siguiente proceso solicita 18462 bytes; si asignamos exactamente el bloque solicitado, nos quedará un agujero de 2 bytes. El espacio de memoria adicional requerido para llevar el control de este agujero será sustancialmente mayor que el propio agujero. La técnica general para evitar este problema consiste en descomponer la memoria física en bloques de tamaño fijo y asignar la memoria en unidades basadas en el tamaño de bloque. Con esta técnica, la memoria asignada a un proceso puede ser ligeramente superior a la memoria solicitada. La diferencia entre los dos valores será la **fragmentación interna**, es decir, la memoria que es interna a una partición pero que no está siendo utilizada.

Una solución al problema de la fragmentación externa consiste en la **compactación**. El objetivo es mover el contenido de la memoria con el fin de situar toda la memoria libre de manera contigua, para formar un único bloque de gran tamaño. Sin embargo, la compactación no siempre es posible. Si la reubicación es estática y se lleva a cabo en tiempo de ensamblado o en tiempo de carga, no podemos utilizar el mecanismo de la compactación; la compactación *sólo* es posible si la reubicación es dinámica y se lleva a cabo en tiempo de ejecución. Si las direcciones se reubican dinámicamente, la reubicación sólo requerirá mover el programa y los datos y luego cambiar el registro base para reflejar la nueva dirección base utilizada. Cuando la compactación es posible, debemos además determinar cuál es su coste. El algoritmo de compactación más simple consiste en mover todos los procesos hacia uno de los extremos de la memoria; de esta forma, todos los agujeros se moverán en la otra dirección, produciendo un único agujero de memoria disponible de gran tamaño. Sin embargo, este esquema puede ser muy caro de implementar.

Otra posible solución al problema de la fragmentación externa consiste en permitir que el espacio de direcciones lógicas de los procesos no sea contiguo, lo que hace que podamos asignar memoria física a un proceso con independencia de dónde esté situada dicha memoria. Hay dos técnicas complementarias que permiten implementar esta solución: la paginación (Sección 8.4) y la segmentación (Sección 8.6). Asimismo, estas técnicas pueden también combinarse (Sección 8.7).

8.4 Paginación

La **paginación** es un esquema de gestión de memoria que permite que el espacio de direcciones físicas de un proceso no sea contiguo. La paginación evita el considerable problema de encajar fragmentos de memoria de tamaño variable en el almacén de respaldo; la mayoría de los esquema de gestión de memoria utilizados antes de la introducción de la paginación sufrían de este problema, que surgía debido a que, cuando era necesario proceder a la descarga de algunos datos o fragmentos de código que residieran en la memoria principal, tenía que encontrarse el espacio necesario en el almacén de respaldo. El almacén de respaldo también sufre los problemas de fragmentación que hemos mencionado en relación con la memoria principal, pero con la salvedad de que el acceso es mucho más lento, lo que hace que la compactación sea imposible. Debido a sus ventajas con respecto a los métodos anteriores, la mayoría de los sistemas operativos utilizan comúnmente mecanismos de paginación de diversos tipos.

Tradicionalmente, el soporte para la paginación se gestionaba mediante hardware. Sin embargo, algunos diseños recientes implementan los mecanismos de paginación integrando estrechamente el hardware y el sistema operativo, especialmente en los microprocesadores de 64 bytes.

8.4.1 Método básico

El método básico para implementar la paginación implica descomponer la memoria física en una serie de bloques de tamaño fijo denominados **marcos** y descomponer la memoria lógica en bloques del mismo tamaño denominados **páginas**. Cuando hay que ejecutar un proceso, sus páginas se cargan desde el almacén de respaldo en los marcos de memoria disponibles. El almacén de respaldo está dividido en bloques de tamaño fijo que tienen el mismo tamaño que los marcos de memoria.

La Figura 8.7 ilustra el soporte hardware para el mecanismo de paginación. Toda dirección generada por la CPU está dividida en dos partes: un **número de página (p)** y un **desplazamiento de página (d)**. El número de página se utiliza como índice para una tabla de páginas. La tabla de

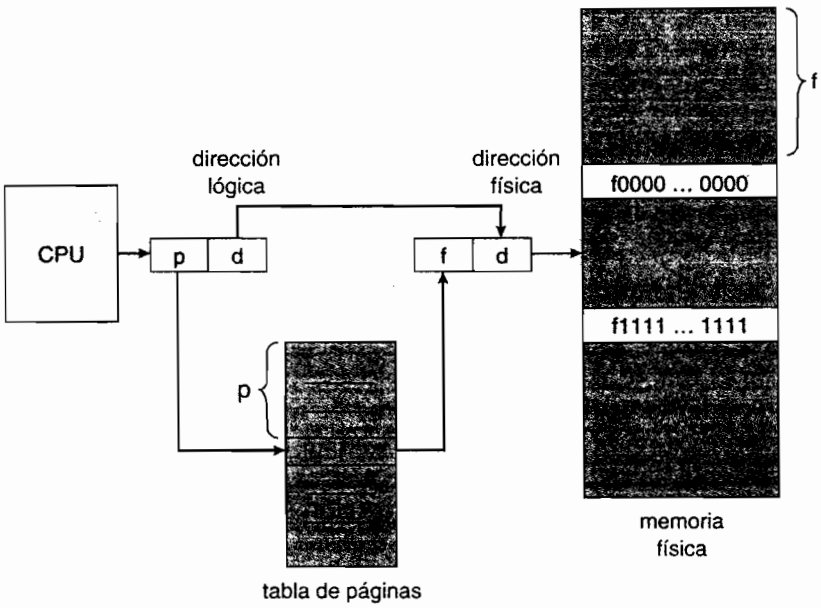


Figura 8.7 Hardware de paginación.

páginas contiene la dirección base de cada página en memoria física; esta dirección base se combina con el desplazamiento de página para definir la dirección de memoria física que se envía a la unidad de memoria. En la Figura 8.8 se muestra el modelo de paginación de la memoria.

El tamaño de página (al igual que el tamaño de marco) está definido por el hardware. El tamaño de la página es, normalmente, una potencia de 2, variando entre 512 bytes y 16 MB por página, dependiendo de la arquitectura de la computadora. La selección de una potencia de 2 como tamaño de página hace que la traducción de una dirección lógica a un número de página y a un desplazamiento de página resulte particularmente fácil. Si el tamaño del espacio de direcciones lógicas es 2^m y el tamaño de página es 2^n unidades de direccionamiento (bytes o palabras), entonces los $m - n$ bits de mayor peso de cada dirección lógica designarán el número de página, mien-

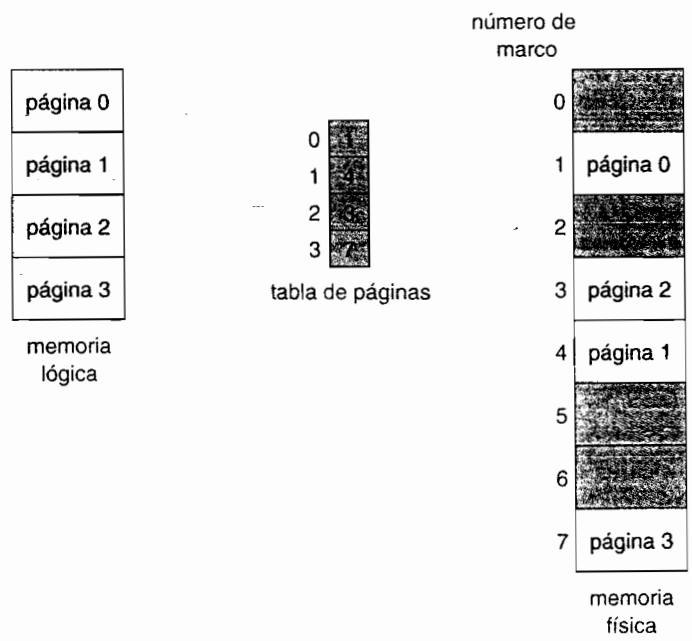
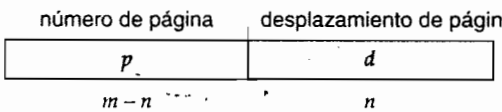


Figura 8.8 Modelo de paginación de la memoria lógica y física.

tras que los n bits de menor peso indicarán el desplazamiento de página. Por tanto, la dirección lógica tiene la estructura siguiente:



donde p es un índice de la tabla de páginas y d es el desplazamiento dentro de la página.

Como ejemplo concreto (aunque minúsculo), considere la memoria mostrada en la Figura 8.9. Utilizando un tamaño de página de 4 bytes y una memoria física de 32 bytes (8 páginas), podemos ver cómo se hace corresponder la memoria física con la visión de la memoria que tiene el usuario. La dirección lógica 0 representa la página 0, desplazamiento 0. Realizando la indexación en la tabla de páginas, vemos que la página 0 se encuentra en el marco 5. Por tanto, la dirección lógica 0 se hace corresponder con la dirección física 20 (= (5 × 4) + 0). La dirección lógica 3 (página 0, desplazamiento 3) se corresponde con la dirección física 23 (= (5 × 4) + 3). La dirección lógica 4 corresponderá a la página 1, desplazamiento 0; de acuerdo con la tabla de páginas, la página 1 se corresponde con el marco 6. Por tanto, la dirección lógica 4 corresponde a la dirección física 24 (= (6 × 4) + 0). La dirección lógica 13 corresponderá, por el mismo procedimiento, a la dirección física 9.

El lector puede haberse dado cuenta de que el propio esquema de paginación es una forma de reubicación dinámica. Cada dirección lógica es asignada por el hardware de paginación a alguna dirección física. La utilización de la paginación es similar al uso de una tabla de registros base (o de reubicación), uno por cada marco de memoria.

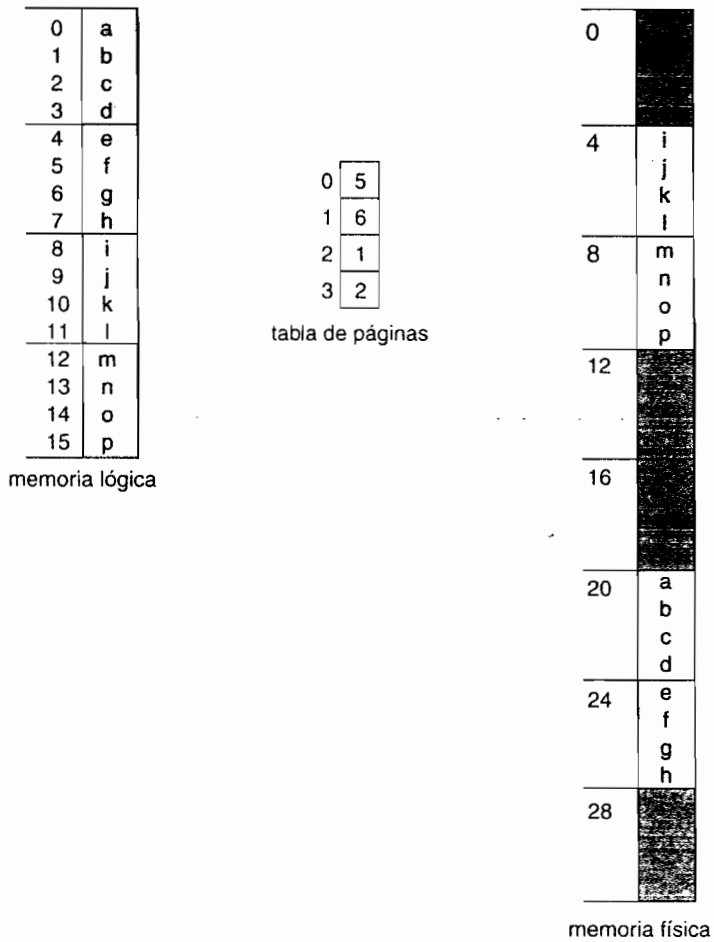


Figura 8.9 Ejemplo de paginación para una memoria de 32 bytes con páginas de 4 bytes.

Cuando usamos un esquema de paginación, no tenemos fragmentación externa: *todos* los marcos libres podrán ser asignados a un proceso que los necesite. Sin embargo, lo que sí podemos tener es un cierto grado de fragmentación interna, ya que los marcos se asignan como unidades y, si los requisitos de memoria de un proceso no coinciden exactamente con las fronteras de página, el *último* marco asignado puede no estar completamente lleno. Por ejemplo, si el tamaño de página es de 2.048 bytes, un proceso de 72.226 bytes necesitará 35 páginas completas más 1.086 bytes. A ese proceso se le asignarían 36 marcos, lo que daría como resultado una fragmentación interna de 2,048 - 1.086 = 962 bytes. En el peor de los casos, un proceso podría necesitar n páginas más 1 byte, por lo que se le asignarían $n + 1$ marcos, dando como resultado una fragmentación interna de tamaño prácticamente igual a un marco completo.

Si el tamaño de los procesos es independiente del tamaño de las páginas, podemos esperar que la fragmentación interna sea, como promedio, igual a media página por cada proceso. Esta consideración sugiere que conviene utilizar tamaños de página pequeños. Sin embargo, se necesita dedicar recursos a la gestión de cada una de las entradas de la tabla de páginas, y estos recursos adicionales se reducen a medida que se incrementa el tamaño de la página. Asimismo, las operaciones de E/S de disco son más eficientes cuanto mayor sea el número de datos transferidos (Capítulo 12). Generalmente, los tamaños de página utilizados en la práctica han ido creciendo con el tiempo, a medida que los procesos, los conjuntos de datos y la memoria principal han aumentado de tamaño. Hoy en día, las páginas utilizadas se encuentran normalmente entre 4 KB y 8 KB de tamaño y algunos sistemas soportan páginas de tamaño mayor aún. Algunos procesadores y algunos *kernels* soportan incluso tamaños de página múltiples. Por ejemplo, Solaris utiliza tamaños de página de 8 KB y 4 MB, dependiendo de los datos almacenados por las páginas. Los investigadores están actualmente desarrollando mecanismos de soporte con tamaños de página variables, que pueden ajustarse sobre la marcha.

Usualmente, cada entrada de una tabla de páginas tiene 4 bytes de longitud, pero también ese tamaño puede variar. Una entrada de 32 bits puede apuntar a una de 2^{32} marcos de página físicos. Si el tamaño del marco es de 4 KB, entonces un sistema con entradas de 4 bytes podrá direccionar 2^{44} bytes (o 16 TB) de memoria física.

Cuando llega un proceso al sistema para ejecutarlo, se examina su tamaño expresado en páginas. Cada página del proceso necesitará un marco. Por tanto, si el proceso requiere n páginas, deberá haber disponibles al menos n marcos en memoria. Si hay disponibles n marcos, se los asignará al proceso que acaba de llegar. La primera página del proceso se carga en uno de los marcos asignados y se incluye el número de marco en la tabla de páginas para este proceso. La siguiente página se carga en otro marco y su número de marco se coloca en la tabla de páginas, y así sucesivamente (Figura 8.10).

Un aspecto importante de la paginación es la clara separación existente entre la visión de la memoria que tiene el usuario y la memoria física real. El programa de usuario ve la memoria como un único espacio que sólo contiene ese programa. En la práctica, el programa de usuario está disperso por toda la memoria física, lo que también sucede para los restantes programas. La diferencia entre la visión de la memoria que tiene el usuario y la memoria física real se resuelve mediante el hardware de traducción de direcciones. Las direcciones lógicas se traducen a direcciones físicas y esta conversión queda oculta a ojos del usuario, siendo controlado por el sistema operativo. Observe que los procesos de usuario son incapaces, por definición, de acceder a la memoria que no les pertenece. No tienen forma de direccionar la memoria situada fuera de su tabla de páginas y esa tabla incluye únicamente aquellas páginas que sean propiedad del proceso.

Puesto que el sistema operativo está gestionando la memoria física, debe ser consciente de los detalles relativos a la asignación de la memoria física: qué marcos han sido asignados, qué marcos están disponibles, cuál es el número total de marcos, etc. Esta información se suele mantener en una estructura de datos denominada **tabla de marcos**. La tabla de marcos tiene una entrada por cada marco físico que indica si está libre o asignado y, en caso de estar asignado, a qué página de qué proceso o procesos ha sido asignado.

Además, el sistema operativo debe ser consciente de que los procesos de usuario operan en el espacio de usuario y de que todas las direcciones lógicas deben ser convertidas con el fin de generar direcciones físicas. Si un usuario hace una llamada al sistema (por ejemplo, para realizar una

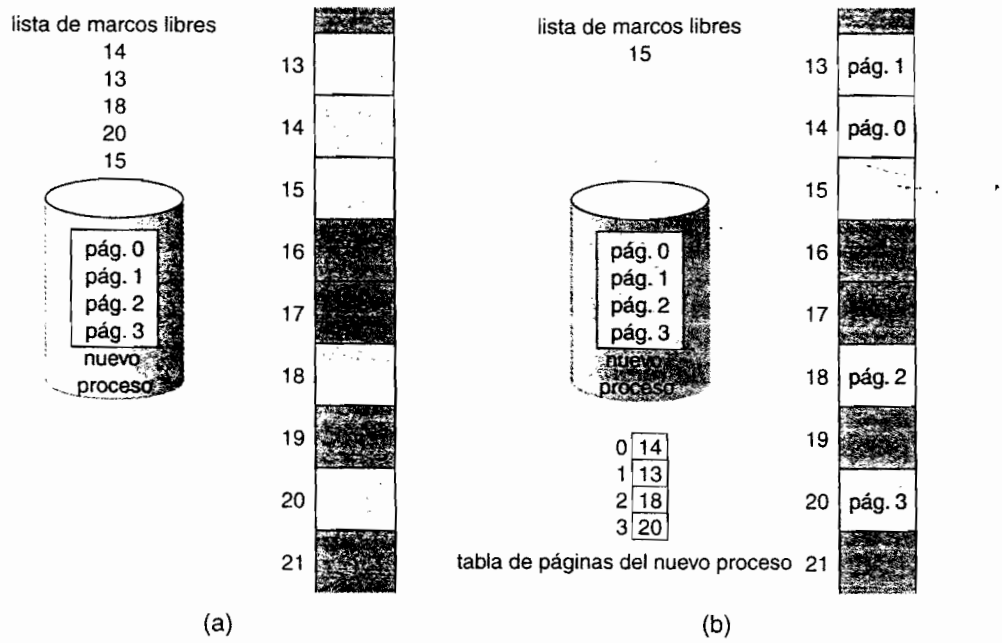


Figura 8.10 Marcos libres (a) antes de la asignación y (b) después de la asignación.

operación de E/S) y proporciona una dirección como parámetro (por ejemplo un búfer), dicha dirección deberá ser convertida para obtener la dirección física correcta. El sistema operativo mantiene una copia de la tabla de páginas de cada proceso, al igual que mantiene una copia del contador de instrucciones y de los contenidos de los registros. Esta copia se utiliza para traducir las direcciones lógicas a direcciones físicas cada vez que el sistema operativo deba convertir manualmente una dirección lógica a una dirección física. El despachador de la CPU utiliza también esa copia para definir la tabla de páginas hardware en el momento de asignar la CPU a un proceso. Los mecanismos de paginación incrementan, por tanto, el tiempo necesario para el cambio de contexto.

8.4.2 Soporte hardware

Cada sistema operativo tiene sus propios métodos para almacenar tablas de páginas. La mayoría de ellos asignan una tabla de páginas para cada proceso, almacenándose un puntero a la tabla de páginas, junto con los otros valores de los registros (como por ejemplo el contador de instrucciones), en el bloque de control del proceso. Cuando se le dice al despachador que inicie un proceso, debe volver a cargar los registros de usuario y definir los valores correctos de la tabla de páginas hardware a partir de la tabla almacenada de páginas de usuario.

La implementación hardware de la tabla de páginas puede hacerse de varias maneras. En el caso más simple, la tabla de páginas se implementa como un conjunto de **registros** dedicados. Estos registros deben construirse con lógica de muy alta velocidad, para hacer que el proceso de traducción de direcciones basado en la paginación sea muy eficiente. Cada acceso a la memoria debe pasar a través del mapa de paginación, por lo que la eficiencia es una de las consideraciones principales de implementación. El despachador de la CPU recarga estos registros al igual que recarga los registros restantes. Las instrucciones para cargar o modificar los registros de la tabla de páginas son, por supuesto, privilegiadas, de modo que sólo el sistema operativo puede cambiar el mapa de memoria. El sistema operativo PDP-11 de DEC es un ejemplo de este tipo de arquitectura; la dirección está compuesta de 16 bits y el tamaño de página es de 8 KB. La tabla de páginas contiene, por tanto ocho entradas que se almacenan en registros de alta velocidad.

El uso de registros para la tabla de páginas resulta satisfactorio si esta tabla es razonablemente pequeña (por ejemplo, 256 entradas). Sin embargo, la mayoría de las computadoras contemporáneas permiten que la tabla de páginas sea muy grande (por ejemplo, un millón de entradas);

para estas máquinas, el uso de registros de alta velocidad para incrementar la tabla de páginas no resulta factible. En lugar de ello, la tabla de páginas se mantiene en memoria principal, utilizándose un **registro base de la tabla de páginas** (PTBR, page-table base register) para apuntar a la tabla de páginas. Para cambiar las tablas de páginas, sólo hace falta cambiar este único registro, reduciéndose así sustancialmente el tiempo de cambio de contexto.

El problema con esta técnica es el tiempo requerido para acceder a una ubicación de memoria del usuario. Si queremos acceder a la ubicación *i*, primero tenemos que realizar una indexación en la tabla de páginas, utilizando el valor del registro PTBR, desplazado según el número de página, para obtener el número de marco. Esta tarea requiere un acceso a memoria y nos proporciona el número de marco, que se combina con el desplazamiento de página para generar la dirección real. Sólo entonces podremos acceder a la ubicación deseada de memoria. Con este esquema, hacen falta *dos* accesos de memoria para acceder a un byte (uno para obtener la entrada de la tabla de páginas y otro para obtener el byte). Por tanto, el acceso a memoria se ralentiza dividiéndose a la mitad. En la mayor parte de las circunstancias, este retardo es intolerable; para eso, es mejor utilizar un mecanismo de intercambio de memoria.

La solución estándar a este problema consiste en utilizar una caché hardware especial de pequeño tamaño y con capacidad de acceso rápido, denominada **búfer de consulta de traducción** (TLB, translation look-aside buffer). El búfer TLB es una memoria asociativa de alta velocidad. Cada entrada del búfer TLB está compuesta de dos partes: una clave (o etiqueta) y un valor. Cuando se le presenta un elemento a la memoria asociativa, ese elemento se compara simultáneamente con todas las claves. Si se encuentra el elemento, se devuelve el correspondiente campo de valor. Esta búsqueda se realiza en paralelo de forma muy rápida, pero el hardware necesario es caro. Normalmente, el número de entradas del búfer TLB es pequeño; suele estar comprendido entre 64 y 1024.

El búfer TLB se utiliza con las tablas de página de la forma siguiente: el búfer TLB contiene sólo unas cuantas entradas de la tabla de páginas; cuando la CPU genera una dirección lógica, se presenta el número de página al TLB y si se encuentra ese número de página, su número de marco correspondiente estará inmediatamente disponible y se utilizará para acceder a la memoria. Toda esta operación puede llegar a ser tan sólo un 10 por ciento más lenta que si se utilizara una referencia a memoria no convertida.

Si el número de página no se encuentra en el búfer TLB (lo que se conoce con el nombre de **fallo de** TLB), es necesario realizar una referencia a memoria para consultar la tabla de páginas. Una vez obtenido el número de marco, podemos emplearlo para acceder a la memoria (Figura 8.11). Además, podemos añadir el número de página y el número de marco al TLB, para poder encontrar los correspondientes valores rápidamente en la siguiente referencia que se realice. Si el TLB ya está lleno, el sistema operativo deberá seleccionar una de las entradas para sustituirla. Las políticas de sustitución utilizadas van desde una política aleatoria hasta algoritmos que lo que hacen es sustituir la entrada menos recientemente utilizada (LRU, least recently used). Además, algunos búferes TLB permiten **cablear** las entradas, lo que significa que esas entradas no pueden eliminarse del TLB. Normalmente, las entradas del TLB correspondientes al código del *kernel* están cableadas.

Algunos búferes TLB almacenan **identificadores del espacio de direcciones** (ASID, address-space identifier) en cada entrada TLB. Cada identificador ASID identifica unívocamente cada proceso y se utiliza para proporcionar mecanismos de protección del espacio de direcciones correspondiente a ese proceso. Cuando el TLB trata de resolver los números de página virtuales, verifica que el identificador ASID del proceso que actualmente se esté ejecutando se corresponda con el identificador ASID asociado con la página virtual. Si ambos identificadores no coinciden, el intento se trata como un fallo de TLB. Además de proporcionar protección del espacio de direcciones, los identificadores ASID permiten al TLB contener entradas simultáneamente para varios procesos distintos. Si el TLB no soporta la utilización de identificadores ASID distintos, cada vez que se seleccione una nueva tabla de páginas (por ejemplo, en cada cambio de contexto), será necesario **vaciar** (o borrar) el TLB para garantizar que el siguiente proceso que se ejecute no utilice una información incorrecta de traducción de direcciones. Si no se hiciera así, el TLB podría incluir entradas antiguas que contuvieran direcciones virtuales válidas pero que tuvieran asociadas direcciones físicas incorrectas o no válidas que se correspondan con el proceso anterior.

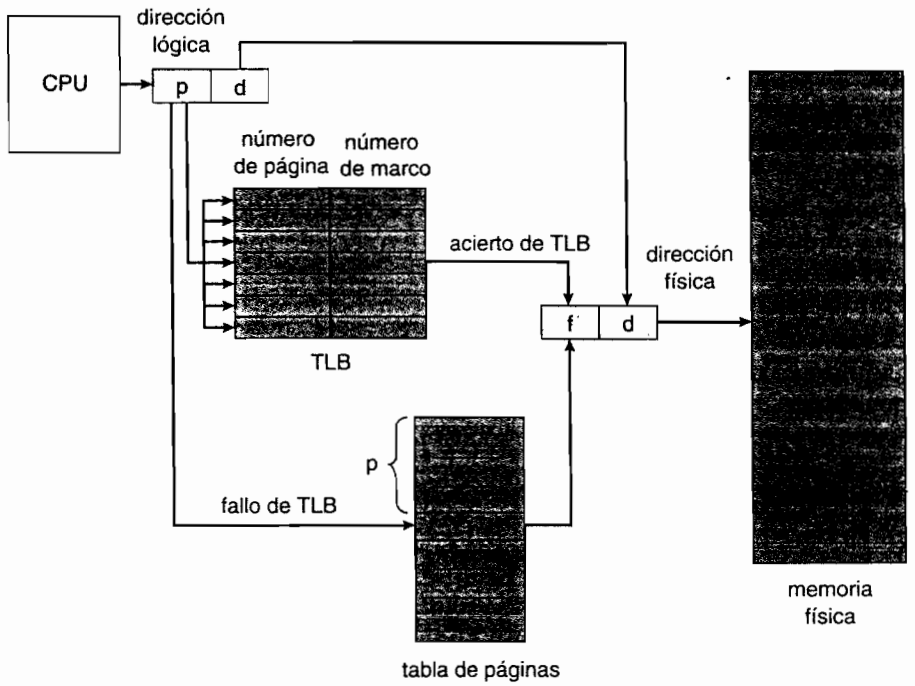


Figura 8.11 Hardware de paginación con TLB.

El porcentaje de veces que se encuentra un número de página concreto en el TLB se denomina **tasa de acierto**. Una tasa de acierto del 80 por ciento significa que hemos encontrado el número de página deseado en el TLB un 80 por ciento de las veces. Si se tarda 20 nanosegundos en consultar el TLB y 100 nanosegundos en acceder a la memoria, entonces un acceso a memoria convertida (mapeada) tardará 120 nanosegundos cuando el número de página se encuentre en el TLB. Si no conseguimos encontrar el número de página en el TLB (20 nanosegundos), entonces será necesario acceder primero a la memoria correspondiente a la tabla de páginas para extraer el número de marco (100 nanosegundos) y luego acceder al byte deseado en la memoria (100 nanosegundos), lo que nos da un total de 220 nanosegundos. Para calcular el **tiempo efectivo de acceso a memoria**, ponderamos cada uno de los casos según su probabilidad:

$$\begin{aligned}\text{tiempo efectivo de acceso} &= 0{,}80 \times 120 + 0{,}20 \times 220 \\ &= 140 \text{ nanosegundos.}\end{aligned}$$

En este ejemplo, vemos que se experimenta un aumento del 40 por ciento en el tiempo de acceso a memoria (de 100 a 140 nanosegundos).

Para una tasa de acierto del 98 por ciento, tendremos

$$\begin{aligned}\text{tiempo efectivo de acceso} &= 0{,}98 \times 120 + 0{,}02 \times 220 \\ &= 122 \text{ nanosegundos.}\end{aligned}$$

Esta tasa de acierto mejorada genera sólo un 22 por ciento de aumento en el tiempo de acceso. Exploraremos con más detalle el impacto de la tasa de acierto sobre el TLB en el Capítulo 9.

8.4.3 Protección

La protección de memoria en un entorno paginado se consigue mediante una serie de bits de protección asociados con cada marco. Normalmente, estos bits se mantienen en la tabla de páginas.

Uno de los bits puede definir una página como de lectura-escritura o de sólo lectura. Toda referencia a memoria pasa a través de la tabla de páginas con el fin de encontrar el número de marco correcto. Al mismo tiempo que se calcula la dirección física, pueden comprobarse los bits de pro-

tección para verificar que no se esté haciendo ninguna escritura en una página de sólo lectura. Todo intento de escribir una página de sólo lectura provocará una interrupción hardware al sistema operativo (o una violación de protección de memoria).

Podemos ampliar fácilmente este enfoque con el fin de proporcionar un nivel más fino de protección. Podemos diseñar un hardware que proporcione protección de sólo lectura, de lectura-escritura o de sólo ejecución, o podemos permitir cualquier combinación de estos accesos, proporcionando bits de protección separados para cada tipo de acceso. Los intentos ilegales provocarán una interrupción hardware hacia el sistema operativo.

Generalmente, se suele asociar un bit adicional con cada entrada de la tabla de páginas: un bit **válido-inválido**. Cuando se configura este bit como "válido", la página asociada se encontrará en el espacio de direcciones lógicas del proceso y será, por tanto, una página legal (o válida). Cuando se configura el bit como "inválido", la página no se encuentra en el espacio de direcciones lógicas del proceso. Las direcciones ilegales se capturan utilizando el bit válido-inválido. El sistema operativo configura este bit para cada página, con el fin de permitir o prohibir el acceso a dicha página.

Suponga, por ejemplo, que en un sistema con un espacio de direcciones de 14 bits (0 a 16383), tenemos un programa que sólo debe utilizar las direcciones comprendidas entre 0 y 10468. Dado un tamaño de página de 2 KB, tendremos la situación mostrada en la Figura 8.12. Las direcciones de las páginas 0, 1, 2, 3, 4 y 5 se convierten normalmente mediante la tabla de páginas, pero cualquier intento de generar una dirección en las páginas 6 o 7, sin embargo, se encontrará con el que bit válido-inválido está configurado como inválido, por lo que la computadora generará una interrupción hacia el sistema operativo (referencia de página inválida).

Observe que este esquema genera un problema: puesto que el programa sólo se extiende hasta la dirección 10468, todas las referencias que se hagan más allá de dicha dirección serán ilegales. Sin embargo, las referencias a la página 5 están clasificadas como válidas, por lo que los accesos a las direcciones que van hasta la posición 12287 son válidas. Sólo las direcciones comprendidas entre 12288 y 16383 están marcadas como inválidas. Este problema surge como resultado del tamaño de página de 2 KB y refleja el problema de la fragmentación interna que sufren los mecanismos de paginación.

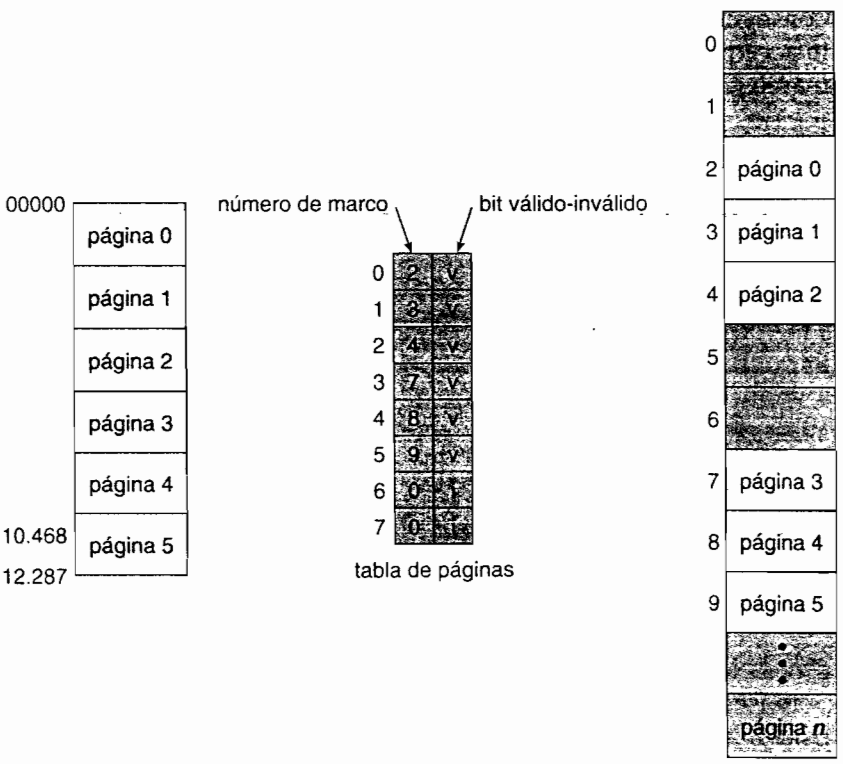


Figura 8.12 Bit válido (v)-inválido (i) en una tabla de páginas.

Los procesos raramente utilizan todo su rango de direcciones. De hecho, muchos procesos sólo emplean una pequeña fracción del espacio de direcciones que tiene disponible. Sería un desperdicio, en estos casos, crear una tabla de páginas con entradas para todas las páginas del rango de direcciones. La mayor parte de esta tabla permanecería sin utilizar, pero ocupando un valioso espacio de memoria. Algunos sistemas proporcionan mecanismos hardware especiales, en la forma de un **registro de longitud de la tabla de páginas** (PTLR, page-table length register), para indicar el tamaño de la tabla de páginas. Este valor se compara con cada dirección lógica para verificar que esa dirección se encuentre dentro del rango de direcciones válidas definidas para el proceso. Si esta comprobación fallara, se produciría una interrupción de error dirigida al sistema operativo.

8.4.4 Páginas compartidas

Una ventaja de la paginación es la posibilidad de *compartir* código común. Esta consideración es particularmente importante en un entorno de tiempo compartido. Considere un sistema que de soporte a 40 usuarios, cada uno de los cuales esté ejecutando un editor de texto. Si el editor de texto está compuesto por 150 KB de código y 50 KB de espacio de datos, necesitaremos 8000 KB para dar soporte a los 40 usuarios. Sin embargo, si se trata de **código reentrante** (o **código puro**), podrá ser compartido, como se muestra en la Figura 8.13. En ella podemos ver un editor de tres páginas (cada página tiene 50 KB de tamaño, utilizándose un tamaño de página tan grande para simplificar la figura) compartido entre los tres procesos. Cada proceso tiene su propia página de datos.

El código reentrante es código que no se auto-modifica; nunca cambia durante la ejecución. De esta forma, dos o más procesos pueden ejecutar el mismo código al mismo tiempo. Cada proceso tendrá su propia copia de los registros y del almacenamiento de datos, para albergar los datos requeridos para la ejecución del proceso. Por supuesto, los datos correspondientes a dos procesos distintos serán diferentes.

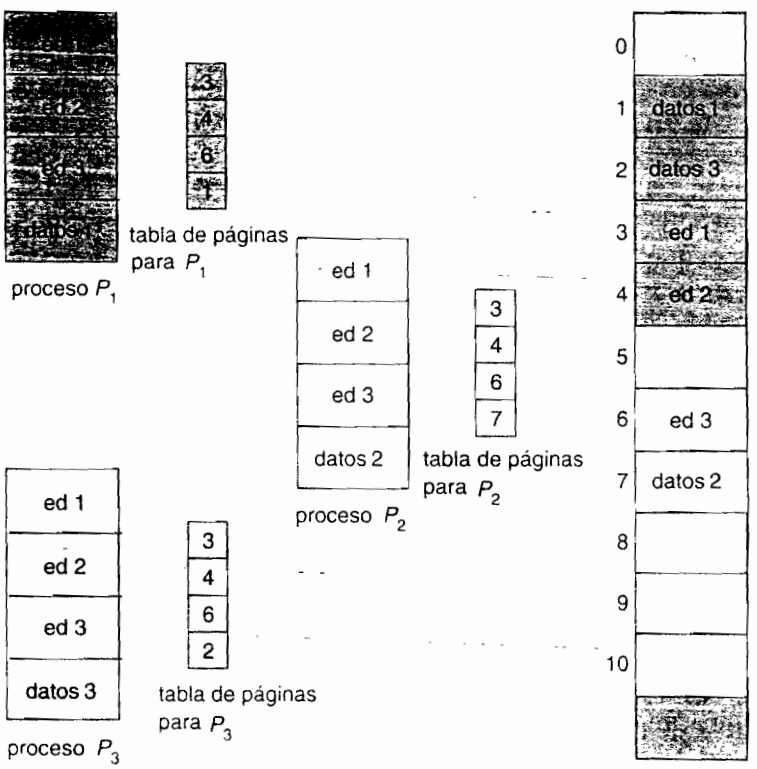


Figura 8.13 Compartición de código en un entorno paginado.

Sólo es necesario mantener en la memoria física una copia del editor. La tabla de páginas de cada usuario se corresponderá con la misma copia física del editor, pero las páginas de datos se harán corresponder con marcos diferentes. Así, para dar soporte a 40 usuarios, sólo necesitaremos una copia del editor (150 KB), más 40 copias del espacio de datos de 50 KB de cada usuario. El espacio total requerido será ahora de 2.150 KB, en lugar de 8.000 KB, lo que representa un ahorro considerable.

También pueden compartirse otros programas que se utilicen a menudo, como por ejemplo compiladores, sistemas de ventanas, bibliotecas de tiempo de ejecución, sistemas de base de datos, etc. Para poder compartirlo, el código debe ser reentrante. El carácter de sólo-lectura del código compartido no debe dejarse a expensas de la corrección de ese código, sino que el propio sistema operativo debe imponer esta propiedad.

La compartición de memoria entre procesos dentro de un sistema es similar a la compartición del espacio de direcciones de una tarea por parte de las hebras de ejecución, descrita en el Capítulo 4. Además, recuerde que en el Capítulo 3 hemos descrito la memoria compartida como un método de comunicación interprocesos. Algunos sistemas operativos implementan la memoria compartida utilizando páginas compartidas.

La organización de la memoria en páginas proporciona numerosos beneficios, además de permitir que varios procesos compartan las mismas páginas físicas. Hablaremos de algunos de los otros beneficios derivados de este esquema en el Capítulo 9.

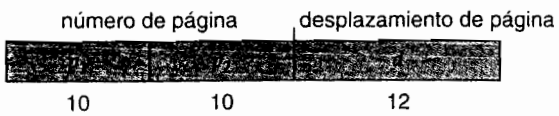
8.5 Estructura de la tabla de páginas

En esta sección, vamos a explorar algunas de las técnicas más comunes para estructurar la tabla de páginas.

8.5.1 Paginación jerárquica

La mayoría de los sistemas informáticos modernos soportan un gran espacio de direcciones lógicas (2^{32} a 2^{64}). En este tipo de entorno, la propia tabla de páginas llega a ser excesivamente grande. Por ejemplo, considere un sistema con un espacio lógico de direcciones de 32 bits. Si el tamaño de página en dicho sistema es de 4 KB (2^{12}), entonces la tabla de páginas puede estar compuesta de hasta un millón de entradas ($2^{32}/2^{12}$). Suponiendo que cada entrada esté compuesta por 4 bytes, cada proceso puede necesitar hasta 4 MB de espacio físico de direcciones sólo para la tabla de páginas. Obviamente, no conviene asignar la tabla de páginas de forma contigua en memoria principal. Una solución simple a este problema consiste en dividir la tabla de páginas en fragmentos más pequeños y podemos llevar a cabo esta división de varias formas distintas.

Una de esas formas consiste en utilizar un algoritmo de paginación de dos niveles, en el que la propia tabla de páginas está también paginada (Figura 8.14). Recuerde nuestro ejemplo de una máquina de 32 bits con un tamaño de página de 4 KB; una dirección lógica estará dividida en un número de página de 20 bits y un desplazamiento de página de 12 bits. Puesto que la tabla de páginas está paginada, el número de páginas se divide a su vez en un número de página de 10 bits y un desplazamiento de página de10 bits. De este modo, una dirección lógica tendría la estructura siguiente:



donde p_1 es un índice a la tabla de páginas externa y p_2 es el desplazamiento dentro de la página de la tabla de páginas externa. El método de traducción de direcciones para esta arquitectura se muestra en la Figura 8.15. Puesto que la traducción de las direcciones se realiza comenzando por la tabla de páginas externa y continuando luego por la interna, este esquema también se conoce con el nombre de **tabla de páginas con correspondencia (mapeo) directa**.

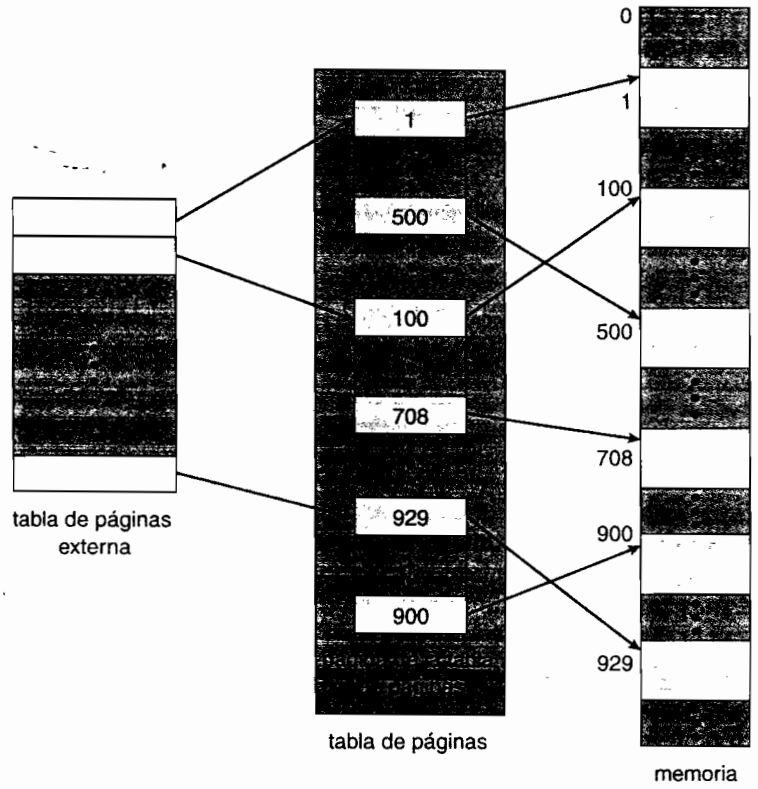


Figura 8.14 Un esquema de tabla de páginas en dos niveles.

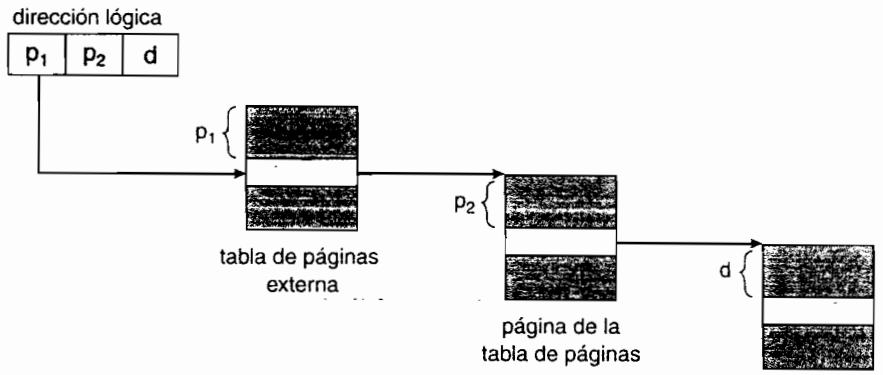
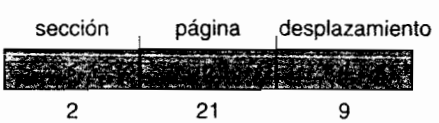


Figura 8.15 Traducción de una dirección para una arquitectura de paginación en dos niveles de 32 bits.

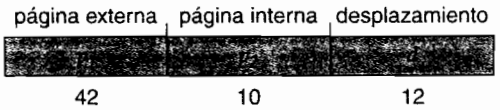
La arquitectura VAX también soporta una variante del mecanismo de paginación en dos niveles. El VAX es una máquina de 32 bits con un tamaño de página de 512 bytes. El espacio lógico de direcciones de un proceso se divide en cuatro secciones iguales, cada una de las cuales está compuesta de 2^{30} bytes. Cada sección representa una parte distinta del espacio lógico de direcciones de un proceso. Los primeros 2 bits de mayor peso de la dirección lógica designan la sección apropiada, los siguientes 21 bits representan el número lógico de página de dicha sección y los 9 bits finales representan un desplazamiento dentro de la página deseada.

Particionando la tabla de páginas de esta manera, el sistema operativo puede dejar particiones sin utilizar hasta que un proceso las necesite. Una dirección en la arquitectura VAX tiene la estructura siguiente:



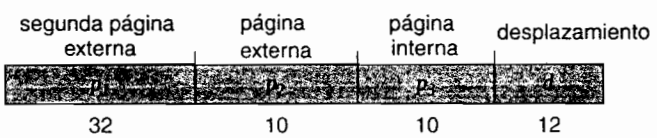
donde s designa el número de la sección, p es un índice a la tabla de páginas y d es el desplazamiento dentro de la página. Aunque se utiliza este esquema, el tamaño de una tabla de página de un sólo nivel para un proceso VAX que utilice una sección es de 2^{21} bits * 4 bytes por entrada, 8 MB. Para reducir aún más la utilización de memoria principal, la arquitectura VAX pagina las tablas de páginas de los procesos de usuario.

Para un sistema con un espacio lógico de direcciones de 4 bits, un esquema de paginación de dos niveles ya no resulta apropiado. Para ilustrar este punto, supongamos que el tamaño de página en dicho sistema fuera de 4 KB (2^{12}). En este caso, la tabla de páginas estaría compuesta de hasta 2^{52} entradas. Si utilizáramos un esquema de paginación en dos niveles, las tablas de páginas internas podrían tener (como solución más fácil) una página de longitud, es decir, contener 1K entradas de 4 bytes. Las direcciones tendrían la siguiente estructura:



La tabla de páginas externas estará compuesta de 2^{42} entradas, es decir, 2^{44} bytes. La forma obvia para evitar tener una tabla tan grande consiste en dividir la tabla de páginas externa en fragmentos más pequeños. Esta técnica se utiliza también en algunos procesadores de 32 bits, para aumentar la flexibilidad y la eficiencia.

Podemos dividir la tabla de páginas externas de varias formas. Podemos paginar la tabla de páginas externa, obteniendo así un esquema de paginación en tres niveles. Suponga que la tabla de páginas externa está formada por páginas de tamaño estándar (2^{10} entradas o 2^{12} bytes); el espacio de direcciones de 64 bits seguirá siendo inmanejable:



La tabla de páginas externas seguirá teniendo 2^{34} bytes de tamaño.

El siguiente paso sería un esquema de paginación en cuatro niveles, en el que se paginaría también la propia tabla de páginas externas de segundo nivel. La arquitectura SPARC (con direccionamiento de 32 bits) permite el uso de un esquema de paginación en tres niveles, mientras que la arquitectura 68030 de Motorola, también de 32 bits, soporta un esquema de paginación en cuatro niveles.

Para las arquitecturas de 64 bits, las tablas de páginas jerárquicas se consideran, por lo general, inapropiadas. Por ejemplo, la arquitectura UltraSPARC de 64 bits requeriría siete niveles de paginación (un número prohibitivo de accesos a memoria) para traducir cada dirección lógica.

8.5.2 Tablas de páginas hash

Una técnica común para gestionar los espacios de direcciones superiores a 32 bits consiste en utilizar una **tabla *hash* de páginas**, donde el valor *hash* es el número de página virtual. Cada entrada de la tabla *hash* contiene una lista enlazada de elementos que tienen como valor *hash* una misma ubicación (con el fin de tratar las colisiones). Cada elemento está compuesto de tres campos: (1) el número de página virtual, (2) el valor del marco de página mapeado y (3) un punto del siguiente elemento de la lista enlazada.

El algoritmo funciona de la forma siguiente: al número de página virtual de la dirección virtual se le aplica una función *hash* para obtener un valor que se utiliza como índice para la tabla *hash*. El número de página virtual se compara con el campo 1 del primer elemento de la lista enlazada. Si hay una correspondencia, se utiliza el marco de página correspondiente (campo 2) para formar la dirección física deseada. Si no se produce una correspondencia, se exploran las subsiguientes entradas de la lista enlazada hasta encontrar el correspondiente número de página virtual. Este esquema se muestra en la Figura 8.16.

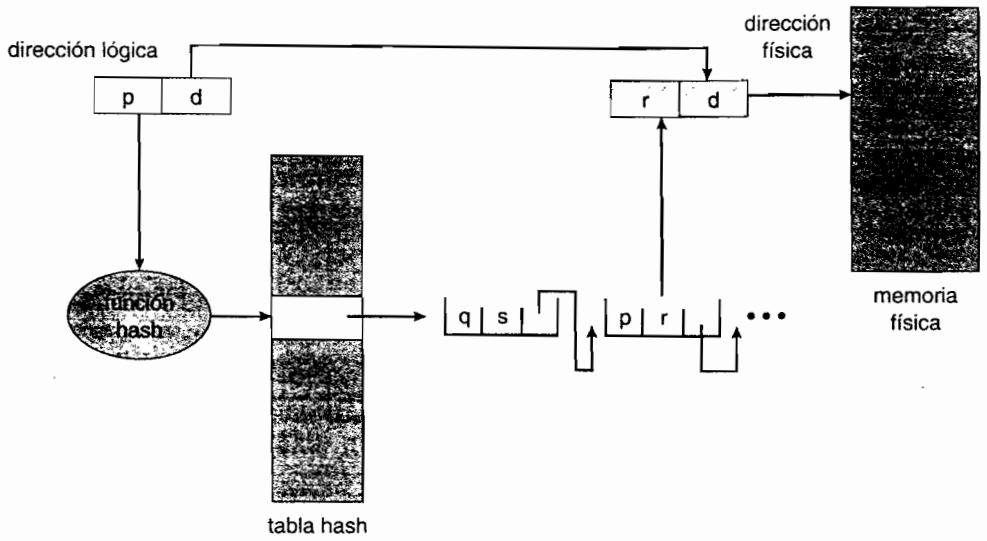


Figura 8.16 Tabla de páginas hash.

También se ha propuesto una variante de este esquema que resulta más adecuada para los espacios de direcciones de 64 bits. Esta variante utiliza **tablas de páginas en clúster**, que son similares a las tablas de páginas *hash*, salvo porque cada entrada de la tabla *hash* apunta a varias páginas (como por ejemplo 16) en lugar de a una sola página. De este modo, una única entrada de la tabla de páginas puede almacenar las correspondencias (mapeos) para múltiples marcos de página física. Las tablas de páginas en *cluster* son particularmente útiles para espacios de direcciones **dispersos** en los que las referencias a memoria son no continuas y están dispersas por todo el espacio de direcciones.

8.5.3 Tablas de páginas invertidas

Usualmente, cada proceso tiene una tabla de páginas asociada. La tabla de páginas incluye una entrada por cada página que el proceso esté utilizando (o un elemento por cada dirección virtual, independientemente de la validez de ésta). Esta representación de la tabla resulta natural, ya que los procesos hacen referencia a las páginas mediante las direcciones virtuales de éstas. El sistema operativo debe entonces traducir cada referencia a una dirección física de memoria. Puesto que la tabla está ordenada según la dirección virtual, el sistema operativo podrá calcular dónde se encuentra en la tabla la entrada de dirección física asociada y podrá utilizar dicho valor directamente. Pero una de las desventajas de este método es que cada tabla de página puede estar compuesta por millones de entradas. Estas tablas pueden ocupar una gran cantidad de memoria física, simplemente para controlar el modo en que se están utilizando otras partes de la memoria física.

Para resolver este problema, podemos utilizar una **tabla de páginas invertida**. Las tablas de páginas invertidas tienen una entrada por cada página real (o marco) de la memoria. Cada entrada está compuesta de la dirección virtual de la página almacenada en dicha ubicación de memoria real, e incluye información acerca del proceso que posee dicha página. Por tanto, en el sistema sólo habrá una única tabla de páginas y esa tabla sólo tendrá una entrada por cada página de memoria física. La Figura 8.17 muestra la operación de una tabla de páginas invertida. Compárela con la Figura 8.7, donde se muestra el funcionamiento de una tabla de páginas estándar. Las tablas de páginas invertidas requieren a menudo que se almacene un identificador del espacio de direcciones (Sección 8.4.2) en cada entrada de la tabla de páginas, ya que la tabla contiene usualmente varios espacios de direcciones distintos que se hacen corresponder con la memoria física. Almacenar el identificador del espacio de direcciones garantiza que cada página lógica de un proceso concreto se relacione con el correspondiente marco de página física. Como ejemplos de sistemas que utilizan las tablas de páginas invertidas podemos citar el PowerPC y la arquitectura UltraSPARC de 64 bits.

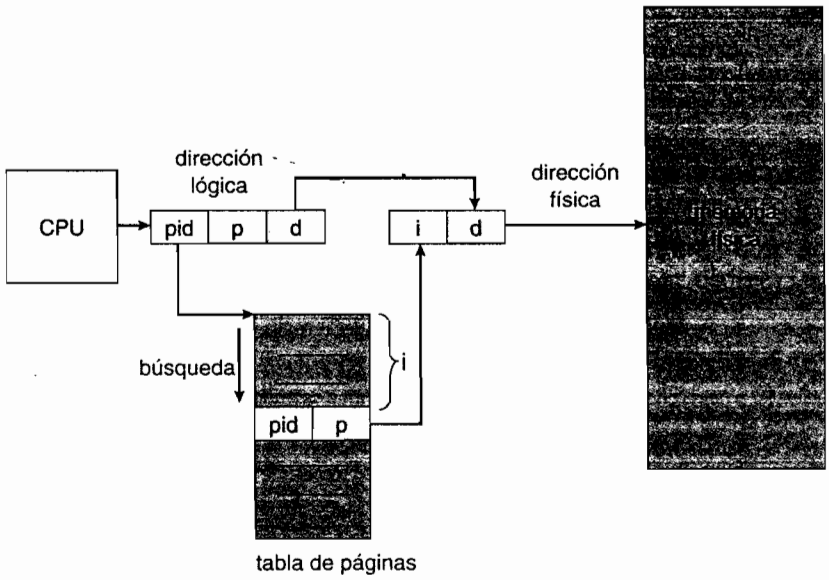


Figura 8.17 Tabla de páginas invertida.

Para ilustrar este método, vamos a describir una versión simplificada de la tabla de páginas invertida utilizada en IBM RT. Cada dirección virtual del sistema está compuesta por una tripleta

<id-proceso, número-página, desplazamiento>.

Cada entrada de la tabla de páginas invertida es una pareja <id-proceso, número-página> donde el identificador id-proceso asume el papel de identificador del espacio de direcciones. Cuando se produce una referencia a memoria, se presenta al subsistema de memoria una parte de la dirección virtual, compuesta por <id-proceso, número-página>. Entonces, se explora la tabla de páginas invertida en busca de una correspondencia; si se encuentra esa correspondencia (por ejemplo, en la entrada *i*), se generará la dirección física <*i*, desplazamiento>. Si no se encuentra ninguna correspondencia, querrá decir que se ha realizado un intento de acceso ilegal a una dirección.

Aunque este esquema permite reducir la cantidad de memoria necesaria para almacenar cada tabla de páginas, incrementa la cantidad de tiempo necesaria para explorar la tabla cuando se produce una referencia a una página. Puesto que la tabla de páginas invertida está ordenada según la dirección física, pero las búsquedas se producen según las direcciones virtuales, puede que sea necesario explorar toda la tabla hasta encontrar una correspondencia y esta exploración consumiría demasiado tiempo. Para aliviar este problema, se utiliza una tabla *hash*, como la descrita en la Sección 8.5.2, con el fin de limitar la búsqueda a una sola entrada de la tabla de páginas o (como mucho) a unas pocas entradas. Por supuesto, cada acceso a la tabla *hash* añade al procedimiento una referencia a memoria, por lo que una referencia a memoria virtual requiere al menos dos lecturas de memoria real: una para la entrada de la tabla *hash* y otra para la tabla de páginas. Para mejorar la velocidad, recuerde que primero se explora el búfer TLB, antes de consultar la tabla *hash*.

Los sistemas que utilizan tablas de páginas invertidas tienen dificultades para implementar el concepto de memoria compartida. La memoria compartida se implementa usualmente en forma de múltiples direcciones virtuales (una para cada proceso que comparte la memoria), todas las cuales se hacen corresponder con una misma dirección física. Este método estándar no puede utilizarse con las tablas de páginas invertidas, porque sólo hay una única entrada de página virtual para cada página física y, como consecuencia, una misma página física no puede tener dos (o más) direcciones virtuales compartidas. Una técnica simple para resolver este problema consiste en permitir que la tabla de páginas sólo contenga una única correspondencia de una dirección virtual con la dirección física compartida. Esto significa que las referencias a direcciones virtuales que no estén asociadas darán como resultado un fallo de página.