Lección 8

Administración de la Memoria (cont.)

3.3 MEMORIA VIRTUAL

- Mientras que los registros base y límite se pueden utilizar para crear la abstracción de los espacios de direcciones, hay otro problema que se tiene que resolver: la administración del agrandamiento del software (*bloatware*). Aunque el tamaño de las memorias se incrementa con cierta rapidez, el del software aumenta con una mucha mayor.
- Existe la necesidad de ejecutar programas que son demasiado grandes como para caber en la memoria y sin duda existe también la necesidad de tener sistemas que puedan soportar varios programas ejecutándose al mismo tiempo, cada uno de los cuales cabe en memoria, pero que en forma colectiva exceden el tamaño de la misma. El intercambio no es una opción atractiva, ya que un disco SATA ordinario tiene una velocidad de transferencia pico de 100 MB/segundo a lo más, lo cual significa que requiere por lo menos 10 segundos para intercambiar un programa de 1 GB de memoria a disco y otros 10 segundos para intercambiar un programa de 1 GB del disco a memoria.
- Una solución que se adoptó en la década de 1960 fue dividir los programas en pequeñas partes, conocidas como **sobrepuestos** (***overlays***). Cuando empezaba un programa, todo lo que se cargaba en memoria era el administrador de sobrepuestos, que de inmediato cargaba y ejecutaba el sobrepuesto 0; cuando éste terminaba, indicaba al administrador de sobrepuestos que cargara el 1 encima del sobrepuesto 0 en la memoria (si había espacio) o encima del mismo (si no había espacio). Aunque el trabajo real de intercambiar sobrepuestos hacia adentro y hacia afuera de la memoria lo realizaba el sistema operativo, el de dividir el programa en partes tenía que realizarlo el programador en forma manual.
- 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.

- **3.3.1 Paginación**
- La mayor parte de los sistemas de memoria virtual utilizan una técnica llamada **paginación**, que describiremos a continuación. En cualquier computadora, los programas hacen referencia a un conjunto de direcciones de memoria. Cuando un programa ejecuta una instrucción como
  - MOV REG, 1000
- lo hace para copiar el contenido de la dirección de memoria 1000 a REG (o viceversa, dependiendo de la computadora). 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**. En las computadoras sin memoria virtual, la dirección física se coloca directamente en el bus de memoria y hace que se lea o escriba la palabra de memoria física con la misma dirección. Cuando se utiliza memoria virtual, las direcciones virtuales no van directamente al bus de memoria. En vez de ello, van a una **MMU** (*Memory Managemente Unit*, Unidad de administración de memoria) que asocia las direcciones virtuales a las direcciones de memoria físicas, como se ilustra en la figura 3-8.

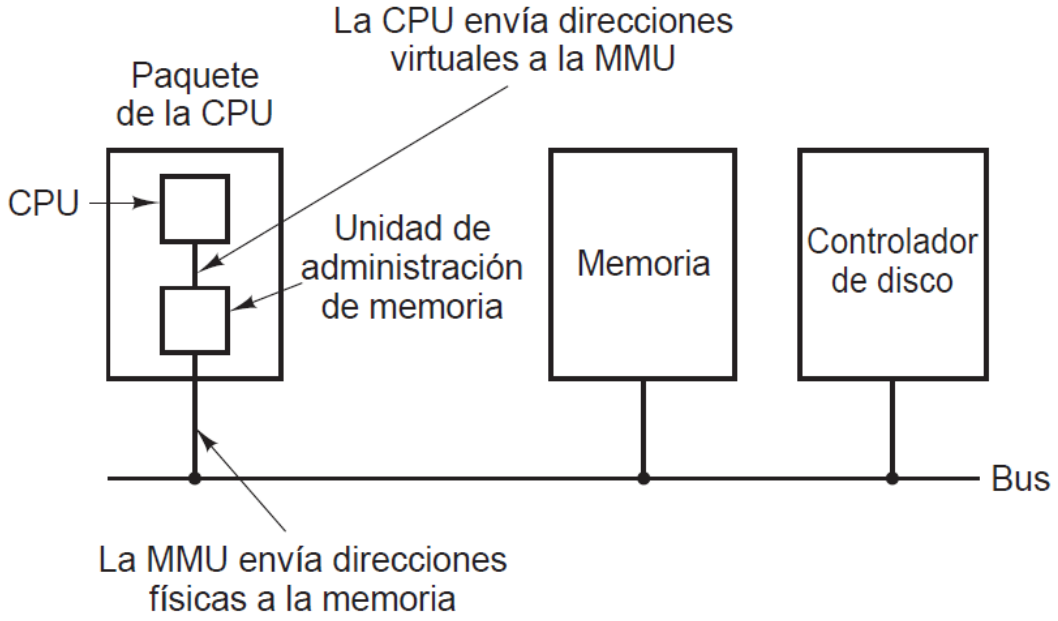

Figura 3-8. La posición y función de la MMU. Aquí la MMU se muestra como parte del chip de CPU, debido a que es común esta configuración en la actualidad. Sin embargo, lógicamente podría ser un chip separado y lo era hace años.

En este ejemplo, tenemos una computadora que genera direcciones de 16 bits, desde 0 hasta 64 K. Éstas son las direcciones virtuales. Sin embargo, esta computadora sólo tiene 32 KB de memoria física. Así, aunque se pueden escribir programas de 64 KB, no se pueden cargar completos en memoria y ejecutarse.

No obstante, una copia completa de la imagen básica de un programa, de hasta 64 KB, debe estar presente en el disco para que las partes se puedan traer a la memoria según sea necesario.

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

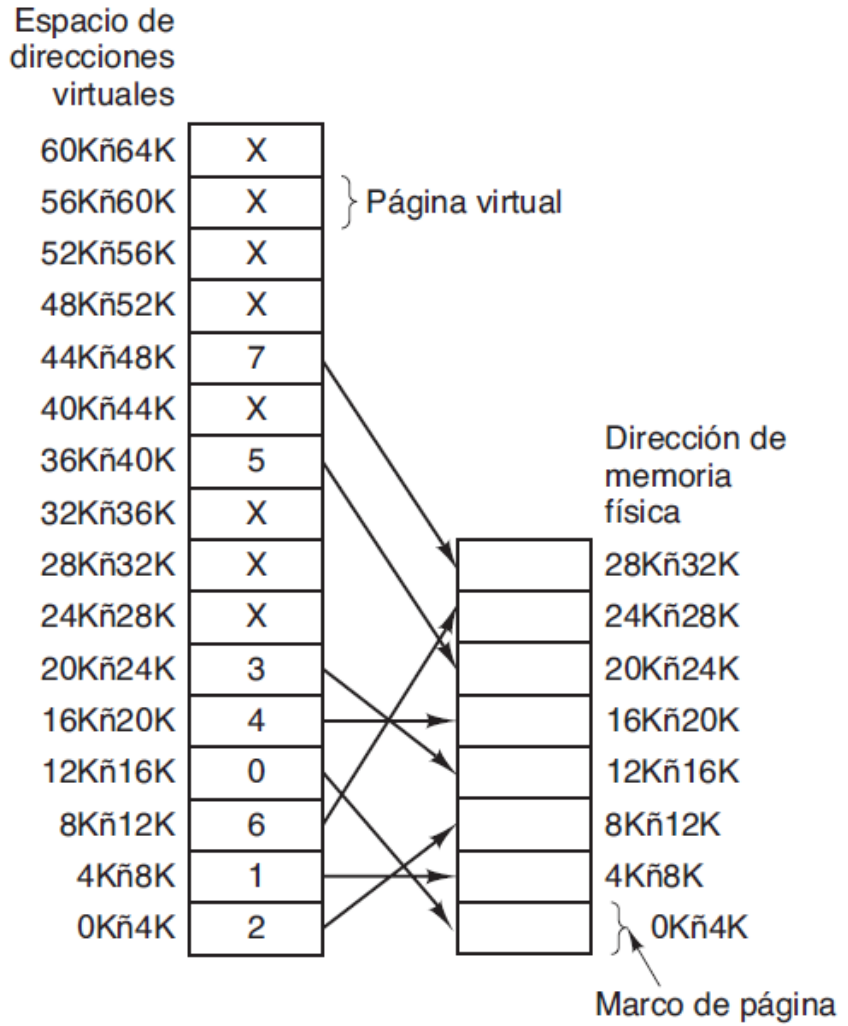

Figura 3-9. La relación entre las direcciones virtuales y las direcciones de memoria física está dada por la tabla de páginas. Cada página empieza en un múltiplo de 4096 y termina 4095 direcciones más arriba, por lo que de 4 K a 8 K en realidad significa de 4096 a 8191 y de 8 K a 12 K significa de 8192 a 12287.

En la figura 3-10 vemos un ejemplo de una dirección virtual, 8196 (0010000000000100 en binario), que se va a asociar usando la asociación de la MMU en la figura 3-9. La dirección virtual entrante de 16 bits se divide en un número de página de 4 bits y en un desplazamiento de 12 bits. Con 4 bits para el número de página, podemos tener 16 páginas y con los 12 bits para el desplazamiento, podemos direccionar todos los 4096 bytes dentro de una página.

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.

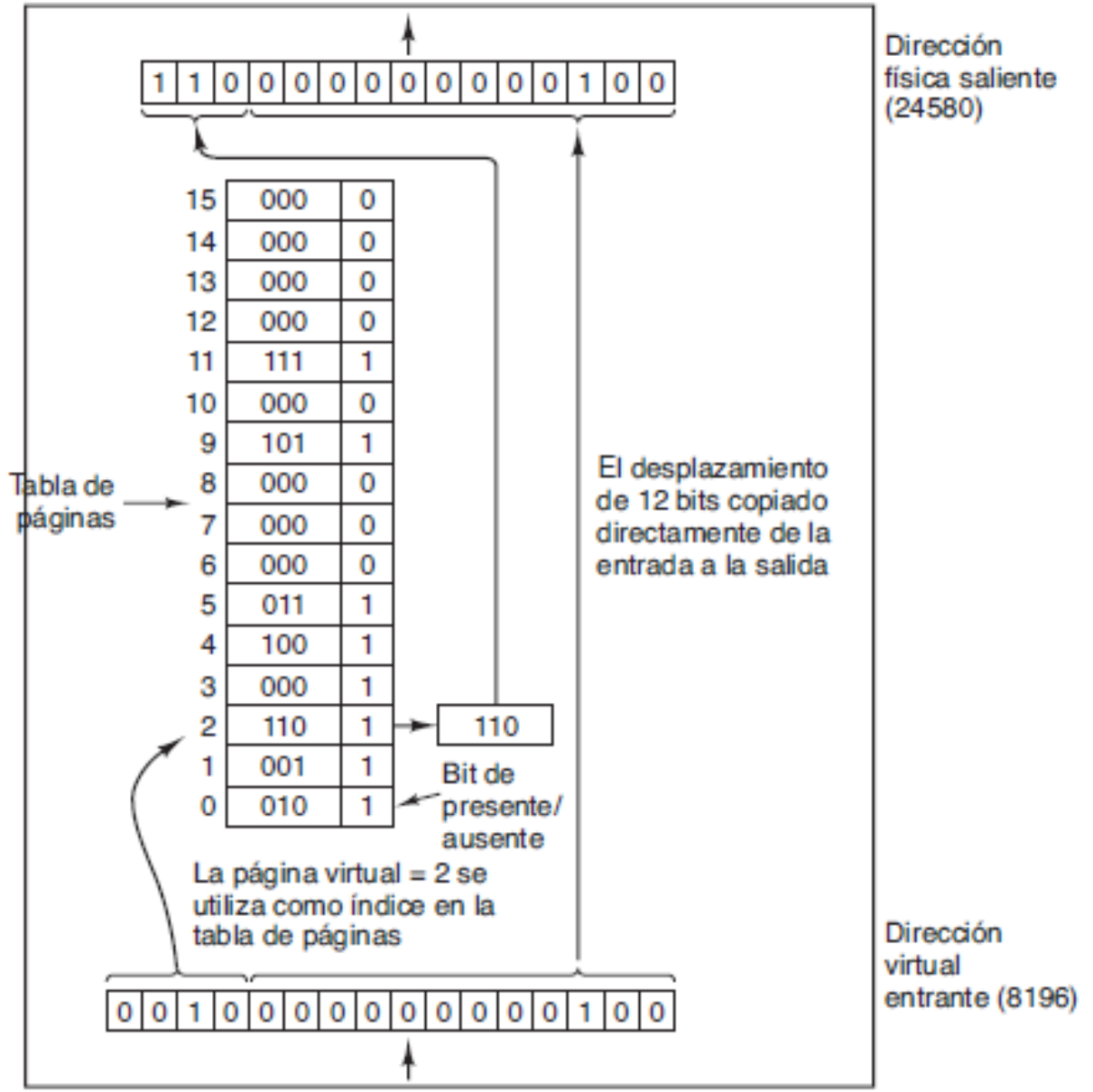

Figura 3-10. La operación interna de la MMU con 16 páginas de 4 KB.

- **3.3.2 Tablas de páginas**
- En una implementación simple, la asociación de direcciones virtuales a direcciones físicas se puede resumir de la siguiente manera: la dirección virtual se divide en un número de página virtual (bits de mayor orden) y en un desplazamiento (bits de menor orden). Por ejemplo, con una dirección de 16 bits y un tamaño de página de 4 KB, los 4 bits superiores podrían especificar una de las 16 páginas virtuales y los 12 bits inferiores podrían entonces especificar el desplazamiento de bytes (0 a 4095) dentro de la página seleccionada. Sin embargo, también es posible una división con 3, 5 u otro número de bits para la página. Las distintas divisiones implican diferentes tamaños de página.
- El número de página virtual se utiliza como índice en la tabla de páginas para buscar la entrada para esa página virtual. En la entrada en la tabla de páginas, se encuentra el número de marco de página (si lo hay). El número del marco de página se adjunta al extremo de mayor orden del desplazamiento, reemplazando el número de página virtual, para formar una dirección física que se pueda enviar a la memoria. Por ende, el propósito de la tabla de páginas es asociar páginas virtuales a los marcos de página.
- Hablando en sentido matemático, la tabla de páginas es una función donde el número de página virtual es un argumento y el número de marco físico es un resultado. Utilizando el resultado de esta función, el campo de la página virtual en una dirección virtual se puede reemplazar por un campo de marco de página, formando así una dirección de memoria física.

- **Estructura de una entrada en la tabla de páginas**
- Ahora vamos a pasar de la estructura de las tablas de páginas en general a los detalles de una sola entrada en la tabla de páginas. La distribución exacta de una entrada depende en gran parte de la máquina, pero el tipo de información presente es aproximadamente el mismo de una máquina a otra.
- En la figura 3-11 proporcionamos un ejemplo de una entrada en la tabla de páginas. El tamaño varía de una computadora a otra, pero 32 bits es un tamaño común. El campo más importante es el *número de marco de página*. Después de todo, el objetivo de la asociación de páginas es mostrar este valor. Enseguida de este campo tenemos el bit de *presente/ausente*. Si este bit es 1, la entrada es válida y se puede utilizar. Si es 0, la página virtual a la que pertenece la entrada no se encuentra actualmente en la memoria. Al acceder a una entrada en la tabla de página con este bit puesto en 0 se produce un fallo de página.

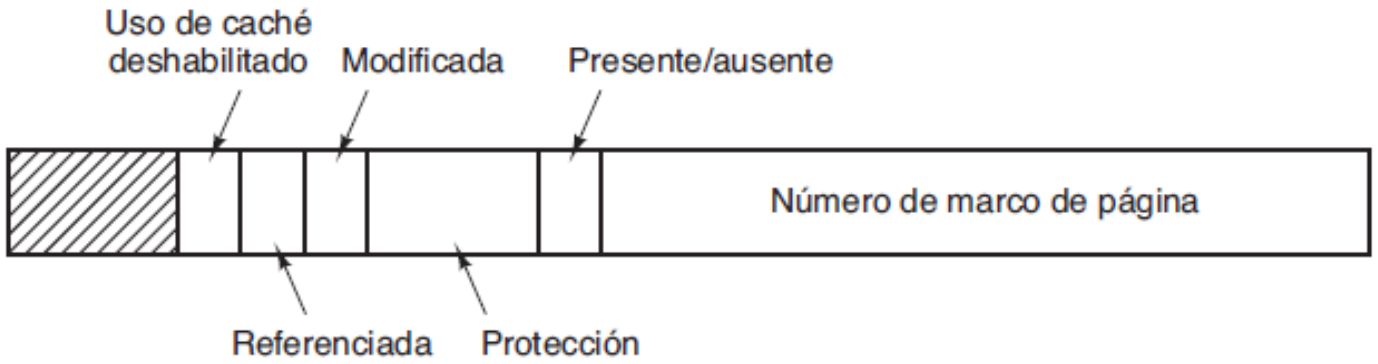

Figura 3-11. Una típica entrada en la tabla de páginas.

- **3.3.3 Aceleración de la paginación**
- En cualquier sistema de paginación hay que abordar dos cuestiones principales:

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

- El primer punto es una consecuencia del hecho de que la asociación virtual-a-física debe realizarse en cada referencia de memoria. Todas las instrucciones deben provenir finalmente de la memoria y muchas de ellas hacen referencias a operandos en memoria también. En consecuencia, es necesario hacer una, dos o algunas veces más referencias a la tabla de páginas por instrucción. 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.
- El segundo punto se deriva del hecho de que todas las computadoras modernas utilizan direcciones virtuales de por lo menos 32 bits, donde 64 bits se vuelven cada vez más comunes. Por decir, con un tamaño de página de 4 KB, un espacio de direcciones de 32 bits tiene 1 millón de páginas y un espacio de direcciones de 64 bits tiene más de las que desearíamos contemplar. 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).

- **Búferes de traducción adelantada**
- Ahora veamos esquemas implementados ampliamente para acelerar la paginación y manejar espacios de direcciones virtuales extensos, empezando con la aceleración de la paginación. El punto inicial de la mayor parte de las técnicas de optimización es que la tabla de páginas está en la memoria.
- Potencialmente, este diseño tiene un enorme impacto sobre el rendimiento. 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. Como la velocidad de ejecución está comúnmente limitada por la proporción a la que la CPU puede obtener instrucciones y datos de la memoria, al tener que hacer dos referencias a memoria por cada una de ellas se reduce el rendimiento a la mitad. Bajo estas condiciones, nadie utilizaría la paginación.
- Los diseñadores de computadoras han sabido acerca de este problema durante años y han ideado una solución que está basada en la observación de que la mayor parte de los programas tienden a hacer un gran número de referencias a un pequeño número de páginas y no viceversa. Por ende, sólo se lee con mucha frecuencia una pequeña fracción de las entradas en la tabla de páginas; el resto se utiliza muy pocas veces.

- 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 ilustra en la figura 3-12. Por lo general se encuentra dentro de la MMU y consiste en un pequeño número de entradas, ocho en este ejemplo, pero raras veces más de 64.
- Cada entrada contiene información acerca de una página, incluyendo el número de página virtual, un bit que se establece cuando se modifica la página, el código de protección (permisos de lectura/escritura/ejecución) y el marco de página físico en el que se encuentra la página. Estos campos tienen una correspondencia de uno a uno con los campos en la tabla de páginas, excepto por el número de página virtual, que no se necesita en la tabla de páginas. Otro bit indica si la entrada es válida (es decir, si está en uso) o no.

El hardware primero comprueba si su número de página virtual está presente en el TLB al compararla con todas las entradas en forma simultánea. 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.
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. De esta forma, si esa página se utiliza pronto otra vez, la segunda vez se producirá un acierto en el TLB en vez de un fracaso. Cuando se purga una entrada del TLB, el bit modificado se copia de vuelta a la entrada en la tabla de páginas en memoria. Los otros valores ya están ahí, excepto el bit de referencia. Cuando el TLB se carga de la tabla de páginas, todos los campos se toman de la memoria.

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.

- **Administración del TLB mediante software**
- Hasta ahora hemos supuesto que toda máquina con memoria virtual paginada tiene tablas de páginas reconocidas por el hardware, más un TLB. En este diseño, la administración y el manejo de fallas del TLB se realiza por completo mediante el hardware de la MMU. Las traps, o trampas, para el sistema operativo ocurren sólo cuando una página no se encuentra en memoria.
- Mediante Software, las entradas del TLB se cargan de manera explícita mediante el sistema operativo. Cuando no se encuentra una coincidencia en el TLB, en vez de que la MMU vaya a las tablas de páginas para buscar y obtener la referencia a la página que se necesita, sólo genera un fallo del TLB y pasa el problema al sistema operativo.
- De manera sorprendente, si el TLB es razonablemente grande (por ejemplo, de 64 entradas) para reducir la proporción de fallos, la administración del TLB mediante software resulta tener una eficiencia aceptable.
- Se han desarrollado varias estrategias para mejorar el rendimiento en equipos que realizan la administración del TLB mediante software. Un método se enfoca en reducir los fallos del TLB y el costo de un fallo del TLB cuando llega a ocurrir.

- La forma normal de procesar un fallo del TLB, ya sea en hardware o en software, es ir a la tabla de páginas y realizar las operaciones de indexado para localizar la página referenciada. 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. Al comprobar primero la caché de software, el sistema operativo puede reducir de manera substancial los fallos del TLB.

- Cuando se utiliza la administración del TLB mediante software, es esencial comprender la diferencia entre los dos tipos de fallos. 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.

- **3.3.4 Tablas de páginas para memorias extensas**
- Otro problema es cómo lidiar con espacios de direcciones virtuales muy extensos. A continuación veremos dos maneras de hacerlo.
- **Tablas de páginas multinivel**
- Como primer método, considere el uso de una **tabla de páginas multinivel**. En la figura 3-13 se muestra un ejemplo simple. En la figura 3-13(a) tenemos una dirección virtual de 32 bits que se particiona en un campo *TP1* de 10 bits, un campo *TP2* de 10 bits y un campo *Desplazamiento* de 12 bits. Como los desplazamientos son de 12 bits, las páginas son de 4 KB y hay un total de 220.
- 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. Por ejemplo, suponga que un proceso necesita 12 megabytes: los 4 megabytes inferiores de memoria para el texto del programa, lo siguientes 4 megabytes para datos y los 4 megabytes superiores para la pila. Entre la parte superior de los datos y la parte inferior de la pila hay un hueco gigantesco que no se utiliza.
- En la figura 3-13(b) podemos ver cómo funciona la tabla de página de dos niveles en este ejemplo.
- A la izquierda tenemos la tabla de páginas de nivel superior, con 1024 entradas, que corresponden al campo *TP1* de 10 bits. Cuando se presenta una dirección virtual a la MMU, primero extrae el campo *TP1* y utiliza este valor como índice en la tabla de páginas de nivel superior. Cada una de estas 1024 entradas representa 4 M, debido a que todo el espacio de direcciones virtuales de 4 gigabytes (es decir, de 32 bits) se ha dividido en trozos de 4096 bytes.

La entrada 0 de la tabla de páginas de nivel superior apunta a la tabla de páginas para el texto del programa, la entrada 1 apunta a la tabla de páginas para los datos y la entrada 1023 apunta a la tabla de páginas para la pila. Las otras entradas (sombreadas) no se utilizan. Ahora el campo *TP2* se utiliza como índice en la tabla de páginas de segundo nivel seleccionada para buscar el número de marco de página para esta página en sí.

Como ejemplo, considere la dirección virtual de 32 bits 0x00403004 (4,206,596 decimal), que se encuentra 12,292 bytes dentro de los datos. Esta dirección virtual corresponde a *TP1* = 1, *TP2* = 2 y *Desplazamiento* = 4.

La MMU utiliza primero a *TP1* para indexar en la tabla de páginas de nivel superior y obtener la entrada 1, que corresponde a las direcciones de 4M a 8M. Después utiliza *PT2* para indexar en la tabla de páginas de segundo nivel que acaba de encontrar y extrae la entrada 3, que corresponde a las direcciones de 12288 a 16383 dentro de su trozo de 4M (es decir, las direcciones absolutas de 4,206,592 a 4,210,687). Esta entrada contiene el número de marco de la página que contiene la dirección virtual 0x00403004. Si esa página no está en la memoria, el bit de *presente/ausente* en la entrada de la tabla de páginas será cero, con lo cual se producirá un fallo de página. Si la página está en la memoria, el número del marco de página que se obtiene de la tabla de páginas de segundo nivel se combina con el desplazamiento (4) para construir la dirección física. Esta dirección se coloca en el bus y se envía a la memoria.

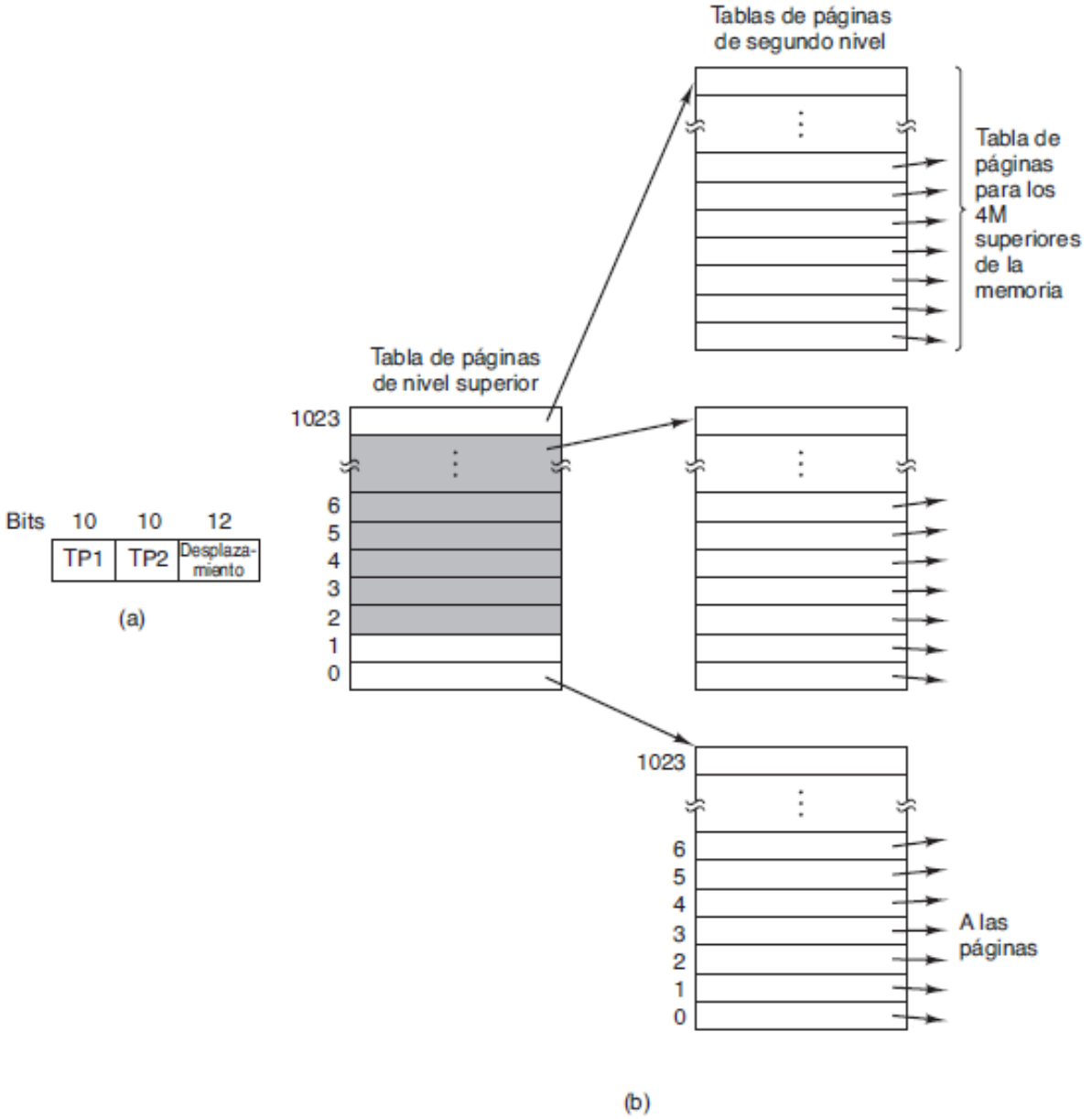

Figura 3-13. (a) Una dirección de 32 bits con dos campos de tablas de páginas. (b) Tablas de páginas de dos niveles.

- **Tablas de páginas invertidas**
- Para los espacios de direcciones virtuales de 32 bits, la tabla de páginas multinivel funciona bastante bien. Sin embargo, a medida que las computadoras de 64 bits se hacen más comunes, la situación cambia de manera drástica. Si el espacio de direcciones es de 264 bytes, con páginas de 4 KB, necesitamos una tabla de páginas con 252 entradas. Si cada entrada es de 8 bytes, la tabla es de más de 30 millones de gigabyes (30 PB). Ocupar 30 millones de gigabytes sólo para la tabla de páginas no es una buena idea.
- Una de esas soluciones es la **tabla de páginas invertida**. En este diseño hay una entrada por cada marco de página en la memoria real, en vez de tener una entrada por página de espacio de direcciones virtuales. Por ejemplo, con direcciones virtuales de 64 bits, una página de 4 KB y 1 GB de RAM, una tabla de páginas invertida sólo requiere 262,144 entradas. La entrada lleva el registro de quién (proceso, página virtual) se encuentra en el marco de página.
- Aunque las tablas de página invertidas ahorran grandes cantidades de espacio, al menos cuando el espacio de direcciones virtuales es mucho mayor que la memoria física, tienen una seria desventaja:
- la traducción de dirección virtual a dirección física se hace mucho más difícil. Cuando el proceso *n* hace referencia a la página virtual *p*, el hardware ya no puede buscar la página física usando *p* como índice en la tabla de páginas. En vez de ello, debe buscar una entrada (*n*, *p*) en toda la tabla de páginas invertida. Lo que es peor: esta búsqueda se debe realizar en cada referencia a memoria, no sólo en los fallos de página. Buscar en una tabla de 256 K en cada referencia a memoria no es la manera de hacer que la máquina sea deslumbrantemente rápida.

- La forma de salir de este dilema es utilizar el TLB. Si el TLB puede contener todas las páginas de uso frecuente, la traducción puede ocurrir con igual rapidez que con las tablas de páginas regulares.
- Sin embargo, en un fallo de TLB la tabla de páginas invertida tiene que buscarse mediante software. Una manera factible de realizar esta búsqueda es tener una tabla de hash arreglada según el hash de la dirección virtual. Todas las páginas virtuales que se encuentren en memoria y tengan el mismo valor de hash se encadenan en conjunto, como se muestra en la figura 3-14. Si la tabla de hash tiene tantas ranuras como las páginas físicas de la máquina, la cadena promedio tendrá sólo una entrada, con lo cual se acelera de manera considerable la asociación. Una vez que se ha encontrado el número de marco de página, se introduce el nuevo par (virtual, física) en el TLB.

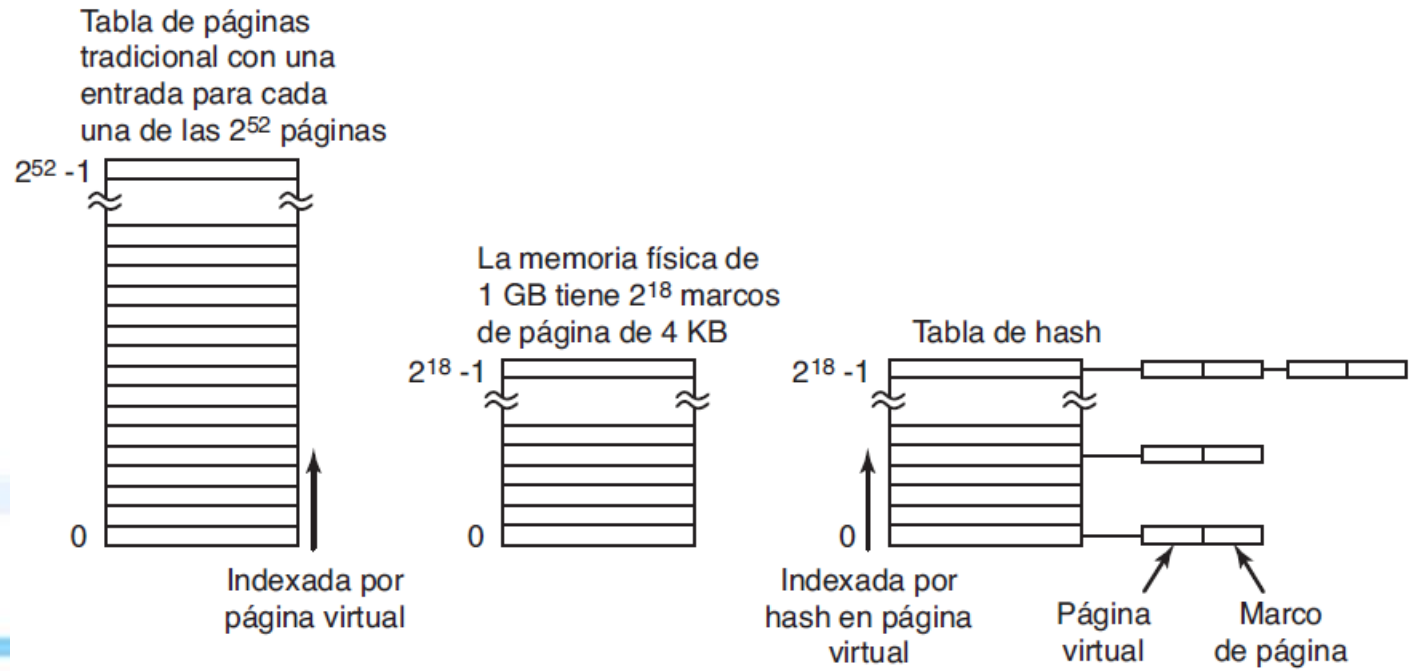

Figura 3-14. Comparación de una tabla de páginas tradicional con una tabla de páginas invertida.