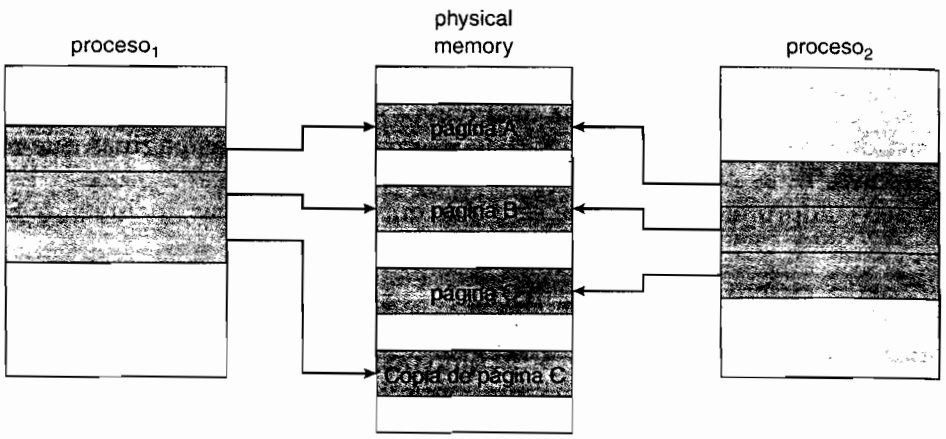

Figura 9.8 Después de que el proceso 1 modifique la página C.

Cuando se determina que se va a duplicar una página utilizando el mecanismo de copia durante la escritura, es importante fijarse en la ubicación desde la que se asignará la página libre. Muchos sistemas operativos proporcionan un **conjunto compartido** de páginas libres para satisfacer tales solicitudes. Esas páginas libres se suelen asignar cuando la pila o el cúmulo de un proceso deben expandirse o cuando es necesario gestionar páginas de copia durante la escritura. Los sistemas operativos asignan normalmente esas páginas utilizando una técnica denominada **relleno de ceros bajo demanda**. Las páginas de relleno de cero bajo demanda se llenan de ceros antes de asignarlas, eliminando así el contenido anterior.

Diversas versiones de UNIX (incluyendo Solaris y Linux) proporcionan también una variante de la llamada al sistema `fork()`: se trata de la llamada `vfork()` (que quiere decir **fork para memoria virtual**). `vfork()` opera de forma diferente de `fork()` en las operaciones de copia durante la escritura. Con `vfork()`, el proceso padre se suspende y el proceso hijo utiliza el espacio de direcciones del padre. Puesto que `vfork()` no utiliza el mecanismo de copia durante la escritura, si el proceso hijo modifica cualquiera de las páginas del espacio de direcciones del padre, las páginas modificadas serán visibles para el padre una vez que éste reanude su ejecución. Por tanto, `vfork()` debe utilizarse con precaución, para garantizar que el proceso hijo no modifique el espacio de direcciones del padre. `vfork()` está pensado para usarse cuando el proceso hijo invoca `exec()` inmediatamente después de la creación. Puesto que no se produce ninguna copia de páginas, `vfork()` es un método extremadamente eficiente para la creación de procesos y se utiliza en ocasiones para implementar interfaces *shell* de línea de comandos UNIX.

9.4 Sustitución de páginas

En nuestra explicación anterior acerca de la tasa de fallos de página, hemos supuesto que para cada página sólo se produce como mucho un fallo, la primera vez que se hace referencia a la misma. Sin embargo, esta suposición no es del todo precisa. Si un proceso de diez páginas sólo utiliza en realidad la mitad de ellas, el mecanismo de paginación bajo demanda nos ahorrará las operaciones de E/S necesarias para cargar las cinco páginas que nunca se utilizan. De este modo, podemos incrementar el grado de multiprogramación ejecutando el doble de procesos. Así, si tuviéramos cuarenta marcos, podríamos ejecutar ocho procesos en lugar de los cuatro que podrían ejecutarse si cada uno de los procesos requiriera diez marcos (cinco de los cuales jamás se utilizarían).

Si incrementamos nuestro grado de multiprogramación, estaremos **sobreasignando** la memoria. Si ejecutamos seis procesos, cada uno de los cuales tiene diez páginas de tamaño pero utiliza en realidad únicamente cinco páginas, tendremos una tasa de utilización de la CPU y una tasa de procesamiento más altas, quedándonos diez marcos libres. Es posible, sin embargo, que cada uno de estos procesos, para un determinado conjunto de datos, trate repentinamente de utilizar sus diez páginas, lo que implicará que hacen falta 60 marcos cuando sólo hay cuarenta disponibles

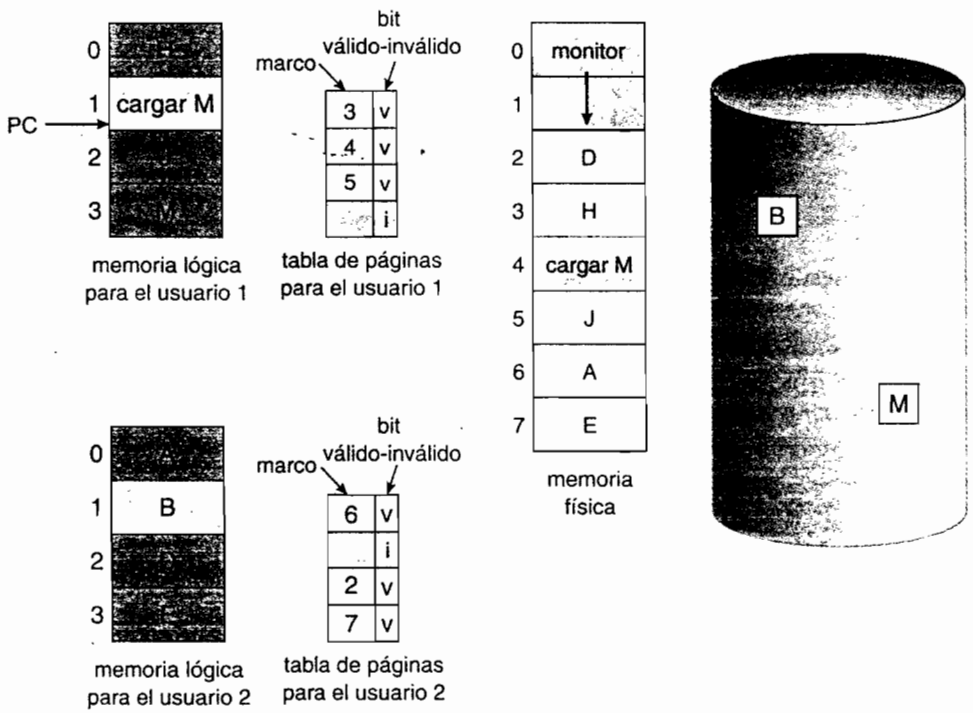

Figura 9.9 La necesidad de la sustitución de páginas.

Además, tenga en cuenta que la memoria del sistema no se utiliza sólo para almacenar páginas de programas. Los búferes de E/S también consumen una cantidad significativa de memoria. Este otro tipo de uso puede incrementar las demandas impuestas a los algoritmos de asignación de memoria. Decidir cuánta memoria asignar a la E/S y cuánta a las páginas de programa representa un considerable desafío. Algunos sistemas asignan un porcentaje fijo de memoria para los búferes de E/S, mientras que otros permiten que los procesos de usuario y el subsistema de E/S compitan por la memoria del sistema.

La sobreasignación de memoria se manifiesta de la forma siguiente. Imagine que, cuando se está ejecutando un proceso de usuario, se produce un fallo de página. El sistema operativo determina dónde reside la página deseada dentro del disco y entonces se encuentra con que no hay *ningún* marco libre en la lista de marcos libres; toda la memoria está siendo utilizada (Figura 9.9).

El sistema operativo dispone de varias posibilidades en este punto. Una de ellas sería terminar el proceso de usuario; sin embargo, la paginación bajo demanda es, precisamente, un intento del sistema operativo para mejorar la tasa de utilización y la tasa de procesamiento del sistema informático. Los usuarios no deben ser conscientes de que sus procesos se están ejecutando en un sistema paginado, es decir, la paginación debería ser lógicamente transparente para el usuario. Por tanto, esta opción no es la mejor de las posibles.

En lugar de ello, el sistema operativo puede descargar un proceso de memoria, liberando todos los marcos correspondientes y reduciendo el nivel de multiprogramación. Esta opción resulta adecuada en algunas circunstancias, y la analizaremos con más detalle en la Sección 9.6. Sin embargo, vamos a ver aquí la solución más comúnmente utilizada: la **sustitución de páginas**.

9.4.1 Sustitución básica de páginas

La sustitución de páginas usa la siguiente técnica. Si no hay ningún marco libre, localizamos uno que no esté siendo actualmente utilizado y lo liberamos. Podemos liberar un marco escribiendo su contenido en el espacio de intercambio y modificando la tabla de páginas (y todas las demás tablas) para indicar que esa página ya no se encuentra en memoria (Figura 9.10). Ahora podremos utilizar el marco liberado para almacenar la página que provocó el fallo de página en el proceso.

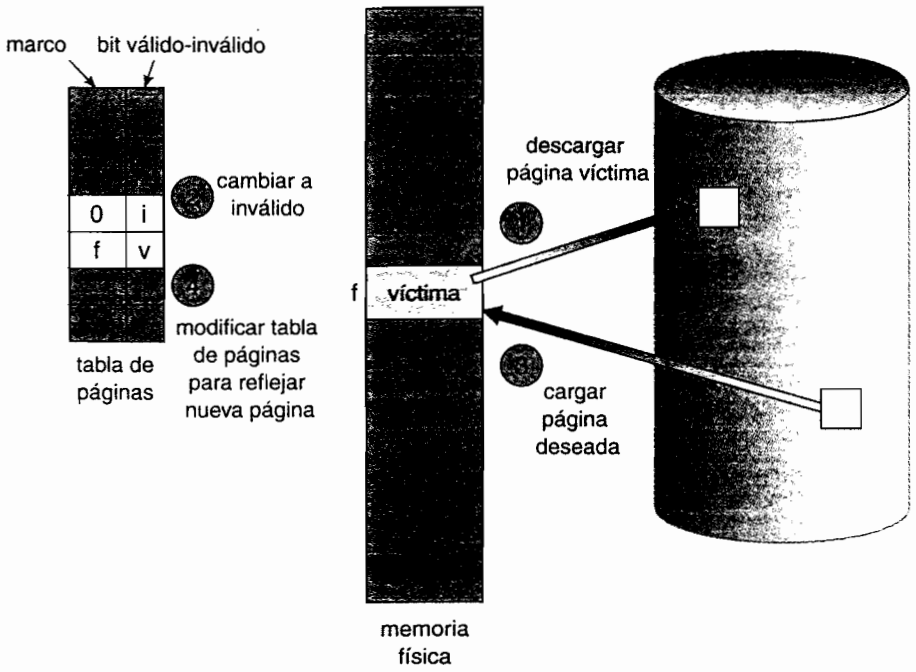

Figura 9.10 Sustitución de páginas.

Modifiquemos la rutina de servicio del fallo de página para incluir este mecanismo de sustitución de páginas:

1. Hallar la ubicación de la página deseada dentro del disco.
2. Localizar un marco libre:
   a. Si hay un marco libre, utilizarlo.
   b. Si no hay ningún marco libre, utilizar un algoritmo de sustitución de páginas para seleccionar un **marco víctima**
   c. Escribir el marco víctima en el disco; cambiar las tablas de páginas y de marcos correspondientemente.
3. Leer la página deseada y cargarla en el marco recién liberado; cambiar las tablas de páginas y de marcos.
4. Reiniciar el proceso de usuario

Observe que, si no hay ningún marco libre, se necesitan *dos* transferencias de páginas (una descarga y una carga). Esta situación duplica, en la práctica, el tiempo de servicio del fallo de página e incrementa correspondientemente el tiempo efectivo de acceso.

Podemos reducir esta carga de trabajo adicional utilizando un **bit de modificación** (o **bit sucio**). Cuando se utiliza este esquema, cada página o marco tiene asociado en el hardware un bit de modificación. El bit de modificación para una página es activado por el hardware cada vez que se escribe en una palabra o byte de la página, para indicar que la página ha sido modificada. Cuando seleccionemos una página para sustitución, examinamos su bit de modificación. Si el bit está activado, sabremos que la página ha sido modificada desde que se la leyó del disco. En este caso, deberemos escribir dicha página en el disco. Sin embargo, si el bit de modificación no está activado, la página *no* habrá sido modificada desde que fue leída y cargada en memoria. Por tanto, si la copia de la página en el disco no ha sido sobreescrita (por ejemplo, por alguna otra página), no necesitaremos escribir la página de memoria en el disco, puesto que ya se encuentra allí. Esta técnica también se aplica a las páginas de sólo lectura (por ejemplo, las páginas de código binario): dichas páginas no pueden ser modificadas, por lo que se las puede descartar cuando se desee.

Este esquema puede reducir significativamente el término requerido para dar servicio a un fallo de página, ya que reduce a la mitad el tiempo de E/S *si* la página no ha sido modificada.

El mecanismo de sustitución de páginas resulta fundamental para la paginación bajo demanda. Completa la separación entre la memoria lógica y la memoria física y, con este mecanismo, se puede proporcionar a los programadores una memoria virtual de gran tamaño a partir de una memoria física más pequeña. Sin el mecanismo de paginación bajo demanda, las direcciones de usuario se mapearían sobre direcciones físicas, por lo que los dos conjuntos de direcciones podrían ser distintos, pero todas las páginas de cada proceso deberían cargarse en la memoria física. Por el contrario, con la paginación bajo demanda, el tamaño del espacio lógico de direcciones ya no está restringido por la memoria física. Si tenemos un proceso de usuario de veinte páginas, podemos ejecutarlo en diez marcos de memoria simplemente utilizando la paginación bajo demanda y un algoritmo de sustitución para encontrar un marco libre cada vez que sea necesario. Si hay que sustituir una página que haya sido modificada, su contenido se copiará en el disco. Una referencia posterior a dicha página provocaría un fallo de página y, en ese momento, la página volvería a cargarse en memoria, quizás sustituyendo a alguna otra página en el proceso.

Debemos resolver dos problemas principales a la hora de implementar la paginación bajo demanda: hay que desarrollar un **algoritmo de asignación de marcos** y un **algoritmo de sustitución de páginas**. Si tenemos múltiples procesos en memoria, debemos decidir cuántos marcos asignar a cada proceso. Además, cuando se necesita una sustitución de páginas, debemos seleccionar los marcos que hay que sustituir. El diseño de los algoritmos apropiados para resolver estos problemas es una tarea de gran importancia, porque las operaciones del E/S de disco son muy costosas en términos de rendimiento. Cualquier pequeña mejora en los métodos de paginación bajo demanda proporciona un gran beneficio en términos de rendimiento del sistema.

Existen muchos algoritmos distintos de sustitución de páginas; probablemente, cada sistema operativo tiene su propio esquema de sustitución. ¿Cómo podemos seleccionar un algoritmo de sustitución concreto? En general, intentaremos seleccionar aquel que tenga la tasa de fallos de página más baja.

Podemos evaluar un algoritmo ejecutándolo con una cadena concreta de referencias de memoria y calculando el número de fallos de página. La cadena de referencias de memoria se denomina **cadena de referencia**. Podemos generar cadenas de referencia artificialmente (por ejemplo, utilizando un generador de números aleatorios) o podemos obtener una traza de un sistema determinado y registrar la dirección de cada referencia de memoria. Esta última opción produce un gran número de datos (del orden de un millón de direcciones por segundo). Para reducir el número de datos, podemos emplear dos hechos.

En primer lugar, para un cierto tamaño de página (y el tamaño de página generalmente está fijo por el hardware o por el sistema), tan sólo necesitamos considerar el número de página, más que la dirección completa. En segundo lugar, si tenemos una referencia a la página p, todas las referencias *inmediatamente* siguientes que se hagan a la página p nunca provocaran un fallo de página. La página p estará en la memoria después de la primera referencia, por lo que todas las referencias que la sigan de forma inmediata no provocarán ningún fallo de página.

Por ejemplo, si trazamos un proceso concreto, podríamos registrar la siguiente secuencia de direcciones:

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,
0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

Suponiendo 100 bytes por página, esta secuencia se reduciría a la siguiente cadena de referencia:

1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

Para determinar el número de fallos de página para una cadena de referencia concreta y un algoritmo de sustitución de páginas determinado, también necesitamos conocer el número de marcos de página disponibles. Obviamente, a medida que se incrementa el número de marcos disponibles, se reduce el número de fallos de página. Para la cadena de referencia considerada ante-

riormente, por ejemplo, si tuviéramos tres o más marcos, sólo se producirían tres fallos de página: uno para la primera referencia a cada página.

Por contraste, si sólo hubiera un marco disponible, tendríamos una sustitución para cada referencia, lo que daría como resultado once fallos de página. En general, podemos esperar obtener una curva como la que se muestra en la Figura 9.11. A medida que se incrementa el número de marcos, el número de fallos de página cae hasta un cierto nivel mínimo. Por supuesto, si se añade memoria física se incrementará el número de marcos.

A continuación vamos a ilustrar diversos algoritmos de sustitución de páginas. Para ello, usaremos la cadena de referencia

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

para una memoria con tres marcos.

9.4.2 Sustitución de páginas FIFO

El algoritmo más simple de sustitución de páginas es un algoritmo de tipo FIFO (first-in, first-out). El algoritmo de sustitución FIFO asocia con cada página el instante en que dicha página fue cargada en memoria. Cuando hace falta sustituir una página, se elige la página más antigua. Observe que no es estrictamente necesario anotar el instante en que se cargó cada página, ya que podríamos simplemente crear una cola FIFO para almacenar todas las páginas en memoria y sustituir la página situada al principio de la cola. Cuando se carga en memoria una página nueva, se la inserta al final de la cola.

Para nuestra cadena de referencia de ejemplo, los tres marcos estarán inicialmente vacíos. Las primeras tres referencias (7, 0, 1) provocan fallos de página y esas páginas se cargan en esos marcos vacíos. La siguiente referencia (2) sustituirá a la página 7, porque la página 7 fue la primera en cargarse. Puesto que 0 es la siguiente referencia y 0 ya está en memoria, no se producirá ningún fallo de página para esta referencia. La primera referencia a 3 da como resultado la sustitución de la página 0, ya que ahora es la primera de la cola. Debido a esta sustitución, la siguiente referencia a 0, provocará un fallo de página. Entonces, la página 1 será sustituida por la página 0. Este proceso continua como se muestra en la Figura 9.12. Cada vez que se produce un fallo de página, mostramos las páginas que se encuentran en los tres marcos disponibles. En total, hay 15 fallos de página.

El algoritmo de sustitución de páginas FIFO es fácil de entender y de programar. Sin embargo, su rendimiento no siempre es bueno. Por un lado, la página sustituida puede ser un modulo

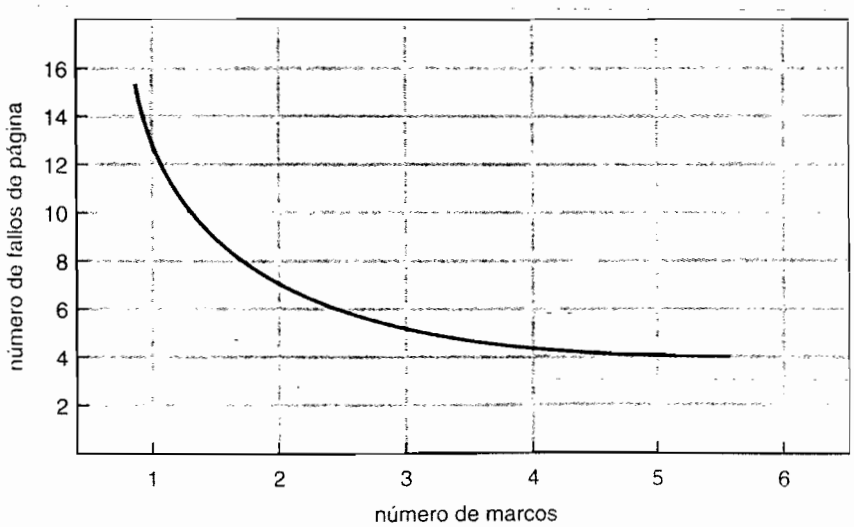

Figura 9.11 Gráfica de fallos de página en función del número de marcos.

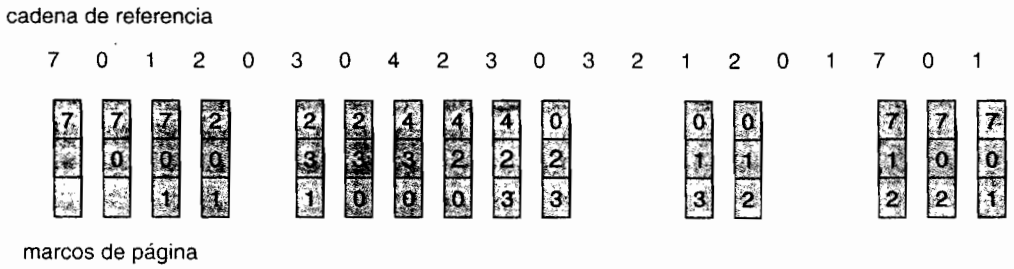

Figura 9.12 Algoritmos de sustitución de páginas FIFO.

de inicialización que hubiera sido empleado hace mucho tiempo y que ya no es necesario, pertambién podría contener una variable muy utilizada que fuera inicializada muy pronto y que se utilice de manera constante.

Observe que, aunque seleccionemos para sustitución una página que se esté utilizando activamente, todo sigue funcionando de forma correcta. Después de sustituir una página activa por otra nueva, se producirá casi inmediatamente un nuevo fallo de página que provocará la recarga de la página activa. Entonces, será necesario sustituir alguna otra página con el fin de volver a cargar en la memoria la página activa. De este modo, una mala elección de la página que hay que sustituir incrementa la tasa de fallos de página y ralentiza la ejecución de los procesos. Aunque, como vemos, eso no implica que la ejecución sea incorrecta.

Para ilustrar los problemas que pueden producirse con los algoritmos de sustitución de páginas FIFO, vamos a considerar la siguiente cadena de referencia:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

La Figura 9.13 muestra la curva de fallos de página para esta cadena de referencia, en función del número de marcos disponibles. Observe que el número de fallos para cuatro marcos (diez) es *superior* que el número de fallos para tres marcos (nueve). Este resultado completamente inesperado se conoce con el nombre de **anomalía de Belady**: para algunos algoritmos de sustitución de páginas, la tasa de fallos de página puede *incrementarse* a medida que se incrementa el número de marcos asignados. En principio, cabría esperar que al asignar más memoria a un proceso se mejorara su rendimiento, pero en algunos trabajos pioneros de investigación, los investigadores observaron que esta suposición no siempre es correcta. La anomalía de Belady se descubrió como consecuencia de estas investigaciones.

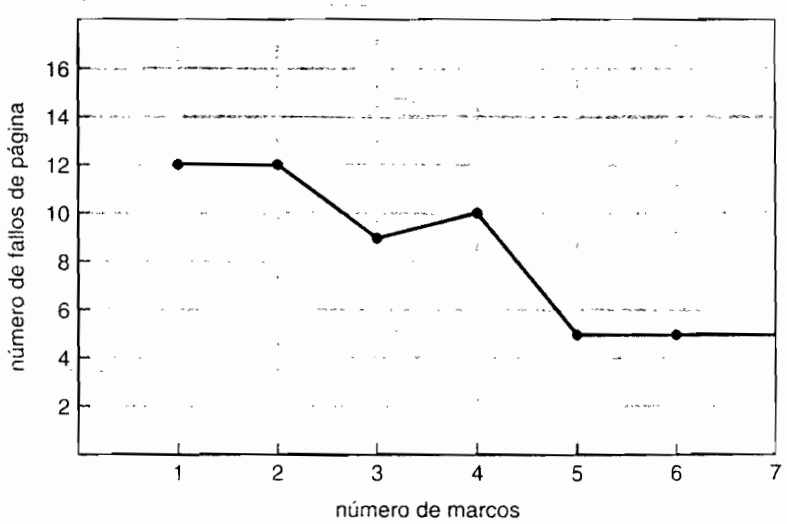

Figura 9.13 Curva de fallos de página para una sustitución FIFO con una determinada cadena de referencia.

9.4.3 Sustitución óptima de páginas

Uno de los resultados del descubrimiento de la anomalía de Belady fue la búsqueda de un a**ritmo óptimo de sustitución de páginas**. Un algoritmo óptimo de sustitución de páginas aquel que tenga la tasa más baja de fallos de página de entre todos los algoritmos y que nunca sujeto a la anomalía de Belady. Resulta que dicho algoritmo existe, habiéndoselo denominado o MIN. Consiste simplemente en lo siguiente:

Sustituir la página que no vaya a ser utilizada durante el período de tiempo más largo.

La utilización de este algoritmo de sustitución de página garantiza la tasa de fallos de pág más baja posible para un número fijo de marcos.

Por ejemplo, para nuestra cadena de referencia de ejemplo, el algoritmo óptimo de sustituc de páginas nos daría nueve fallos de página, como se muestra en la Figura 9.14. Las primeras t referencias provocan sendos fallos de página que rellenarán los tres marcos libres. La referenc a la página 2 sustituirá a la página 7, porque 7 no va a utilizarse hasta la referencia 18, mientr que la página 0 será utilizada en 5 y la página 1 en 14. La referencia a la página 3 sustituye a página 1, ya que la página 1 será la última de las tres páginas de memoria a la que se vuelva hacer referencia. Con sólo nueve fallos de página, el algoritmo óptimo de sustitución es much mejor que un algoritmo FIFO, que dará como resultado quince fallos de página (si ignoramos l primeros tres fallos de página, a los que todos los algoritmos están sujetos, entonces el algoritm óptimo de sustitución es el doble de bueno que un algoritmo de sustitución FIFO). De hecho, ni gún algoritmo de sustitución puede procesar esta cadena de referencia para tres marcos co menos de nueve fallos de página.

Desafortunadamente, el algoritmo óptimo de sustitución de páginas resulta difícil de impl mentar, porque requiere un conocimiento futuro de la cadena de referencia (nos hemos encontra do con una situación similar al hablar del algoritmo SJF de planificación de la CPU en la Secció 5.3.2). Como resultado, el algoritmo óptimo se utiliza principalmente con propósitos comparati vos. Por ejemplo, puede resultar útil saber que, aunque un nuevo algoritmo no sea óptimo, tien una desviación de menos del 12,3 por ciento con respecto al óptimo en caso peor y el 4,7 por cien to como promedio.

9.4.4 Sustitución de páginas LRU

Si el algoritmo óptimo no resulta factible, quizá sí sea posible una aproximación a ese algoritmo óptimo. La distinción clave entre los algoritmos FIFO y OPT (dejando a parte el hecho de que uno mira hacia atrás en el tiempo, mientras que el otro mira hacia adelante) es que el algoritmo FIFO utiliza el instante de tiempo en que se cargó una página en memoria, mientras que el algoritmo OPT utiliza el instante en el que hay que *utilizar* una página. Si utilizamos el pasado reciente como aproximación del futuro próximo, podemos entonces sustituir la página que *no haya sido utilizada* durante el período más largo de tiempo (Figura 9.15). Esta técnica se conoce como **algoritmo LRU** (least-recently-used, menos recientemente utilizada).

El algoritmo de sustitución LRU asocia con cada página el instante correspondiente al último uso de dicha página. Cuando hay que sustituir una página, el algoritmo LRU selecciona la página que no haya sido utilizada durante un período de tiempo más largo. Debemos considerar esta estrategia como el algoritmo óptimo de sustitución de páginas si miramos hacia atrás en el tiem-

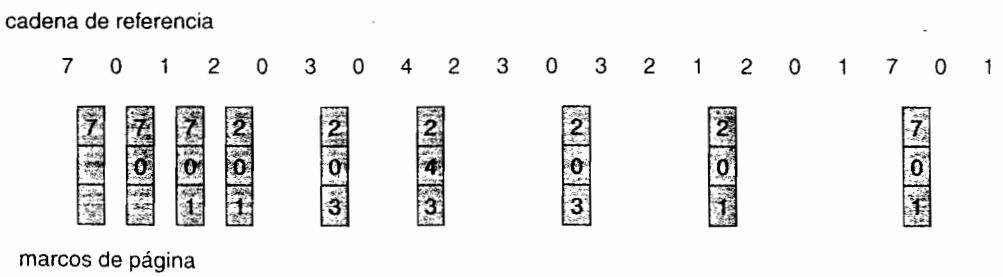

Figura 9.14 Algoritmo óptimo de sustitución de páginas.

po, en lugar de mirar hacia adelante (extrañamente, si denominamos S^R a la inversa de una cadena de referencia S, la tasa de fallos de página para el algoritmo OPT aplicado a S es igual que la tasa de fallos de página para el algoritmo OPT aplicado a S^R. De forma similar, la tasa de fallos de página para el algoritmo LRU aplicado a S es igual que la tasa de fallos de página del algoritmo LRU aplicado a S^R).

El resultado de aplicar el algoritmo de sustitución LRU a nuestra cadena de referencia de ejemplo se muestra en la Figura 9.15. El algoritmo LRU produce 12 fallos de página. Observe que los primeros cinco fallos coinciden con los del algoritmo óptimo de sustitución. Sin embargo, cuando se produce la referencia a la página 4, el algoritmo de sustitución LRU comprueba que, de los tres marcos que hay en la memoria, la página 2 es la que ha sido utilizada menos recientemente. Por tanto, el algoritmo LRU sustituye la página 2, al no ser consciente de que la página 2 va a ser utilizada en breve. Cuando se produce a continuación el fallo de página correspondiente a la página 2, el algoritmo LRU sustituye la página 3, ya que ahora será la menos recientemente utilizada de las tres páginas que hay en la memoria. A pesar de estos problemas, una sustitución LRU con 12 fallos de página es mucho mejor que una sustitución FIFO con 15 fallos de página.

Esta política LRU se utiliza a menudo como algoritmo de sustitución de páginas y se considera que es bastante buena. El principal problema es *cómo* implementar ese mecanismo de sustitución LRU. Un algoritmo LRU de sustitución de páginas puede requerir una considerable asistencia hardware. El problema consiste en determinar un orden para los marcos definido por el instante correspondiente al último uso. Existen dos posibles implementaciones:

- **Contadores**. En el caso más simple, asociamos con cada entrada en la tabla de páginas un campo de tiempo de uso y añadimos a la CPU un reloj lógico o contador. El reloj se incrementa con cada referencia a memoria. Cuando se realiza una referencia a una página, se copia el contenido del registro de reloj en el campo de tiempo de uso de la entrada de la tabla de páginas correspondiente a dicha página. De esta forma, siempre tenemos el "tiempo" de la última referencia a cada página y podremos sustituir la página que tenga el valor temporal menor. Este esquema requiere realizar una búsqueda en la tabla de páginas para localizar la página menos recientemente utilizada y realizar una escritura en memoria (para continuar el campo de tiempo de uso en la tabla de páginas) para cada acceso a memoria. Los tiempos deben también mantenerse apropiadamente cuando se modifiquen las tablas de páginas (debido a las actividades de planificación de la CPU). Asimismo, es necesario tener en cuenta el desbordamiento del reloj.

- **Pila**. Otra técnica para implementar el algoritmo de sustitución LRU consiste en mantener una pila de números de página. Cada vez que se hace referencia a una página, se extrae esa página de la pila y se la coloca en la parte superior. De esta forma, la página más recientemente utilizada se encontrará siempre en la parte superior de la pila y la menos recientemente utilizada en la inferior (Figura 9.16). Puesto que es necesario eliminar entradas de la parte intermedia de la pila, lo mejor para implementar este mecanismo es utilizar una lista doblemente enlazada con un puntero a la cabecera y otro a la cola. Entonces, eliminar una página y colocarla en la parte superior de la pila requiere modificar seis punteros en el caso peor. Cada actualización es algo más cara que con el otro método, pero no hay necesi-

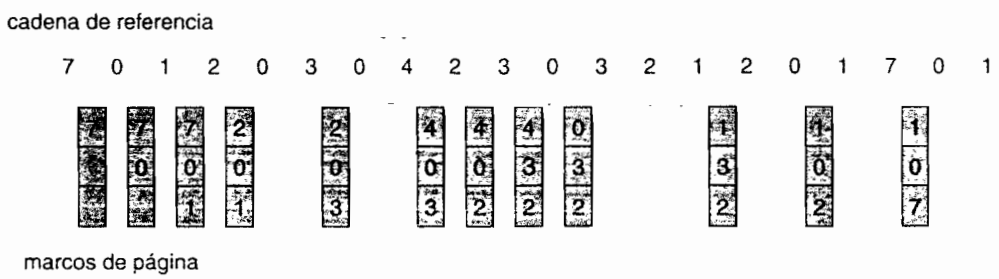

Figura 9.15 Algoritmo LRU de sustitución de páginas.

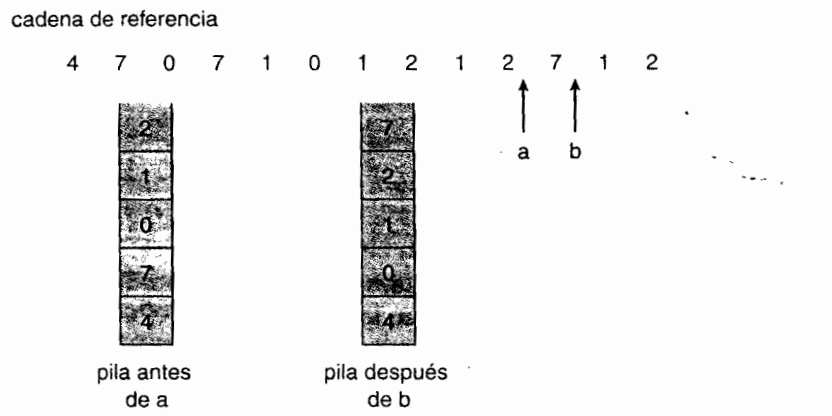

Figura 9.16 Utilización de una pila para registrar las referencias de página más recientes.

dad de buscar la página que hay que sustituir: el puntero de cola siempre apuntará a la parte inferior de la pila, que será la página menos recientemente utilizada. Esta técnica resulta particularmente apropiada para las implementaciones de algoritmos de sustitución LRU realizadas mediante software o microcódigo.

Al igual que el algoritmo óptimo de sustitución, el algoritmo de sustitución LRU no sufre la anomalía de Belady. Ambos algoritmos pertenecen a una clase de algoritmos de sustitución de páginas, denominada **algoritmos de pila**, que jamás pueden exhibir la anomalía de Belady. Un algoritmo de pila es un algoritmo para el que puede demostrarse que el conjunto de páginas en memoria para n marcos es siempre un *subconjunto* del conjunto de páginas que habría en memoria con $n + 1$ marcos. Para el algoritmo de sustitución LRU, el conjunto de páginas en memoria estaría formado por las n páginas más recientemente referenciadas. Si se incrementa el número de marcos, estas n páginas seguirán siendo las más recientemente referenciadas y, por tanto, continuarán estando en la memoria.

Observe que ninguna de las dos implementaciones del algoritmo LRU sería concebible sin disponer de una asistencia hardware más compleja que la que proporcionan los registros TLB estándar. La actualización de los campos de reloj o de la pila debe realizarse para *todas* las referencias de memoria. Si utilizáramos una interrupción para cada referencia con el fin de permitir la actualización por software de dichas estructuras de datos, todas las referencias a memoria se ralentizarían en un factor de al menos diez, ralentizando también según un factor de diez todos los procesos de usuario. Pocos sistemas podrían tolerar este nivel de carga de trabajo adicional debida a la gestión de memoria.

9.4.5 Sustitución de páginas mediante aproximación LRU

Pocos sistemas informáticos proporcionan el suficiente soporte hardware como para implementar un verdadero algoritmo LRU de sustitución de páginas. Algunos sistemas no proporcionan soporte hardware en absoluto, por lo que deben utilizarse otros algoritmos de sustitución de páginas (como por ejemplo el algoritmo FIFO). Sin embargo, dichos sistemas proporcionan algo de ayuda, en la forma de un **bit de referencia**. El bit de referencia para una página es activado por el hardware cada vez que se hace referencia a esa página (cada vez que se lee o escribe cualquier byte de dicha página). Los bits de referencia están asociados con cada entrada de la tabla de páginas.

Inicialmente, todos los bits son desactivados (con el valor 0) por el sistema operativo. A medida que se ejecuta un proceso de usuario, el bit asociado con cada una de las páginas referenciadas es activado (puesto a 1) por el hardware. Después de un cierto tiempo, podemos determinar qué páginas se han utilizado y cuáles no examinando los bits de referencia, aunque no podremos saber el *orden* de utilización de las páginas. Esta información constituye la base para muchos algoritmos de sustitución de páginas que traten de aproximarse al algoritmo de sustitución LRU.

9.4.5.1 Algoritmo de los bits de referencia adicionales

Podemos disponer de información de ordenación adicional registrando los bits de referencia a intervalos regulares. Podemos mantener un byte de 8 bits para cada página en una tabla de memoria. A intervalos regulares (por ejemplo, cada 100 milisegundos), una interrupción de temporización transfiere el control al sistema operativo. El sistema operativo desplaza el bit de referencia de cada página, transfiriéndolo al bit de mayor peso de ese byte de 8 bits, desplazando asimismo los otros bits una posición hacia la derecha descartando el bit de menor peso. Estos registros de desplazamiento de 8 bits contienen el historial de uso de las páginas en los 8 últimos períodos temporales. Por ejemplo, si el registro de desplazamiento contiene 00000000, entonces la página no habrá sido utilizada durante ocho períodos temporales; una página que haya sido utilizada al menos una vez en cada período tendrá un valor de su registro de desplazamiento igual a 11111111. Una página con un valor del registro de historial igual a 11000100 habrá sido utilizada más recientemente que otra con un valor de 01110111. Si interpretamos estos bytes de 8 bits como enteros sin signo, la página con el número más bajo será la menos recientemente utilizada y podremos sustituirla. Observe, sin embargo, que no se garantiza la unicidad de esos números. Por ello, podemos sustituir (descargar) todas las páginas que tengan el valor más pequeño o utilizar el método FIFO para elegir una de esas páginas.

El número de bits de historial puede, por supuesto, variarse y se selecciona (dependiendo del hardware disponible) para hacer que la actualización sea lo más rápida posible. En el caso extremo, ese número puede reducirse a cero, dejando sólo el propio bit de referencia. Este algoritmo se denomina **algoritmo de segunda oportunidad para la sustitución de páginas**.

9.4.5.2 Algoritmo de segunda oportunidad

El algoritmo básico de segunda oportunidad para la sustitución de páginas es un algoritmo de sustitución FIFO. Sin embargo, cuando se selecciona una página, inspeccionamos su bit de referencia. Si el valor es 0, sustituimos dicha página, pero si el bit de referencia tiene el valor 1, damos a esa página una segunda oportunidad y seleccionamos la siguiente página de la FIFO. Cuando una página obtiene una segunda oportunidad, su bit de referencia se borra y el campo que indica su instante de llegada se reconfigura, asignándole el instante actual. Así, una página a la que se le haya dado una segunda oportunidad no será sustituida hasta que todas las demás páginas hayan sido sustituidas (o se les haya dado también una segunda oportunidad). Además, si una página se utiliza de forma lo suficientemente frecuente como para que su bit de referencia permanezca activado, nunca será sustituida.

Una forma de implementar el algoritmo de segunda oportunidad (algunas veces denominado algoritmo del *reloj*) es una cola circular. Con este método, se utiliza un puntero (es decir, una manecilla del reloj) para indicar cuál es la siguiente página que hay que sustituir. Cuando hace falta un marco, el puntero avanza hasta que encuentra una página con un bit de referencia igual a 0. Al ir avanzando, va borrando los bits de referencia (Figura 9.17). Una vez que se encuentra una página víctima, se sustituye la página y la nueva página se inserta en la cola circular en dicha posición. Observe que, en el peor de los casos, cuando todos los bits estén activados, el puntero recorrerá la cola completa, dando a cada una de las páginas una segunda oportunidad y borrando todos los bits de referencia antes de seleccionar la siguiente página que hay que sustituir. El algoritmo de sustitución de segunda oportunidad degenera convirtiéndose en un algoritmo de sustitución FIFO si todos los bits están activados.

9.4.5.3 Algoritmo mejorado de segunda oportunidad

Podemos mejorar el algoritmo de segunda oportunidad considerando el bit de referencia y el bit de modificación (descrito en la Sección 9.4.1) como una pareja ordenada. Con estos dos bits, tenemos los cuatro casos posibles siguientes:

1. (0, 0) no se ha utilizado ni modificado recientemente: es la mejor página para sustitución.

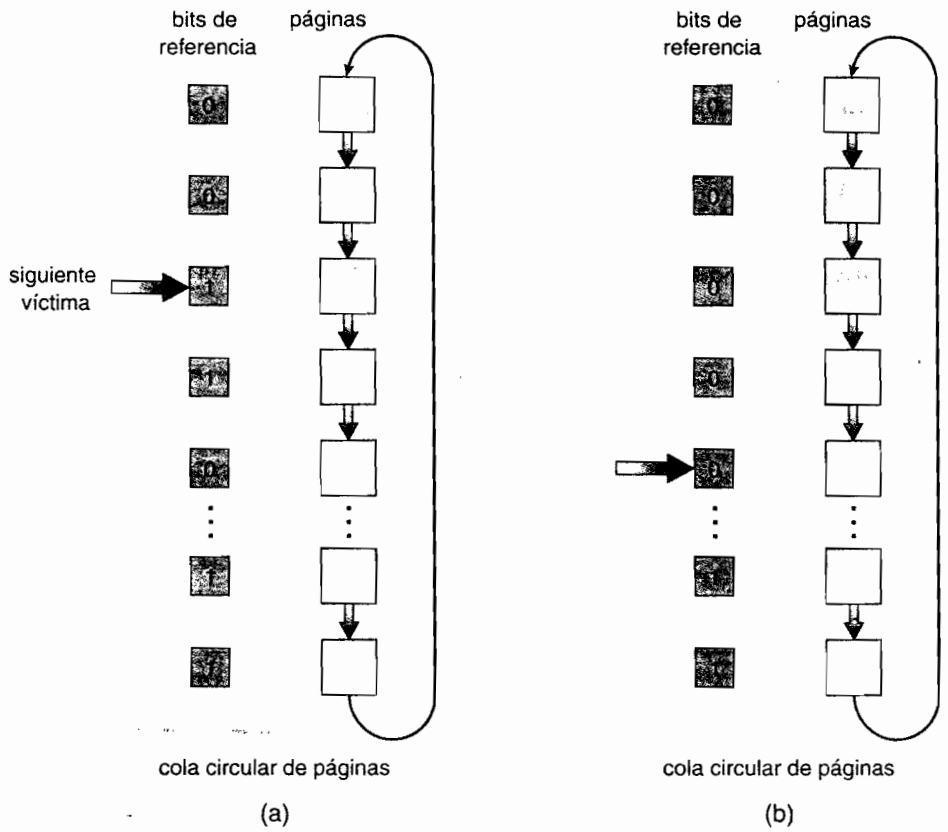

Figura 9.17 Algoritmo de sustitución de páginas de segunda oportunidad (del reloj).

2. (0, 1) no se ha usado recientemente pero sí que se ha modificado: no es tan buena como la anterior, porque será necesario escribir la página antes de sustituirla.
3. (1, 0) se ha utilizado recientemente pero está limpia: probablemente se la vuelva a usar pronto.
4. (1, 1) se la ha utilizado y modificado recientemente: probablemente se la vuelva a usar pronto y además sería necesario escribir la página en disco antes de sustituirla.

Cada página pertenece a una de estas cuatro clases. Cuando hace falta sustituir una página, se utiliza el mismo esquema que en el algoritmo del reloj; pero en lugar de examinar si la página a la que estamos apuntando tiene el bit de referencia puesto a 1, examinamos la clase a la que dicha página pertenece. Entonces, sustituimos la primera página que encontremos en la clase no vacía más baja. Observe que podemos tener que recorrer la cola circular varias veces antes de encontrar una página para sustituir.

La principal diferencia entre este algoritmo y el algoritmo más simple del reloj es que aquí damos preferencia a aquellas páginas que no hayan sido modificadas, con el fin de reducir el número de operaciones de E/S requeridas.

9.4.6 Sustitución de páginas basada en contador

Hay muchos otros algoritmos que pueden utilizarse para la sustitución de páginas. Por ejemplo, podemos mantener un contador del número de referencias que se hayan hecho a cada página y desarrollar los siguientes dos esquemas:

- El **algoritmo de sustitución de páginas LFU (least frequently used, menos frecuentemente utilizada)** requiere sustituir la página que tenga el valor más pequeño de contador. La razón para esta selección es que las páginas más activamente utilizadas deben tener un

valor grande en el contador de referencias. Sin embargo, puede surgir un problema cuando una página se utiliza con gran frecuencia durante la fase inicial de un proceso y luego ya no se la vuelve a utilizar. Puesto que se la emplea con gran frecuencia, tendrá un valor de contador grande y permanecerá en memoria a pesar de que ya no sea necesaria. Una solución consiste en desplazar una posición a la derecha los contenidos del contador a intervalos regulares, obteniendo así un contador de uso medio con decrecimiento exponencial.

- El **algoritmo de sustitución de páginas MFU (most frequently used, más frecuentemente utilizada)** se basa en el argumento de que la página que tenga el valor de contador más pequeño acaba probablemente de ser cargada en memoria y todavía tiene que ser utilizada.

Como cabría esperar, ni el algoritmo de sustitución LFU ni el MFU se utilizan de forma común. La implementación de estos algoritmos resulta bastante cara y tampoco constituyen buenas aproximaciones al algoritmo de sustitución OPT.

9.4.7 Algoritmos de búfer de páginas

Además de un algoritmo de sustitución de páginas específico, a menudo se utilizan otros procedimientos. Por ejemplo, los sistemas suelen mantener un conjunto compartido de marcos libres. Cuando se produce un fallo de página, se selecciona un marco víctima como antes. Sin embargo, la página deseada se lee en un marco libre extraído de ese conjunto compartido, antes de escribir en disco la víctima. Este procedimiento permite que el proceso se reinicie lo antes de posible, sin tener que esperar a que se descargue la página víctima. Posteriormente, cuando la víctima se descarga por fin, su marco se añade al conjunto compartido de marcos libre.

Una posible ampliación de este concepto consiste en mantener una lista de páginas modificadas. Cada vez que el dispositivo de paginación está inactivo, se selecciona una página modificada y se la escribe en el disco, desactivando a continuación su bit de modificación. Este esquema incrementa la probabilidad de que una página esté limpia en el momento de seleccionarla para sustitución, con lo que no será necesario descargarla.

Otra posible modificación consiste en mantener un conjunto compartido de marcos libres, pero recordando qué página estaba almacenada en cada marco. Puesto que el contenido de un marco no se modifica después de escribir el marco en el disco, la página antigua puede reutilizarse directamente a partir del conjunto compartido de marcos libres en caso de que fuera necesaria y si dicho marco no ha sido todavía reutilizado. En este caso, no sería necesaria ninguna operación de E/S. Cuando se produce un fallo de página, primero comprobamos si la página deseada se encuentra en el conjunto compartido de marcos libres. Si no está allí, deberemos seleccionar un marco libre y cargar en él la página deseada.

Esta técnica se utiliza en el sistema VAX/VMS, junto con un algoritmo de sustitución FIFO. Cuando el algoritmo de sustitución FIFO sustituye por error una página que continúa siendo activamente utilizada, dicha página vuelve a ser rápidamente extraída del conjunto de marcos libres y no hace falta ninguna operación de E/S. El búfer de marcos libre proporciona una protección frente al algoritmo FIFO de sustitución, que es relativamente poco eficiente, pero bastante simple. Este método es necesario porque las primeras versiones de VAX no implementaban correctamente el bit de referencia.

Algunas versiones del sistema UNIX utilizan este método en conjunción con el algoritmo de segunda oportunidad y dicho método puede ser una extensión útil de cualquier algoritmo de sustitución de páginas, para reducir el coste en que se incurre si se selecciona la página víctima incorrecta.

9.4.8 Aplicaciones y sustitución de páginas

En ciertos casos, las aplicaciones que acceden a los datos a través de la memoria virtual del sistema operativo tienen un rendimiento peor que si el sistema operativo no proporcionara ningún mecanismo de búfer. Un ejemplo típico sería una base de datos, que proporciona sus propios mecanismos de gestión de memoria y de búferes de E/S. Las aplicaciones como éstas conocen su

utilización de la memoria y del disco mejor de lo que lo hace un sistema operativo, que implementa algoritmos de propósito general. Si el sistema operativo proporciona mecanismos de búfer E/S y la aplicación también, se utilizará el doble de memoria para las operaciones de E/S.

Otro ejemplo serían los almacenes de datos, en los que frecuentemente se realizan lecturas de disco secuenciales masivas, seguidas de cálculos intensivos y escrituras. Un algoritmo LRU se encargaría a eliminar las páginas antiguas y a preservar las nuevas, mientras que la aplicación tendrá que leer más las páginas antiguas que las nuevas (a medida que comience de nuevo sus lecturas secuenciales). En un caso como éste, el algoritmo MFU sería, de hecho, más eficiente que el LRU.

Debido a tales problemas, algunos sistemas operativos dan a ciertos programas especiales la capacidad de utilizar una partición del disco como si fuera una gran matriz secuencial de bloques lógicos, sin ningún tipo de estructura de datos propia de los sistemas de archivos. Esta matriz se denomina en ocasiones **disco sin formato** y las operaciones de E/S que se efectúan sobre esta matriz se denominan E/S sin formato. La E/S sin formato puentea todos los servicios del sistema de archivos, como por ejemplo la paginación bajo demanda para la E/S de archivos, el bloqueo de archivos, la preextracción, la asignación de espacio, los nombres de archivo y los directorios. Observe que, aunque ciertas aplicaciones son más eficientes a la hora de implementar sus propios servicios de almacenamiento de propósito especial en una partición sin formato, la mayoría de las aplicaciones tienen un mejor rendimiento cuando utilizan los servicios normales del sistema de archivos.

9.5 Asignación de marcos

Vamos a volver ahora nuestra atención a la cuestión de la asignación. ¿Cómo asignamos la cantidad fija de memoria libre existente a los distintos procesos? Si tenemos 93 marcos libres y dos procesos, ¿cuántos marcos asignamos a cada proceso?

El caso más simple es el de los sistemas monousuario. Considere un sistema monousuario con 128 KB de memoria compuesta de páginas de 1 KB de tamaño. Este sistema tendrá 128 marcos. El sistema operativo puede ocupar 35 KB, dejando 93 marcos para el proceso de usuario. Con una paginación bajo demanda pura, los 93 marcos se colocarían inicialmente en la lista de marcos libres. Cuando un proceso de usuario comenzara su ejecución, generaría una secuencia de fallos de página. Los primeros 93 fallos de página obtendrían marcos extraídos de la lista de marcos libres. Una vez que se agotara la lista de marcos libres, se utilizaría un algoritmo de sustitución de páginas para seleccionar una de las 93 páginas en memoria con el fin de sustituirla por la página 94, y así sucesivamente. Cuando el proceso terminara, los 93 marcos volverían a ponerse en la lista de marcos libres.

Hay muchas variantes de esta estrategia simple. Podemos, por ejemplo, obligar al sistema operativo a asignar todo su espacio de búferes y de tablas a partir de la lista de marcos libres. Cuando este espacio no esté siendo utilizado por el sistema operativo, podrá emplearse para soportar las necesidades de paginación del usuario. Asimismo, podemos tratar de reservar en todo momento tres marcos libres dentro de la lista de marcos libres; así, cuando se produzca un fallo de página, siempre habrá un marco libre en el que cargar la página. Mientras que esté teniendo lugar el intercambio de la página, se puede elegir un sustituto, que será posteriormente escrito en disco mientras el proceso de usuario continúa ejecutándose. También son posibles otras variantes, pero la estrategia básica está clara: al proceso de usuario se le asignan todos los marcos libres.

9.5.1 Número mínimo de marcos

Nuestras estrategias para la asignación de marcos están restringidas de varias maneras. No podemos, por ejemplo, asignar un número de marcos superior al número total de marcos disponibles (a menos que existan mecanismos de compartición de páginas). Asimismo, debemos asignar al menos un número mínimo de marcos. En esta sección, vamos a examinar más detalladamente este último requisito.

Una razón para asignar al menos un número mínimo de marcos se refiere al rendimiento. Obviamente, a medida que el número de marcos asignados a un proceso se reduzca, se incremen-