en el TLB en una ubicación fija, cuya página siempre se mantenga en el TLB.

3.4.4. Un **fallo suave** ocurre cuando la página referenciada no está en el TLB, sino en memoria. Todo lo que se necesita aquí es que el TLB se actualice. No se necesita E/S de disco. Por lo general, un fallo suave requiere de 10 a 20 instrucciones de máquina y se puede completar en unos cuantos nanosegundos. Por el contrario, un **fallo duro** ocurre cuando la misma página no está en memoria (y desde luego, tampoco en el TLB). Se requiere un acceso al disco para traer la página, lo cual tarda varios milisegundos. Un fallo duro es en definitiva un millón de veces más lento que un fallo suave.

4. Tablas de páginas para memorias extensas

4.1. Tablas de páginas multinivel

4.1.1. 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

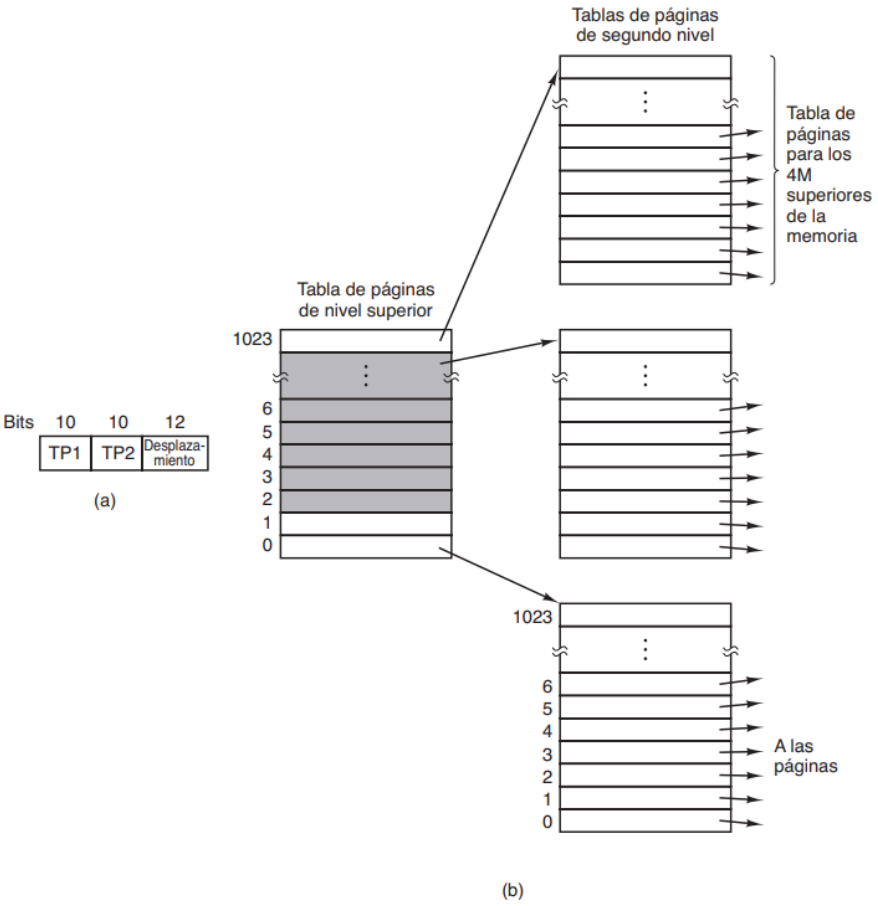


Figura 3-13. (a) Una dirección de 32 bits con dos campos de tablas de páginas. (b) Tablas de páginas de dos niveles.

4.1.2. Lo interesante acerca de la figura 3-13 es que, aunque el espacio de direcciones contiene más de un millón de páginas, en realidad sólo cuatro tablas de páginas son requeridas: la tabla de nivel superior y las tablas de segundo nivel de 0 a 4M (para el texto del programa), de 4M a 8M (para los datos) y los 4M superiores (para la pila).

4.1.3. Sin embargo, si el espacio de direcciones es de 264 bytes, con páginas de 4KB, 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). Ocupar 30 millones de gigabytes sólo para la tabla de páginas no es una buena idea por ahora y probablemente tampoco lo sea para el próximo año

4.2. Tablas de páginas invertidas

4.2.1. 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.

4.2.2. 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.

4.2.3. 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. Una vez que se ha encontrado el número de marco de página, se introduce el nuevo par (virtual, física) en el TLB

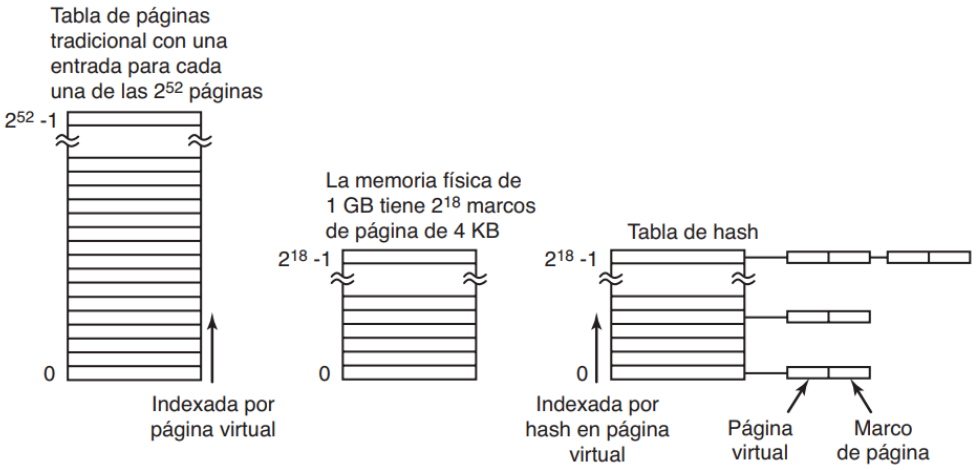


Figura 3-14. Comparación de una tabla de páginas tradicional con una tabla de páginas invertida.

- 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.

Un segundo ejemplo es en un servidor Web. El servidor puede mantener cierto número de páginas Web de uso muy frecuente en su memoria caché. Sin embargo, cuando la memoria caché está llena y se hace referencia a una nueva página, hay que decidir cuál página Web se debe desalojar.

1. El algoritmo de reemplazo de páginas óptimo

1.1. El mejor algoritmo de reemplazo de páginas posible es fácil de describir, pero imposible de implementa

1.2. 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.

1.3. 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

2. El algoritmo de reemplazo de páginas: no usadas recientemente

    2.1. El sistema operativo establece el bit R (en sus tablas internas), cambia la entrada en la tabla de páginas para que apunte a la página correcta, con modo de SÓLO LECTURA, y reinicia la instrucción. Si la página se modifica después, ocurrirá otro fallo de página que permite al sistema operativo establecer el bit M y cambiar el modo de la página a LECTURA/ESCRITURA

    2.2. 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.

    2.3. 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.

3. El algoritmo de reemplazo de páginas: Primera en entrar, primera en salir (FIFO)

    3.1. 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.

    3.2. Cuando se aplica en las tiendas, FIFO podría eliminar la gomina para el bigote, pero también podría eliminar harina, sal o mantequilla. Cuando se aplica a las computadoras, surge el mismo problema. Por esta razón es raro que se utilice FIFO en su forma pura.

4. El algoritmo de reemplazo de páginas: segunda oportunidad

    4.1. 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.

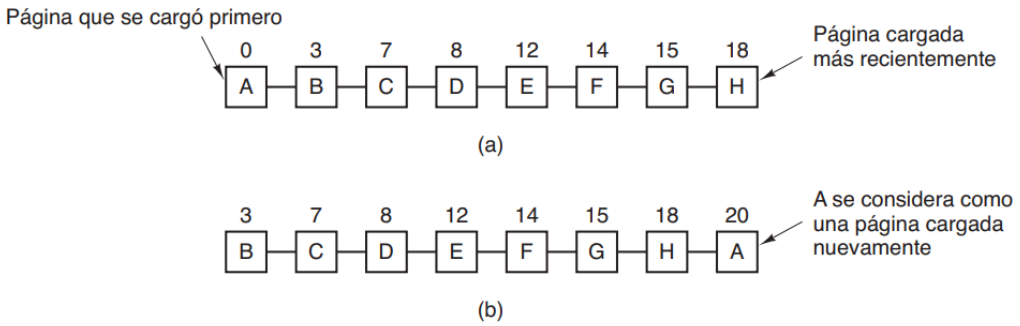


Figura 3-15. Operación del algoritmo de la segunda oportunidad. (a) Páginas ordenadas con base en FIFO. (b) Lista de las páginas si ocurre un fallo de página en el tiempo 20 y *A* tiene su bit *R* activado. Los números encima de las páginas son sus tiempos de carga.

4.2. Suponga que ocurre un fallo de página en el tiempo 20. La página más antigua es A, que llegó en el tiempo 0, cuando se inició el proceso. Si el bit R de A está desactivado, se desaloja de la memoria, ya sea escribiéndola en el disco (si está sucia) o sólo se abandona (si está limpia). Por otro lado, si el bit R está activado, A se coloca al final de la lista y su "tiempo de carga" se restablece al tiempo actual (20). El bit R también está desactivado. La búsqueda de una página adecuada continúa con B.

5. El algoritmo de reemplazo de páginas: reloj

5.1. 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.

5.2. 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.

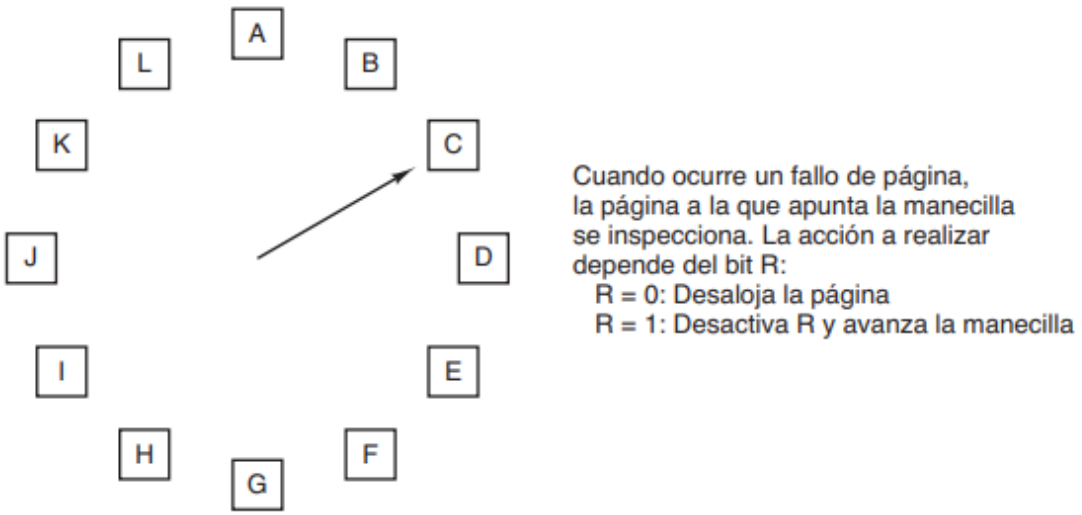


Figura 3-16. El algoritmo de reemplazo de páginas en reloj.

6. El algoritmo de reemplazo de páginas: menos usadas recientemente (LRU)

6.1. 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).

6.2. 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. Buscar una página en la lista, eliminarla y después pasarla al frente es una operación que consume mucho tiempo, incluso mediante hardware

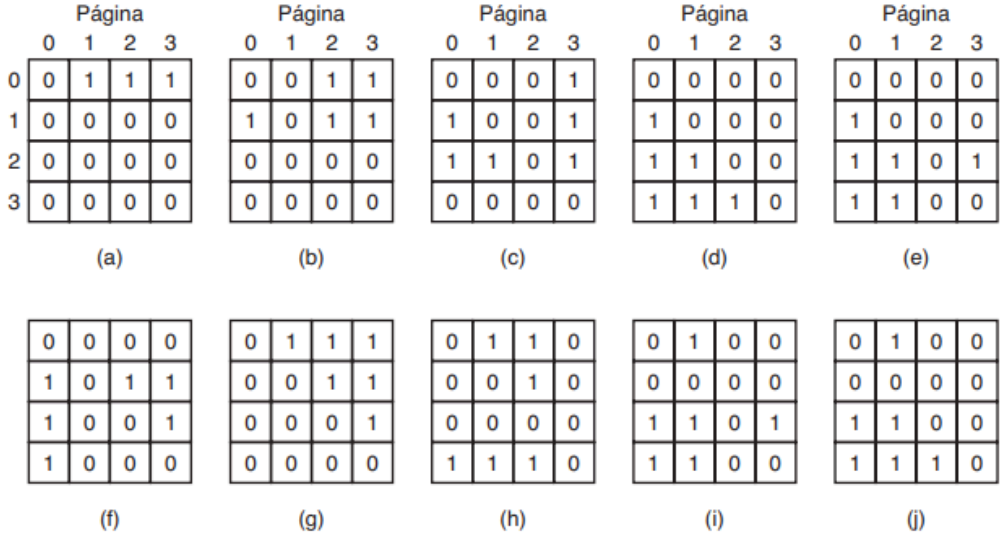


Figura 3-17. LRU usando una matriz cuando se hace referencia a las páginas en el orden 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

7. Simulación de LRU en software

7.1. 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.

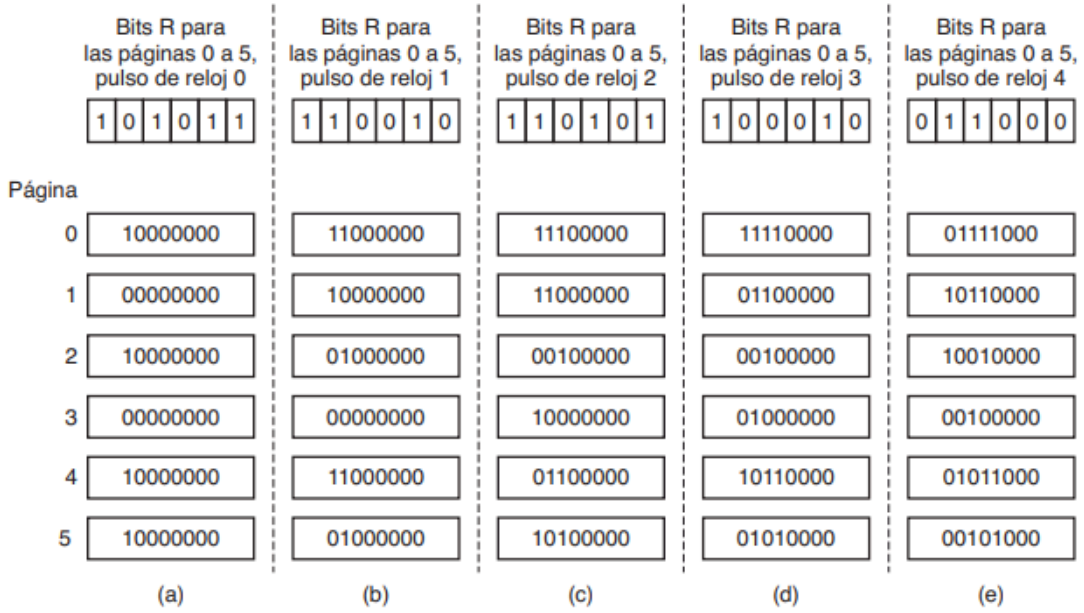


Figura 3-18. El algoritmo de envejecimiento simula el LRU en software. Aquí se muestran seis páginas para cinco pulsos de reloj. Los cinco pulsos de reloj se representan mediante los incisos (a) a (e).

7.2. Este algoritmo difiere del de LRU en dos formas.

7.2.1. Al registrar sólo un bit por cada intervalo de tiempo, hemos perdido la capacidad de diferenciar las referencias anteriores en el intervalo de reloj de las que ocurrieron después.

7.2.2. La segunda diferencia entre los algoritmos de LRU y envejecimiento es que en este último los contadores tienen un número finito de bits (8 en este ejemplo), lo cual limita su horizonte pasado.

8. El algoritmo de reemplazo de páginas: conjunto de trabajo

8.1. En la forma más pura de paginación, los procesos inician sin ninguna de sus páginas en la memoria. Tan pronto como la CPU trata de obtener la primera instrucción, recibe un fallo de página, causando que el sistema operativo tenga que traer la página que contiene la primera instrucción. Por lo general a este fallo le siguen otros fallos de página por las variables globales y la pila. Después de cierto tiempo, el proceso tiene la mayoría de las páginas que necesita y se establece para ejecutarse con relativamente pocos fallos de página. A esta estrategia se le conoce como paginación bajo demanda

8.2. Exhiben una localidad de referencia, lo cual significa que durante cualquier fase de ejecución el proceso hace referencia sólo a una fracción relativamente pequeña de sus páginas.

8.3. El conjunto de páginas que utiliza un proceso en un momento dado se conoce como su conjunto de trabajo. Si todo el conjunto de trabajo está en

memoria, el proceso se ejecutará sin producir muchos fallos hasta que avance a otra fase de ejecución. Si la memoria disponible es demasiado pequeña como para contener todo el conjunto de trabajo completo, el proceso producirá muchos fallos de página y se ejecutará lentamente, ya que para ejecutar una instrucción se requieren unos cuantos nanosegundos y para leer una página del disco se requieren en general 10 milisegundos

8.4. Se dice que un programa que produce fallos de página cada pocas instrucciones está sobrepaginando (thrashing)

8.5. Por lo tanto, 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.

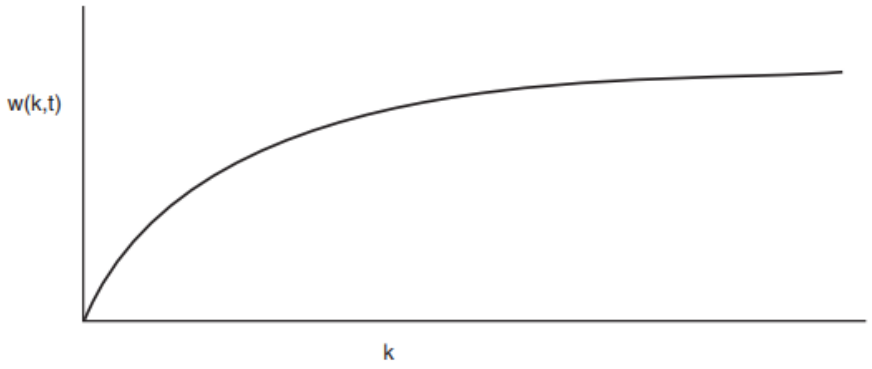


Figura 3-19. El conjunto de trabajo es el conjunto de páginas utilizadas por las k referencias a memoria más recientes. La función $w(k, t)$ es el tamaño del conjunto de trabajo en el tiempo t.

8.6. Para implementar el modelo del conjunto de trabajo, es necesario que el sistema operativo lleve la cuenta de cuáles páginas están en el conjunto de trabajo. Tener esta información también nos conduce de inmediato a un posible algoritmo de reemplazo de páginas: cuando ocurra un fallo de página, hay que buscar una página que no se encuentre en el conjunto de trabajo y desalojar la.

8.7. Para implementar cualquier algoritmo de conjunto de trabajo se debe elegir por adelantado cierto valor de k. Una vez que se ha seleccionado cierto valor, después de cada referencia a memoria, el conjunto de páginas utilizadas por las k referencias a memoria más recientes se determina en forma única.

8.8. En teoría, en un fallo de página, el contenido del registro de desplazamiento se podría extraer y ordenar; las páginas duplicadas entonces pueden ser eliminadas. El resultado sería el conjunto de trabajo. Sin embargo, el proceso de mantener el registro de desplazamiento y procesarlo en un fallo de página sería en extremo costoso, por lo que esta técnica nunca se utiliza.

8.9. Por ejemplo, en vez de definir el conjunto de trabajo como las páginas utilizadas durante los 10 millones de referencias a memoria anteriores, podemos definirlo como el conjunto de páginas utilizadas durante los últimos 100 milisegundos de tiempo de ejecución. En la práctica, dicha definición es igual de conveniente y es mucho más fácil trabajar con ella.

8.10. La cantidad de tiempo de la CPU que ha utilizado en realidad un proceso desde que empezó se conoce comúnmente como su tiempo virtual actual. Con esta aproximación, el conjunto de trabajo de un proceso es el conjunto de páginas a las que se ha hecho referencia durante los últimos τ segundos de tiempo virtual.

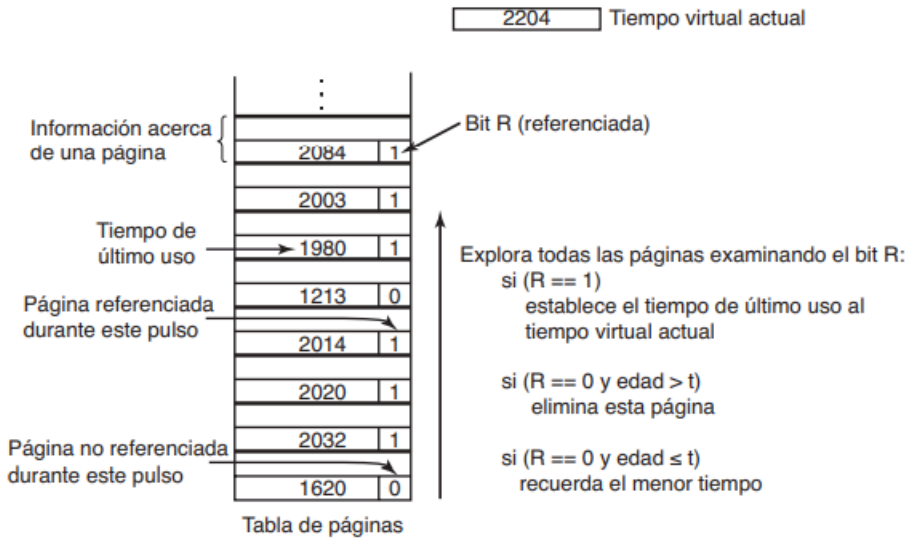


Figura 3-20. El algoritmo del conjunto de trabajo.

8.11. , se examina el bit R. Si es 1, el tiempo virtual actual se escribe en el campo Tiempo de último uso en la tabla de páginas, indicando que la página estaba en uso al momento en que ocurrió el fallo de página. Como se hizo referencia a la página durante el pulso de reloj actual, es evidente que está en el conjunto de trabajo y no es candidata para la eliminación

8.12. Si R es 0, no se ha hecho referencia a la página durante el pulso de reloj actual y puede ser candidata para la eliminación. Para ver si debe o no eliminarse, se calcula su edad (el tiempo virtual actual menos su Tiempo de último uso) y se compara con τ. Si la edad es mayor que τ, la página ya no se encuentra en el conjunto de trabajo y la nueva página la sustituye. La exploración continúa actualizando las entradas restantes.

8.13. No obstante, si R es 0 pero la edad es menor o igual que τ, la página está todavía en el conjunto de trabajo. La página se reserva temporalmente, pero se apunta la página con la mayor edad (el menor valor de Tiempo de último uso). Si toda la tabla completa se explora sin encontrar un candidato para desalojar, eso significa que todas las páginas están en el conjunto de trabajo. En ese caso, si se encontraron una o más páginas con R 0, se desaloja la más antigua

9. El algoritmo de reemplazo de páginas WSClock

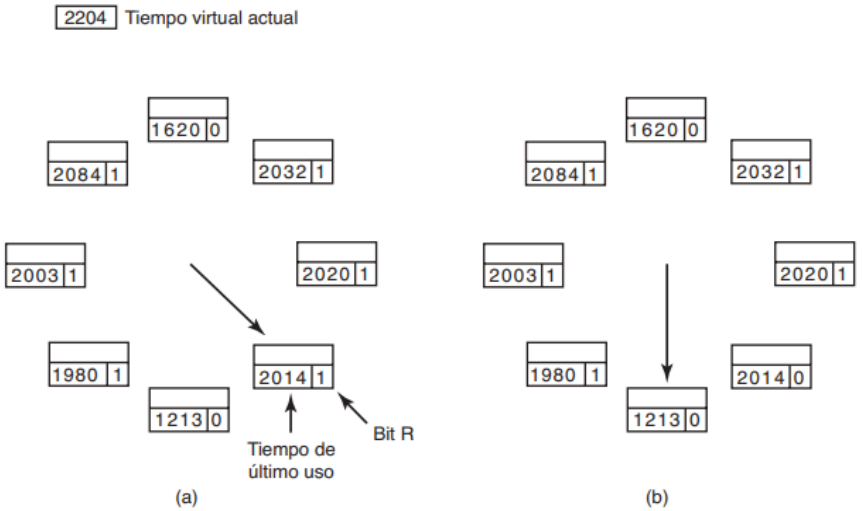
9.1. La estructura de datos necesaria es una lista circular de marcos de página, como en el algoritmo de reloj. Al principio, esta lista está vacía. Cuando se carga la primera página, se agrega a la lista. A medida que se agregan más páginas, pasan a la lista para formar un anillo. Cada entrada contiene 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).

9.2. Al igual que con el algoritmo de reloj, en cada fallo de página se examina primero la página a la que apunta la manecilla. Si el bit R es 1, la página se ha utilizado durante el pulso actual, por lo que no es candidata ideal para la eliminación. Después el bit R se establece en 0, la manecilla se avanza a la siguiente página y se repite el algoritmo para esa página. Si la página a la que apunta la manecilla tiene R 0. Si la edad es mayor que τ y la página está limpia, significa que no se encuentra en el conjunto de trabajo y existe una copia válida en el disco.

9.3. ¿Qué ocurre si la manecilla llega otra vez a su punto inicial?
Hay dos casos a considerar:
1. Se ha planificado por lo menos una escritura.
2. No se han planificado escrituras.

9.4. En el primer caso, la manecilla sólo sigue moviéndose, buscando una página limpia. Como se han planificado una o más escrituras, en algún momento se completará alguna escritura y su página se marcará como limpia. La primera página limpia que se encuentre se desaloja.

9.5. En el segundo caso, todas las páginas están en el conjunto de trabajo, de otra manera se hubiera planificado por lo menos una escritura. Sin información adicional, lo más simple por hacer es reclamar cualquier página limpia y usarla. La ubicación de una página limpia podría rastrearse durante el barrido. Si no existen páginas limpias, entonces se selecciona la página actual como la víctima y se escribe de vuelta al disco.



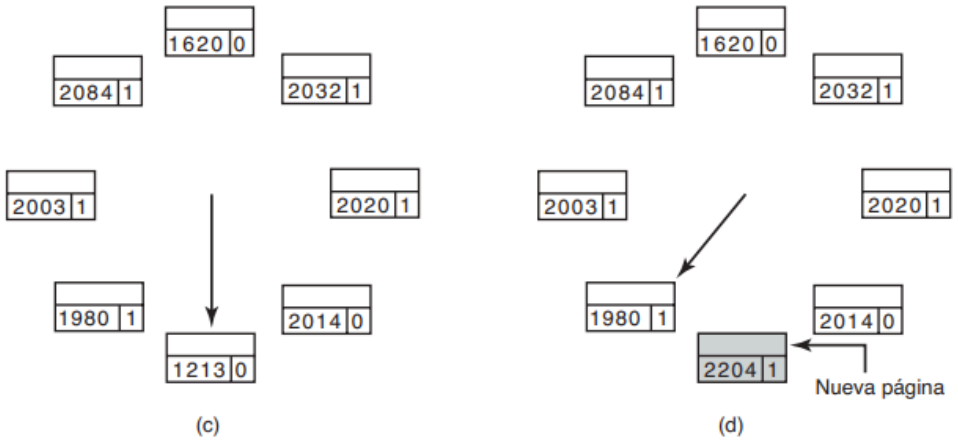


Figura 3-21. Operación del algoritmo WSClock. (a) y (b) dan un ejemplo de lo que ocurre cuando $R = 1$. (c) y (d) dan un ejemplo de cuando $R = 0$.

10. Resumen de los algoritmos de reemplazo de página

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

Figura 3-22. Algoritmos de reemplazo de páginas descritos en el texto.

10.1. El algoritmo óptimo desaloja la página a la que se hará referencia en el futuro más lejano. Por desgracia, no hay forma de determinar cuál página es, por lo que en la práctica no se puede utilizar este algoritmo. Sin embargo, es útil como punto de comparación para los demás algoritmos.

10.2. El algoritmo NRU divide las páginas en cuatro clases dependiendo del estado de los bits R y M. Se selecciona una página aleatoria de la clase con menor numeración. Este algoritmo es fácil de implementar, pero muy burdo. Existen mejores.

10.3. FIFO lleva la cuenta del orden en el que se cargaron las páginas en memoria al mantenerlas en una lista enlazada. Eliminar la página más antigua se vuelve entonces un proceso trivial, pero como esa página podría estar todavía en uso, FIFO es una mala opción.

10.4. El algoritmo de segunda oportunidad es una modificación de FIFO que comprueba si hay una página en uso antes de eliminarla. Si lo está, la página se reserva. Esta modificación mejora de manera considerable el rendimiento. El algoritmo de reloj es simplemente una implementación distinta del algoritmo de segunda oportunidad. Tiene las mismas propiedades de rendimiento, pero toma un poco menos de tiempo para ejecutar el algoritmo.

10.5. LRU es un algoritmo excelente, pero no se puede implementar sin hardware especial. NFU es un burdo intento por aproximarse a LRU; sin embargo, el algoritmo de envejecimiento es una mucho mejor aproximación a LRU y se puede implementar con eficiencia. Es una buena opción.

10.6. Los últimos dos algoritmos utilizan el conjunto de trabajo. El algoritmo del conjunto de trabajo ofrece un rendimiento razonable, pero es un poco costoso de implementar. WSClock es una variante que no sólo da un buen rendimiento, sino que también es eficiente al implementar

10.7. Con todo, los dos mejores algoritmos son el de envejecimiento y WSClock. Se basan en LRU y el conjunto de trabajo, respectivamente. Ambos dan un buen rendimiento en la paginación y pueden implementarse con eficiencia. Existen varios algoritmos más, pero estos dos son tal vez los más importantes en la práctica.

- CUESTIONES DE DISEÑO PARA LOS SISTEMAS DE PAGINACIÓN

1. Políticas de asignación local contra las de asignación global

1.1. En general, los algoritmos globales funcionan mejor, en especial cuando el tamaño del conjunto de trabajo puede variar durante el tiempo de vida de un proceso. Si se utiliza un algoritmo local y el conjunto de trabajo crece, se producirá una sobrepaginación, aun cuando haya muchos marcos de página libres. Si el conjunto de trabajo se reduce, los algoritmos locales desperdician memoria.

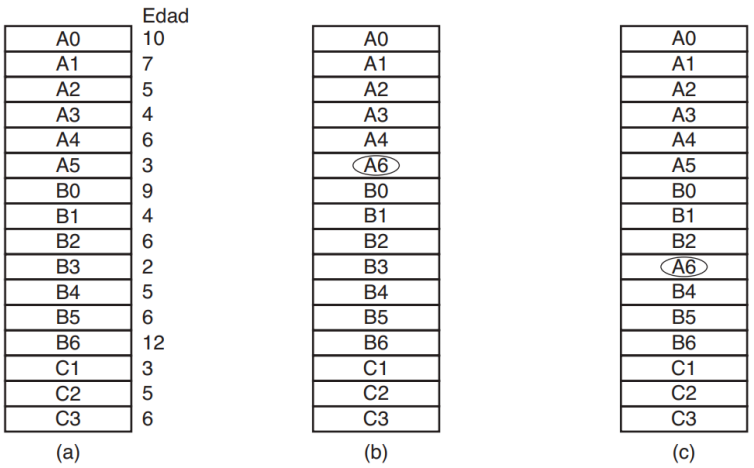


Figura 3-23. Comparación entre el reemplazo de páginas local y el global. (a) Configuración original. (b) Reemplazo de páginas local. (c) Reemplazo de páginas global.

1.2. Si se utiliza un algoritmo global, es posible empezar cada proceso con cierto número de páginas proporcional al tamaño del proceso, pero la asignación se tiene que actualizar dinámicamente a medida que se ejecuten los procesos. Una manera de administrar la asignación es utilizando el algoritmo PFF (Page Fault Frequency, Frecuencia de fallo de páginas). Este algoritmo indica cuándo se debe incrementar o decrementar la asignación de páginas a un proceso, pero no dice nada acerca de cuál página se debe sustituir en un fallo. Sólo controla el tamaño del conjunto de asignación.

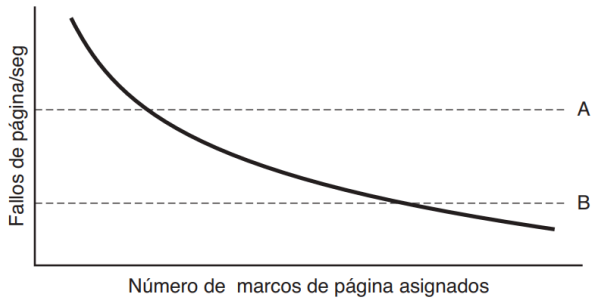


Figura 3-24. Proporción de fallos de página como una función del número de marcos de página asignados.

1.3. La línea punteada marcada como A corresponde a una proporción de fallos de página que es demasiado alta, por lo que el proceso que emitió el fallo recibe más marcos de página para reducir la proporción de fallos. La línea punteada marcada como B corresponde a una proporción de fallos de página tan baja que podemos suponer que el proceso tiene demasiada memoria. En este caso se le pueden quitar marcos de página.

2. Control de carga

2.1. Aun con el mejor algoritmo de reemplazo de páginas y una asignación global óptima de marcos de página a los procesos, puede ocurrir que el sistema se sobrepagine.

2.2. Un síntoma de esta situación es que el algoritmo PFF indica que algunos procesos necesitan más memoria, pero ningún proceso necesita menos memoria. En este caso no hay forma de proporcionar más memoria a esos procesos que la necesitan sin lastimar a algún otro proceso. La única solución real es deshacerse temporalmente de algunos procesos.

2.3. El proceso de intercambiar procesos para liberar la carga en la memoria es semejante a la planificación de dos niveles, donde ciertos procesos se colocan en disco y se utiliza un planificador de corto plazo para planificar los procesos restantes. Periódicamente, ciertos procesos se traen del disco y otros se intercambian hacia el mismo.

2.4. Cuando el número de procesos en la memoria principal es demasiado bajo, la CPU puede estar inactiva durante largos periodos. Esta consideración sostiene que no sólo se debe tomar en cuenta el tamaño del proceso y la

proporción de paginación al decidir qué proceso se debe intercambiar, sino también sus características, tal como si está ligado a la CPU o a la E/S, así como las características que tienen los procesos restantes.

3. Tamaño de página

3.1. El tamaño de página es un parámetro que a menudo el sistema operativo puede elegir. Incluso si el hardware se ha diseñado

3.2. Para determinar el mejor tamaño de página se requiere balancear varios factores competitivos. Como resultado, no hay un tamaño óptimo en general. Para empezar, hay dos factores que están a favor de un tamaño de página pequeño. Un segmento de texto, datos o pila elegido al azar no llenará un número integral de páginas. En promedio, la mitad de la página final estará vacía. El espacio adicional en esa página se desperdicia. A este desperdicio se le conoce como fragmentación interna.

3.3. En general, un tamaño de página grande hará que haya una parte más grande no utilizada del programa que un tamaño de página pequeño. Por otro lado, tener páginas pequeñas implica que los programas necesitarán muchas páginas, lo que sugiere la necesidad de una tabla de páginas grande.

$$\text{sobrecarga} = se/p + p/2$$

3.4. El primer término (tamaño de la tabla de páginas) es grande cuando el tamaño de página es pequeño. El segundo término (fragmentación interna) es grande cuando el tamaño de página es grande. El valor óptimo debe estar entre estos dos.

4. Espacios separados de instrucciones y de datos

4.1. La mayor parte de las computadoras tienen un solo espacio de direcciones que contiene tanto programas como datos. Si este espacio de direcciones es lo bastante grande, todo funciona bien. No obstante, a menudo es demasiado pequeño, lo cual obliga a los programadores a pararse de cabeza tratando de ajustar todo en el espacio de direcciones.

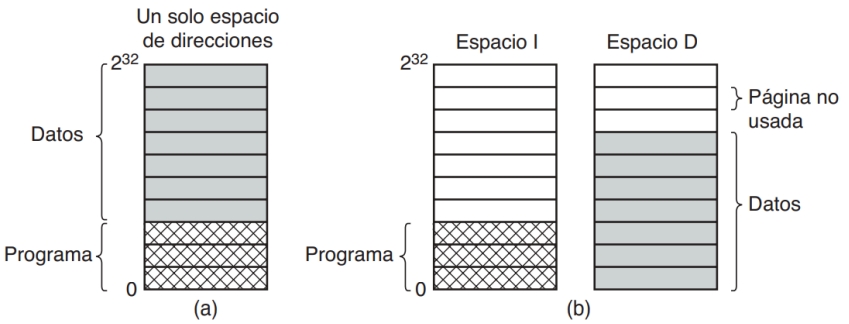


Figura 3-25. (a) Un espacio de direcciones. (b) Espacios I y D separados.

4.2. Una solución utilizada es tener espacios de direcciones separados para las instrucciones (texto del programa) y los datos, llamados espacio I y espacio D. Tener espacios I y D separados no introduce ninguna complicación especial y sí duplica el espacio de direcciones disponible

5. Páginas compartidas

5.1. En un sistema de multiprogramación grande, es común que varios usuarios ejecuten el mismo programa a la vez. Evidentemente es más eficiente compartir las páginas para evitar tener dos copias de la misma página en memoria al mismo tiempo. Un problema es que no todas las páginas se pueden compartir. En especial, sólo pueden compartirse las páginas que son de sólo lectura como el texto del programa, pero las páginas de datos no.

5.2. Si se admiten espacios I y D separados, es relativamente simple compartir los programas al hacer que dos o más procesos utilicen la misma tabla de páginas para su espacio I pero distintas tablas de páginas para sus espacios D.

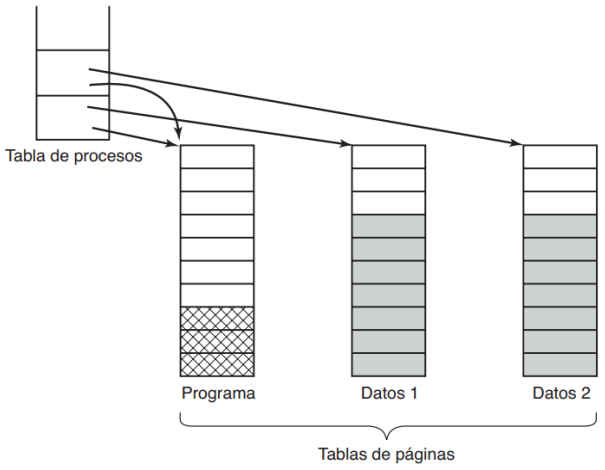


Figura 3-26. Dos procesos comparten el mismo programa compartiendo su tabla de páginas.

5.3. Suponga que los procesos A y B están ejecutando el editor y comparten sus páginas. Si el planificador decide eliminar A de la memoria, desalojando todas sus páginas y llenando los marcos de página vacíos con otro programa, causará que B genere un gran número de fallos de página para traerlas de vuelta.

5.4. Compartir datos es más complicado que compartir código, pero no imposible. En especial, en UNIX después de una llamada al sistema fork, el padre y el hijo tienen que compartir tanto el texto del programa como el texto de los datos. En un sistema paginado, lo que se hace a menudo es dar a cada uno de estos procesos su propia tabla de páginas y hacer que ambos apunten al mismo conjunto de páginas. Así, no se realiza una copia de páginas al momento de la operación fork. Sin embargo, todas las páginas de datos son asociadas en ambos procesos de SÓLO LECTURA (READ ONLY).

5.5. Tan pronto como cualquiera de los procesos actualiza una palabra de memoria, la violación de la protección de sólo lectura produce un trap al sistema operativo. Después se hace una copia de la página ofensora, para que cada proceso tenga ahora su propia copia privada. Ambas copias se establecen ahora como LECTURA-ESCRITURA (READ-WRITE), para que las siguientes operaciones de escritura en cualquiera de las copias continúen sin lanzar un trap. Esta estrategia significa que aquellas páginas que nunca se modifican (incluyendo todas las del programa) no se necesitan copiar. Este método, conocido como copiar en escritura, mejora el rendimiento al reducir el copiado.

6. Bibliotecas compartidas

6.1. En los sistemas modernos hay muchas bibliotecas extensas utilizadas por muchos procesos, por ejemplo, la biblioteca que maneja el diálogo para explorar por archivos que se desean abrir y varias bibliotecas de gráficos. Si se enlazaran en forma estática estas bibliotecas con cada programa ejecutable en el disco se agrandarían aún más.

6.2. En vez de ello, una técnica común es utilizar bibliotecas compartidas (que se conocen como DLLs o Bibliotecas de enlaces dinámicos en Windows). Para aclarar la idea de una biblioteca compartida, primero considere el enlazamiento tradicional. Cuando un programa se enlaza, se nombra uno o más archivos de código objeto y posiblemente algunas bibliotecas en el comando para el enlazador

6.3. (objeto) en el directorio actual y después explora dos bibliotecas, /usr/lib/libc.a y /usr/lib/libm.a. Las funciones a las que se llame en los archivos objeto pero que no estén ahí (por ejemplo, printf) se conocen como externas indefinidas y se buscan en las bibliotecas. Si se encuentran, se incluyen en el binario ejecutable. Cualquier función a la que llamen pero que no esté aún presente también se convierte en externa indefinida.Por ejemplo, printf necesita a write, por lo que si write no está ya incluida, el enlazador la buscará y la incluirá cuando la encuentre.

6.4. Cuando un programa se vincula con bibliotecas compartidas (que son ligeramente diferentes a las estáticas), en vez de incluir la función a la que se llamó, el vinculador incluye una pequeña rutina auxiliar que se enlaza a la función llamada en tiempo de ejecución. Dependiendo del sistema y los detalles de configuración, las bibliotecas compartidas se cargan cuando se carga el programa o cuando las funciones en ellas se llaman por primera vez.

6.5. Observe que cuando se carga o utiliza una biblioteca compartida, no se lee toda la biblioteca en memoria de un solo golpe. Se pagina una página a la vez según sea necesario, de manera que las funciones que no sean llamadas no se carguen en la RAM. Además de reducir el tamaño de los archivos ejecutables y ahorrar espacio en memoria, las bibliotecas compartidas tienen otra ventaja: si una función en una biblioteca compartida se actualiza para eliminar un error, no es necesario recompilar los programas que la llaman, pues los antiguos binarios siguen funcionando.

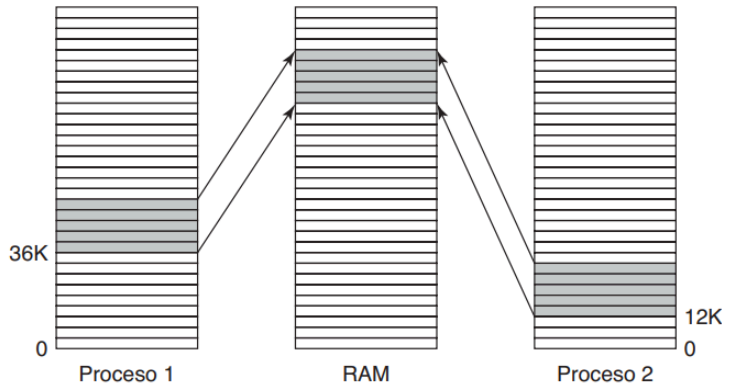


Figura 3-27. Una biblioteca compartida utilizada por dos procesos.

6.6. Una mejor solución es compilar las bibliotecas compartidas con una bandera de compilador especial, para indicar al compilador que no debe producir instrucciones que utilicen direcciones absolutas. En vez de ello, sólo se utilizan instrucciones con direcciones relativas. Al evitar direcciones absolutas, el problema se puede resolver. El código que utiliza sólo desplazamientos relativos se conoce como código independiente de la posición.

7. Archivos asociados

7.1. Las bibliotecas compartidas son realmente un caso de una herramienta más general, conocida como archivos asociados a memoria. La idea aquí es que un proceso puede emitir una llamada al sistema para asociar un archivo a una porción de su espacio de direcciones virtuales.

7.2. Los archivos asociados proporcionan un modelo alternativo para la E/S. En vez de realizar lecturas y escrituras, el archivo se puede accesar como un gran arreglo de caracteres en la memoria.

7.3. Si dos o más procesos se asocian al mismo archivo y al mismo tiempo, se pueden comunicar a través de la memoria compartida. Las escrituras realizadas por un proceso en la memoria compartida son inmediatamente visibles cuando el otro lee de la parte de su espacio de direcciones virtuales asociado al archivo. Por lo tanto, este mecanismo proporciona un canal con un gran ancho de banda entre los procesos, y a menudo se utiliza como tal.

8. Política de limpieza

8.1. La paginación funciona mejor cuando hay muchos marcos de página libres que se pueden reclamar al momento en que ocurran fallos de página. Si cada marco de página está lleno y además modificado, antes de que se pueda traer una nueva página se debe escribir una página anterior en el disco.

8.2. Para asegurar una provisión abundante de marcos de página libres, muchos sistemas de paginación tienen un proceso en segundo plano conocido como demonio de paginación, que está inactivo la mayor parte del tiempo pero se despierta en forma periódica para inspeccionar el estado de la memoria. Si hay muy pocos marcos de página libres, el demonio de paginación empieza a seleccionar páginas para desalojarlas mediante cierto algoritmo de reemplazo de páginas. Si estas páginas han sido modificadas después de haberse cargado, se escriben en el disco.

8.3. Una manera de implementar esta política de limpieza es mediante un reloj con dos manecillas. La manecilla principal es controlada por el demonio de paginación. Cuando apunta a una página sucia, esa página se escribe de vuelta al disco y la manecilla principal se avanza. Cuando apunta a una página limpia, sólo se avanza. La manecilla secundaria se utiliza para reemplazar páginas, como en el algoritmo de reloj estándar. Sólo que ahora, la probabilidad de que la manecilla secundaria llegue a una página limpia se incrementa debido al trabajo del demonio de paginación.

9. Interfaz de memoria virtual

9.1. Una razón por la que se otorga a los programadores el control sobre su mapa de memoria es para permitir que dos o más procesos compartan la misma memoria. Si los programadores pueden nombrar regiones de su memoria, tal vez sea posible para un proceso dar a otro proceso el nombre de una región de memoria, de manera que el proceso también pueda asociarla. Con dos (o más) procesos compartiendo las mismas páginas, la compartición con mucho ancho de banda se hace posible: un proceso escribe en la memoria compartida y otro proceso lee de ella.

9.2. Otra técnica más de administración avanzada de memoria es la memoria compartida distribuida. La idea aquí es permitir que varios procesos compartan a través de la red un conjunto de páginas, posiblemente (pero no es necesario) como un solo espacio de direcciones lineal compartido. Cuando un proceso hace referencia a una página que no está asociada, obtiene un fallo de página. El manejador de fallos de página, que puede estar en espacio de kernel o de usuario, localiza entonces la máquina que contiene la página y le envía un mensaje pidiéndole que la desasocie y la envíe a través de la red. Cuando llega la página, se asocia y la instrucción que falló se reinicia.

- CUESTIONES DE IMPLEMENTACIÓN

Los implementadores de los sistemas de memoria virtual tienen que elegir entre los principales algoritmos teóricos: entre el algoritmo de segunda oportunidad y el de envejecimiento, entre la asignación de páginas local o global, y entre la paginación bajo demanda o la prepaginación.

1. Participación del sistema operativo en la paginación

    1.1. 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.

    1.2. Cuando se crea un proceso en un sistema de paginación, el sistema operativo tiene que determinar qué tan grandes serán el programa y los datos (al principio), y crear una tabla de páginas para ellos. Se debe asignar espacio en memoria para la tabla de páginas y se tiene que inicializar. Además, se debe asignar espacio en el área de intercambio en el disco, para que cuando se intercambie una página, tenga un lugar a donde ir. Algunos sistemas paginan el texto del programa directamente del archivo ejecutable, con lo cual se ahorra espacio en disco y tiempo de inicialización. Por último, la información acerca de la tabla de páginas y el área de intercambio en el disco se debe registrar en la tabla de procesos.

    1.3. Cuando un proceso se planifica para ejecución, la MMU 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. La tabla de páginas del nuevo proceso se tiene que actualizar, por lo general copiándola o mediante un apuntador a éste hacia cierto(s) registro(s) de hardware.

    1.4. Cuando ocurre un fallo de página, el sistema operativo tiene que leer los registros de hardware para determinar cuál dirección virtual produjo el fallo. Con base en esta información debe calcu lar qué página se necesita y localizarla en el disco. Después debe buscar un marco de página disponible para colocar la nueva página, desalojando alguna página anterior si es necesario. Luego debe leer la página necesaria y colocarla en el marco de páginas. Por último, debe respaldar el contador de programa para hacer que apunte a la instrucción que falló y dejar que la instrucción se ejecute de nuevo.

    1.5. 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. Si alguna de las páginas están compartidas con otros procesos, las páginas en memoria y en disco sólo pueden liberarse cuando el último proceso que las utilice haya terminado.

2. Manejo de fallos de página

    2.1. 1. El hardware hace un trap 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.

    2.2. 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.

2.3. 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. De no ser así, el sistema operativo debe obtener el contador de programa, obtener la instrucción y analizarla en software para averiguar lo que estaba haciendo cuando ocurrió el fallo.

2.4. 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.

2.5. 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.

2.6. 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.

2.7. 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.

2.8. 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.

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

2.10. 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.

3. Respaldo de instrucción

3.1. Cuando un programa hace referencia a una página que no está en memoria, la instrucción que produjo el fallo se detiene parcialmente y ocurre un trap al sistema operativo. Una vez que el sistema operativo obtiene la página necesaria, debe reiniciar la instrucción que produjo el trap. Es más fácil decir esto que hacerlo.

3.2. Por fortuna, en algunas máquinas los diseñadores de la CPU proporcionan una solución, por lo general en la forma de un registro interno oculto, en el que se copia el contador de programa justo antes de ejecutar cada instrucción. Estas máquinas también pueden tener un segundo registro que indique cuáles registros se han ya autoincrementado o autodecrementado y por cuánto. Dada esta información, el sistema operativo puede deshacer sin ambigüedad todos los efectos de la instrucción fallida, de manera que se pueda reiniciar.

4. Bloqueo de páginas en memoria

4.1. Considere un proceso que acaba de emitir una llamada al sistema para leer algún archivo o dispositivo y colocarlo en un búfer dentro de su espacio de direcciones. Mientras espera a que se complete la E/S, el proceso se suspende y se permite a otro proceso ejecutarse. Este otro proceso recibe un fallo de página.

4.2. 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. Otra solución es enviar todas las operaciones de E/S a búferes del kernel y después copiar los datos a las páginas de usuario.

5. Almacén de respaldo

5.1. El algoritmo más simple para asignar espacio de página en el disco es tener una partición de intercambio especial en el disco o aún mejor es tenerla en un disco separado del sistema operativo. Esta partición no tiene un sistema de archivos normal, lo cual elimina la sobrecarga de convertir desplazamientos en archivos a direcciones de bloque. En vez de ello, se utilizan números de bloque relativos al inicio de la partición.

5.2. Con cada proceso está asociada la dirección de disco de su área de intercambio; es decir, en qué parte de la partición de intercambio se mantiene su imagen. Esta información se mantiene en la tabla de procesos. El cálculo la dirección en la que se va a escribir una página es simple: sólo se suma el desplazamiento de la página dentro del espacio de direcciones virtual al inicio del área de intercambio.

5.3. Los procesos pueden incrementar su tamaño antes de empezar. Aunque el texto del programa por lo general es fijo, el área de los datos puede crecer algunas veces, y la pila siempre puede crecer. En consecuencia, podría ser mejor reservar áreas de intercambio separadas para el texto, los datos y la pila, permitiendo que cada una de estas áreas consista de más de un trozo en el disco

5.4. El otro extremo es no asignar nada por adelantado y asignar espacio en el disco para cada página cuando ésta se intercambie hacia fuera de la memoria y desasignarlo cuando se vuelva a intercambiar hacia la memoria.

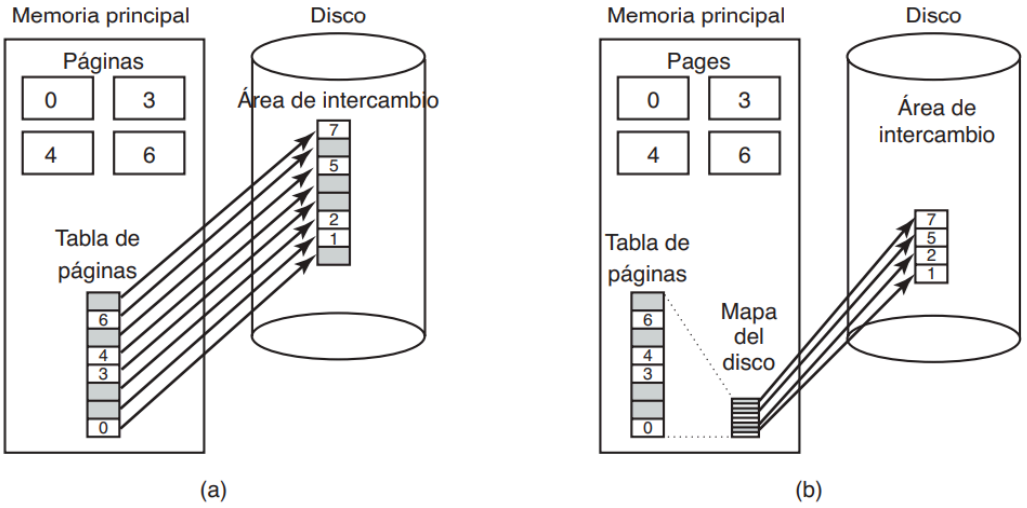


Figura 3-29. (a) Paginación a un área de intercambio estática. (b) Respaldo de páginas en forma dinámica.

6. Separación de política y mecanismo

1. Un manejador de la MMU de bajo nivel.
2. Un manejador de fallos de página que forma parte del kernel.
3. Un paginador externo que se ejecuta en espacio de usuario

6.1. Todos los detalles acerca del funcionamiento de la MMU están encapsulados en el manejador de la MMU, que es código dependiente de la máquina y tiene que volver a escribirse para cada nueva plataforma a la que se porte el sistema operativo. El manejador de fallos de página es código independiente de la máquina y contiene la mayor parte del mecanismo para la paginación. La política se determina en gran parte mediante el paginador externo, que se ejecuta como un proceso de usuario.

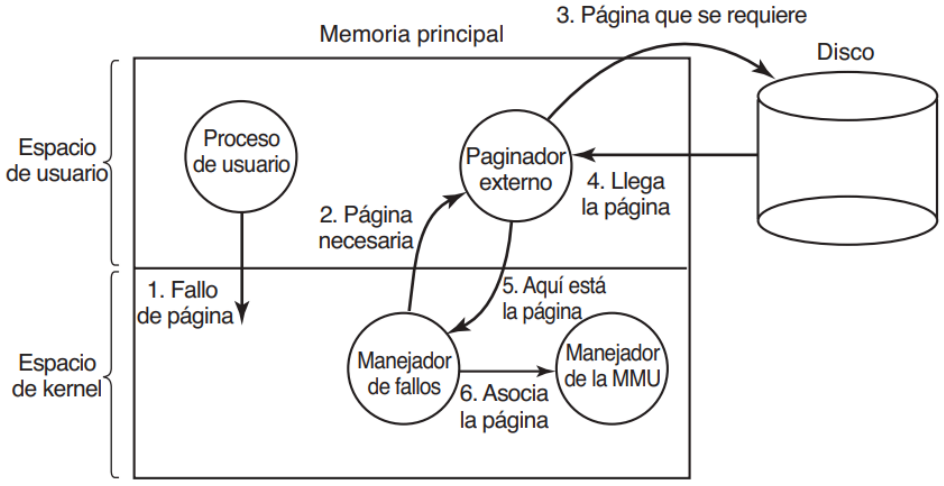


Figura 3-30. Manejo de fallos de página con un paginador externo.

6.2. El problema principal es que el paginador externo no tiene acceso a los bits R y M de todas las páginas. Estos bits desempeñan un papel en muchos de los algoritmos de paginación. Por ende, se necesita algún mecanismo para pasar esta información al paginador externo o el algoritmo de reemplazo de páginas debe ir en el kernel.

6.3. La principal ventaja de esta implementación es que se obtiene un código más modular y una mayor flexibilidad. La principal desventaja es la sobrecarga adicional de cruzar el límite entre usuario y kernel varias veces, y la sobrecarga de los diversos mensajes que se envían entre las partes del sistema.

- SEGMENTACIÓN

Un compilador tiene muchas tablas que se generan a medida que procede la compilación, las cuales posiblemente incluyen:

1. El texto del código fuente que se guarda para el listado impreso (en sistemas de procesamiento por lotes).

2. La tabla de símbolos, que contiene los nombres y atributos de las variables.

3. La tabla que contiene todas las constantes enteras y de punto flotante utilizadas.

4. El árbol de análisis sintáctico, que contiene el análisis sintáctico del programa.

5. La pila utilizada para las llamadas a procedimientos dentro del compilador

Cada una de las primeras cuatro tablas crece en forma continua a medida que procede la compilación. La última crece y se reduce de maneras impredecibles durante la compilación.

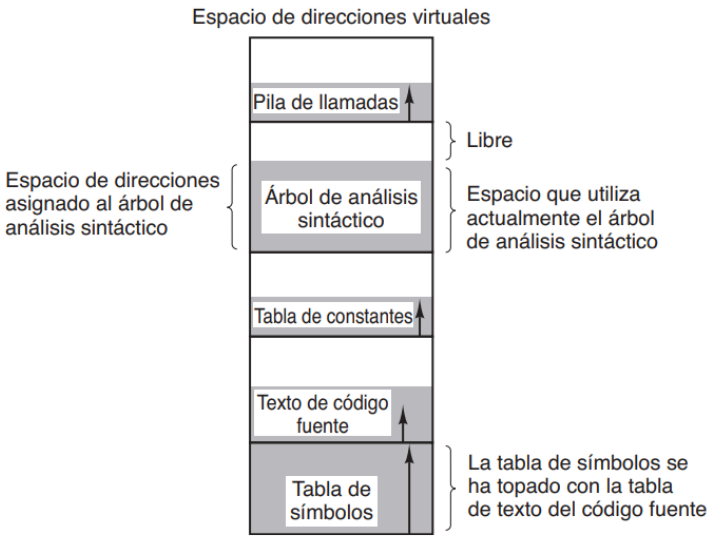


Figura 3-31. En un espacio de direcciones unidimensional con tablas que aumentan de tamaño, una tabla puede toparse con otra.

Lo que se necesita realmente es una forma de liberar al programador de tener que administrar las tablas en expansión y contracción, de la misma forma que la memoria virtual elimina la preocupación de tener que organizar el programa en sobrepuestos.

Una solución simple y en extremado general es proporcionar la máquina con muchos espacios de direcciones por completo independientes, llamados segmentos. Cada segmento consiste en una secuencia lineal de direcciones, desde 0 hasta cierto valor máximo. La longitud de cada segmento puede ser cualquier valor desde 0 hasta el máximo permitido.

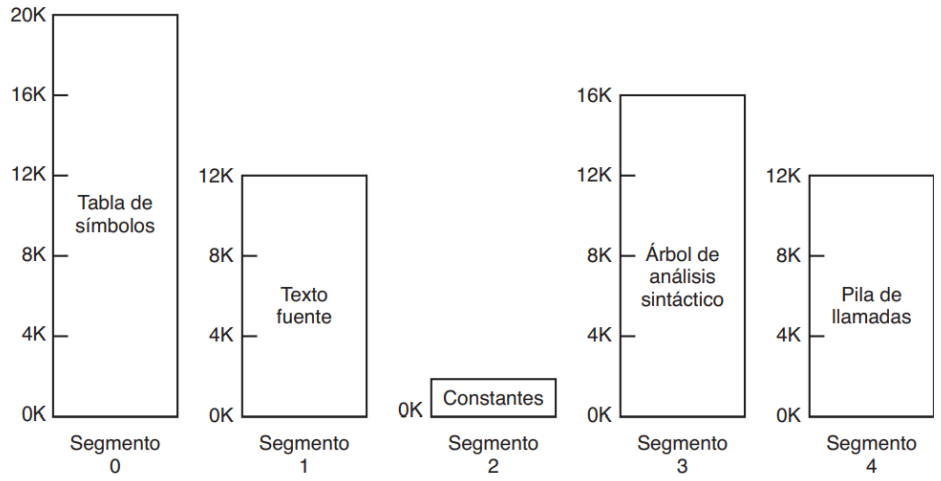


Figura 3-32. Una memoria segmentada permite que cada tabla crezca o se reduzca de manera independiente a las otras tablas.

1. Implementación de segmentación pura

Consideración	Paginación	Segmentación
¿Necesita el programador estar consciente de que se está utilizando esta técnica?	No	Sí
¿Cuántos espacios de direcciones lineales hay?	1	Muchos
¿Puede el espacio de direcciones total exceder al tamaño de la memoria física?	Sí	Sí
¿Pueden los procedimientos y los datos diferenciarse y protegerse por separado?	No	Sí
¿Pueden las tablas cuyo tamaño fluctúa acomodarse con facilidad?	No	Sí
¿Se facilita la compartición de procedimientos entre usuarios?	No	Sí
¿Por qué se inventó esta técnica?	Para obtener un gran espacio de direcciones lineal sin tener que comprar más memoria física	Para permitir a los programas y datos dividirse en espacios de direcciones lógicamente independientes, ayudando a la compartición y la protección

1.1. Una vez que el sistema haya estado en ejecución por un tiempo, la memoria se dividirá en un número de trozos, de los cuales algunos contendrán segmentos y otros huecos. Este fenómeno, llamado efecto de tablero de ajedrez o fragmentación externa

2. Segmentación con paginación: MULTICS

2.1. MULTICS operaba en las máquinas Honeywell 6000 y sus descendientes; proveía a cada programa con una memoria virtual de hasta 218 segmentos (más de 250,000), cada uno de los cuales podría ser de hasta 65,536 palabras (36 bits) de longitud.

2.2. Si los segmentos son extensos, puede ser inconveniente (o incluso imposible) mantenerlos completos en memoria principal. Esto nos lleva a la idea de paginarlos, de manera que sólo las páginas que realmente se necesiten tengan que estar presentes.

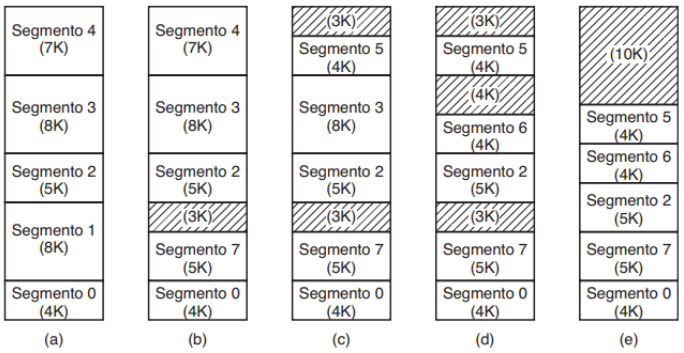


Figura 3-34. (a)-(d) Desarrollo del efecto de tablero de ajedrez. (e) Eliminación del efecto de tablero de ajedrez mediante la compactación.

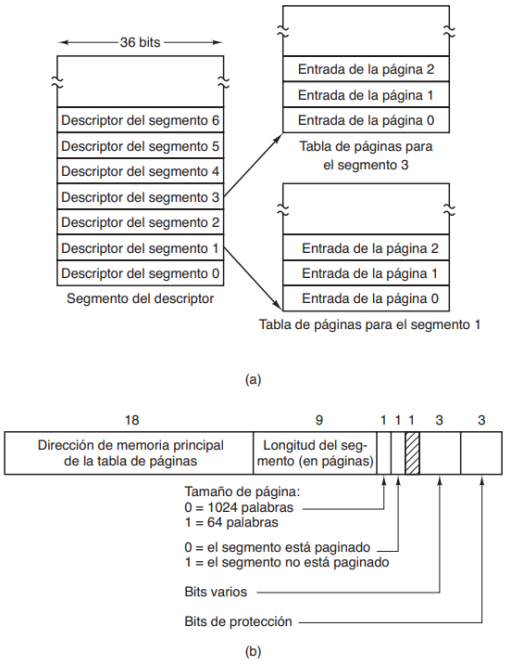


Figura 3-35. La memoria virtual de MULTICS. (a) El segmento del descriptor apunta a las tablas de páginas. (b) Un descriptor de segmento. Los números son las longitudes de los campos.

2.3. Cuando ocurre una referencia a memoria, se lleva a cabo el siguiente algoritmo.

1. El número de segmento se utiliza para encontrar el descriptor de segmentos.

2. Se realiza una comprobación para ver si la tabla de páginas del segmento está en la memoria. Si la tabla de páginas está en memoria, se localiza. Si no, ocurre un fallo de segmento. Si hay una violación a la protección, ocurre un fallo (trap).

3. La entrada en la tabla de páginas para la página virtual solicitada se examina. Si la página en sí no está en memoria, se dispara un fallo de página. Si está en memoria, la dirección de la memoria principal del inicio de la página se extrae de la entrada en la tabla de páginas.

4. El desplazamiento se agrega al origen de la página para obtener la dirección de memoria principal en donde se encuentra la palabra.

5. Finalmente se lleva a cabo la operación de lectura o almacenamiento.

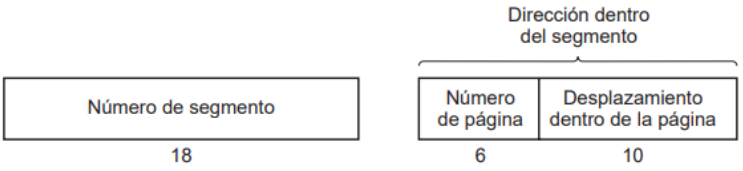


Figura 3-36. Una dirección virtual de MULTICS de 34 bits.

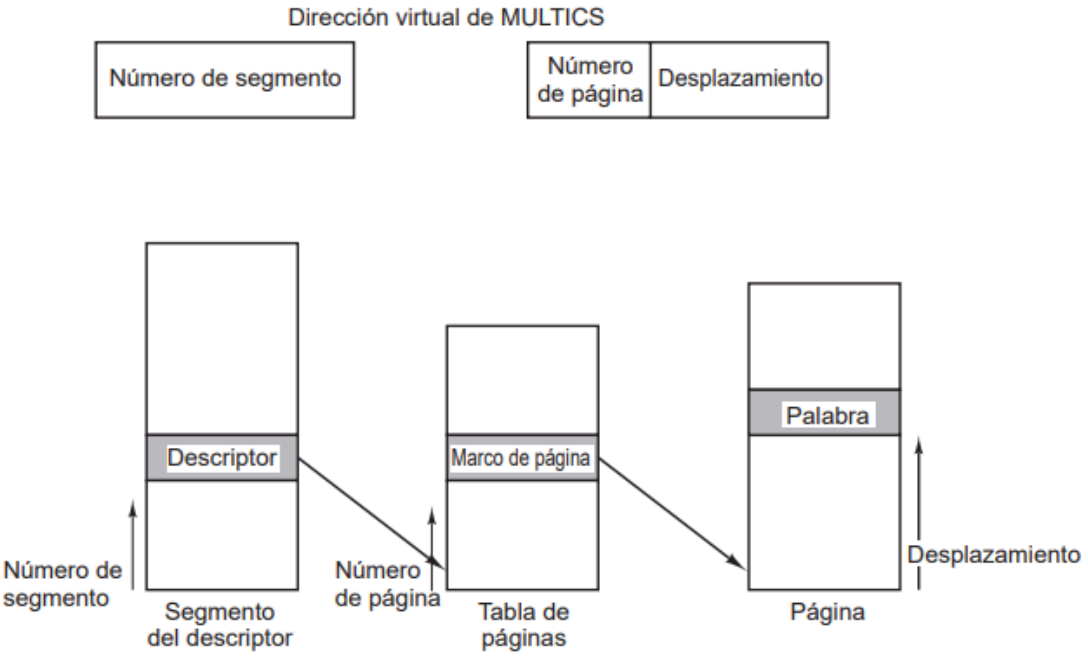


Figura 3-37. Conversión de una dirección MULTICS de dos partes en una dirección de memoria principal.

Campo de comparación					¿Está en uso esta entrada?
Número de segmento	Página virtual	Marco de página	Protección	Edad	
4	1	7	Lectura/escritura	13	1
6	0	2	Sólo lectura	10	1
12	3	1	Lectura/escritura	2	1
					0
2	1	0	Sólo ejecución	7	1
2	2	12	Sólo ejecución	9	1

Figura 3-38. Una versión simplificada del TLB de MULTICS. La existencia de dos tamaños de página hace que el TLB real sea más complicado.

3. Segmentación con paginación: Intel Pentium

3.1. La memoria virtual en el Pentium se asemeja en muchas formas a MULTICS, incluyendo la presencia de segmentación y paginación. Mientras que MULTICS tiene 256K segmentos independientes, cada uno con hasta 64K palabras de 36 bits, el Pentium tiene 16K segmentos independientes, cada uno de los cuales contiene hasta un mil millones de palabras de 32 bits. Aunque hay menos segmentos, entre mayor sea el tamaño del segmento será más importante, ya que pocos programas necesitan más de 1000 segmentos, pero muchos programas necesitan segmentos extensos.

3.2. El corazón de la memoria virtual del Pentium consiste en dos tablas, llamadas LDT (Local Descriptor Table, Tabla de descriptores locales) y GDT (Global Descriptor Table, Tabla de descriptores globales). Cada programa tiene su propia LDT, pero hay una sola GDT compartida por todos los programas en la computadora. La LDT describe los segmentos locales para cada programa, incluyendo su código, datos, pila, etcétera, mientras que la GDT describe los segmentos del sistema, incluyendo el sistema operativo en sí.

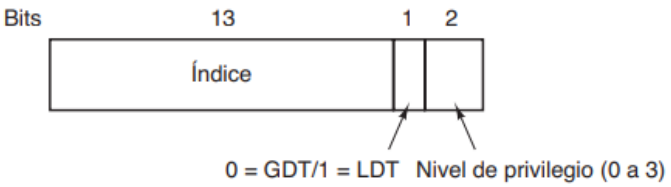


Figura 3-39. Un selector del Pentium.

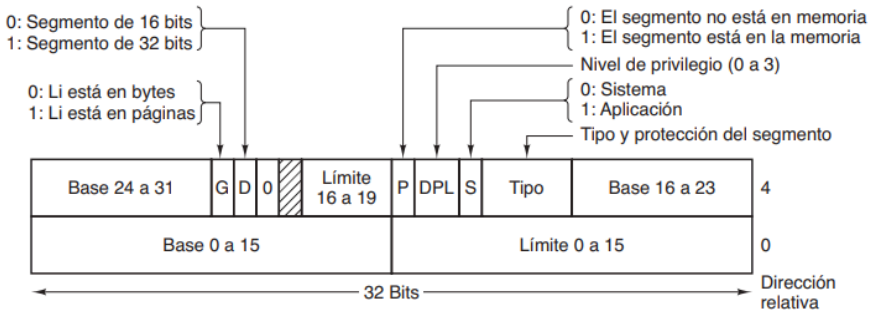


Figura 3-40. Descriptor del segmento de código del Pentium. Los segmentos de datos difieren un poco.

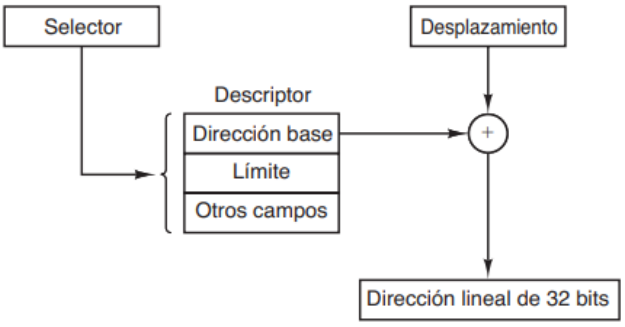


Figura 3-41. Conversión de un par (selector, desplazamiento) en una dirección lineal.

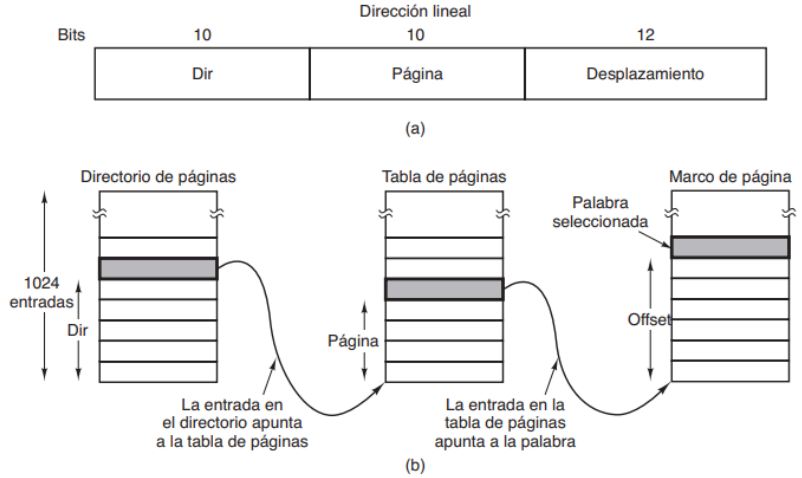


Figura 3-42. Asociación de una dirección lineal a una dirección física.

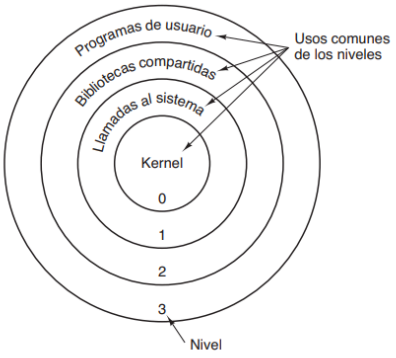


Figura 3-43. La protección en el Pentium.