Tema 2

Memoria virtual

En este tema se introduce el concepto de memoria virtual. Este tipo de sistemas, a diferencia de los estudiados en la asignatura Sistemas Operativos I, permite que un proceso se ejecute sin que su imagen esté cargada completamente en la memoria física. Este modo de funcionamiento necesita un soporte hardware, así como una serie de políticas del sistema operativo. Ambos aspectos serán tratados en este tema.

2.1. Introducción

Los sistemas clásicos de administración de la memoria exigen que un proceso esté cargado completo en ella para poder ejecutarse; esto tiene el inconveniente de que no podemos tener procesos más grandes que la memoria física disponible. Una forma de vencer esta limitación consiste en dividir los programas en fragmentos independientes que puedan ejecutarse de forma separada, que se conocen con el nombre de **superposiciones**. Este sistema tenía el inconveniente de que era el programador el que debía realizar la división.

El concepto de **memoria virtual** supone un avance importante en este campo ya que permite la ejecución de procesos que no están cargados en memoria física en su totalidad. Además tiene la ventaja de liberar al programador de especificar

qué partes deben estar cargadas y cuáles no, ya que todo este trabajo lo realiza el sistema operativo.

Podemos definir los sistemas de memoria virtual como el conjunto de técnicas de gestión de memoria que nos permite ejecutar un proceso sin que su espacio lógico de direcciones se encuentre en memoria física en su totalidad. Éstos permiten abstraer al usuario de las características de la memoria física, tales como su tamaño o el hecho de que esté compartida por otros procesos.

En estos sistemas sólo se mantienen en memoria física aquellas partes del proceso que se necesitan para su ejecución, el resto se guarda en memoria secundaria. Si durante la ejecución del proceso se hace referencia a un dato o instrucción que no resida en memoria física será necesario transferirlo desde la memoria secundaria. Dado que esto puede producir una disminución de la velocidad de ejecución de los procesos y, por tanto, del rendimiento del sistema, en su momento se mantuvieron muchos debates sobre su efectividad. Sin embargo, la experiencia ha demostrado que este tipo de sistemas proporciona un rendimiento adecuado, y de hecho es el sistema de gestión de memoria que se suele emplear actualmente.

En los sistemas de memoria virtual se suele emplear el término **memoria real** para designar a la memoria física, en contraposición al concepto de memoria virtual. Ésta es una memoria mucho más grande, ya que abarca tanto la memoria física como el almacenamiento secundario. La parte de un proceso que está actualmente en memoria principal se denomina **conjunto residente**.

Las ventajas principales que proporciona la memoria virtual es que el tamaño de la imagen de un proceso puede ser mayor al de la memoria física disponible. Como consecuencia de esto, una ventaja aún más importante es que la suma de los espacios direccionables de los procesos activos, puede exceder la capacidad de la memoria física. Además, libera al programador de conocer los límites impuestos por la capacidad de almacenamiento real del sistema.

Los sistemas de memoria virtual se pueden implementar basándose en la paginación, en la segmentación, o en una combinación de ambas; en este último caso el tratamiento es muy similar al de la paginación puesto que se utilizan segmentos divididos en páginas. El estudio que se realiza en este tema se va a centrar fundamentalmente en los sistemas de memoria virtual paginados.

2.2. El principio de localidad

El buen funcionamiento de los sistemas de memoria virtual está basado en el **principio de localidad**. Éste establece que las referencias al código y a los datos dentro de un proceso tienden a estar agrupadas durante períodos cortos de tiempo. Durante períodos largos, estos grupos van cambiando.

El análisis del funcionamiento de los programas reales nos puede ayudar a comprender el principio de localidad:

Ejecución secuencial El flujo de ejecución de un programa suele ser secuencial, salvo en las instrucciones de salto y de llamada, que suponen una pequeña parte del total. Por tanto, la próxima instrucción a ejecutar será normalmente la siguiente a la actual.

Bucles Los bucles suelen constar de un número de instrucciones relativamente pequeño que se repiten muchas veces. Durante la ejecución de una iteración, sólo se están empleando un número pequeño de instrucciones, que se repiten mientras dure la ejecución del bucle.

Código de uso poco frecuente En la mayor parte de los programas suele haber secciones de código que se ejecutan con poca frecuencia. Por ejemplo, las destinadas a la gestión de errores, los correctores sintácticos de los editores, etc.

Procesamiento de estructuras La utilización de estructuras de datos grandes, tales como matrices, emplean normalmente un procesamiento secuencial referenciándose elementos contiguos en la estructura[1].

Espacio reservado para estructuras Normalmente, ciertas estructuras de datos, como matrices, listas y tablas, se les suele asignar más memoria de la que realmente necesitan.

A la vista de estas consideraciones se puede apreciar como, en muchas ocasiones, no es necesario disponer de toda la imagen del proceso en memoria. Incluso en aquellos casos en los que se utilice por completo, no se necesitará en su totalidad durante todo el tiempo que dure la ejecución.

2.3. Fundamentos de la paginación

La paginación fue utilizada por primera vez por los diseñadores del computador Atlas para proporcionar solamente una memoria virtual grande, ya que el sistema no contemplaba inicialmente multiprogramación. Este computador fue construido en la Universidad de Manchester en 1960.

La paginación considera la memoria física dividida en bloques llamados **marcos**, todos del mismo tamaño. La memoria lógica también se considera dividida en bloques del mismo tamaño que los marcos, denominados **páginas**. Al cargar un proceso en memoria, sus páginas pueden ser cargadas en cualquier marco disponible, no siendo necesario que éstos estén contiguos.

La paginación, a diferencia de los esquemas anteriores, introduce una divergencia entre la visión que los usuarios tienen de los procesos en memoria y su situación

[1]Hay que hacer notar, que la forma en que las estructuras de datos están almacenadas en memoria juega un papel importante en el comportamiento de la localidad. Por ejemplo, una matriz bidimensional almacenada por columnas pero procesada por filas podría requerir acceder a datos ubicados físicamente dispersos.

real. Los usuarios consideran que sus procesos ocupan zonas contiguas de memoria, pero si se usa un esquema de paginación, los procesos estarán dispersos en ella. La traducción de direcciones lógicas a físicas es la que hace concordar estas dos visiones. Este proceso se lleva a cabo en tiempo de ejecución y para no degradar el rendimiento del sistema se realiza con ayuda de un hardware adecuado. La figura 2.1 representa la memoria lógica y física en el esquema de paginación, y las estructuras necesarias para su control.

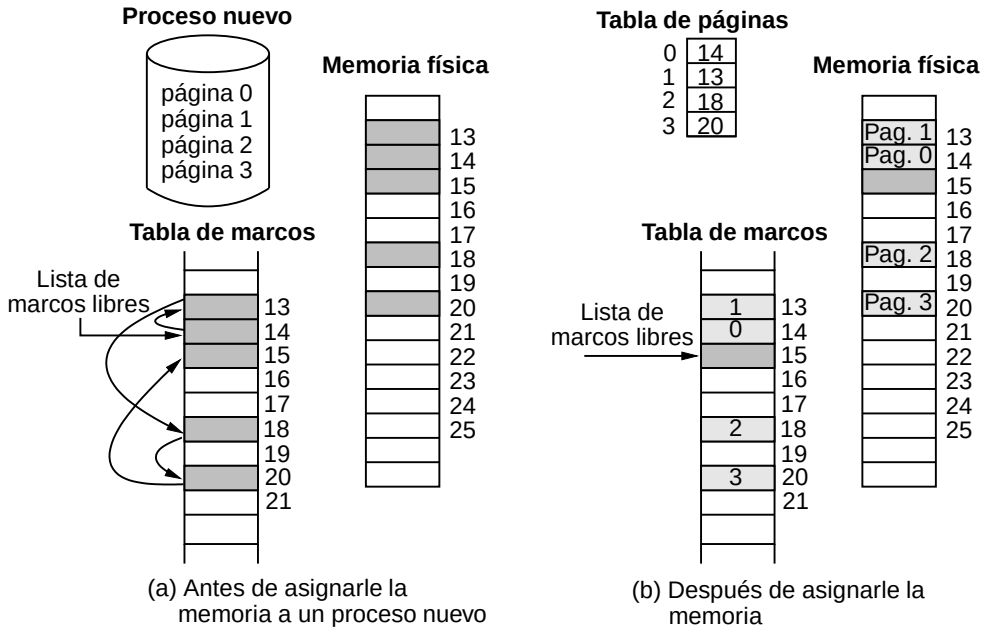

Figura 2.1: Carga de un proceso en un sistema de paginación

Cada página necesita un marco de memoria física. Dado que las páginas no necesitan estar contiguas, los marcos pueden estar en cualquier lugar. El sistema operativo mantiene la información sobre qué marcos están libres y cuáles ocupados en una estructura de datos llamada **tabla de marcos** (apartado 2.6.1).

En un sistema de memoria virtual paginado, cada proceso sólo mantiene en memoria principal unas cuantas páginas del total que componen su imagen, esto es lo que se conoce como conjunto residente. La imagen completa del proceso se sitúa en memoria secundaria, y sólo se traen a memoria principal las páginas que se necesitan en cada instante. Las decisiones de qué páginas traer, cuándo traerlas, y dónde ubicarlas son las que afectan al gestor de memoria virtual. Para saber donde está cargada cada página de un proceso, el sistema mantiene para cada uno de ellos una estructura conocida como **tabla de páginas** (apartado 2.6.2).

2.3.1. Protección

En los sistemas paginados la protección comprende dos aspectos, la comprobación de la validez de las direcciones y el modo de acceso a las páginas.

El primer aspecto hace referencia a la comprobación que hace el sistema para evitar que un proceso acceda al espacio de direcciones de otro. Para ello, se añade un bit de validez a cada entrada de la tabla de páginas que indicará si la página pertenece o no al proceso. Otra posibilidad es utilizar un registro de longitud de la tabla de páginas que contiene el número de páginas del proceso. Cualquier dirección lógica generada se comprueba con este valor para saber si está dentro del rango permitido.

Dependiendo del contenido de las páginas se podrá acceder a ellas de un modo u otro. Para controlar esto se asocian unos bits de acceso a cada entrada de la tabla de páginas, sólo lectura o lectura-escritura. De este modo, el sistema podrá comprobar fácilmente si el proceso está intentando acceder a la página de forma adecuada.

2.3.2. Páginas compartidas

Si varios procesos ejecutan el mismo código, sería conveniente que todos ellos pudieran compartir los marcos de memoria donde reside éste, ya que esto supondría un ahorro considerable de memoria. Los sistemas de paginación permiten hacer esto siempre, claro está, que el código sea reentrante, es decir, no automodificable (el código no cambia durante su ejecución).

Así, si varios procesos comparten el mismo código, éste estará cargado en una serie de marcos compartidos, mientras que los datos de cada proceso estarán en sus propios marcos. Si esto es así, habrá varias entradas de distintas tablas de páginas que apuntarán a los mismos marcos. La tabla de marcos debe contemplar también esta compartición añadiendo un nuevo campo a cada entrada que sea un contador del número de procesos que están compartiéndolo. Esto es necesario porque cuando uno de los procesos que está compartiendo ese marco termine, se debe comprobar el contador para saber si todavía hay procesos que lo estén usando. De este modo, cada proceso al terminar decrementa el contador en una unidad, y cuando llegue a 0 es cuando el marco queda libre.

2.3.3. Fragmentación

Cuando se usa un esquema de paginación no aparece fragmentación externa, ya que cualquier marco puede ser asignado a un proceso que lo necesite. Sin embargo, sí se puede dar fragmentación interna. Si los requisitos de memoria de un proceso no son un múltiplo entero del tamaño de una página, dado que los marcos son asignados como unidades, el último marco asignado no se aprovechará completo.

Si el tamaño del proceso es independiente del tamaño de la página, podemos esperar una fragmentación interna media de media página por proceso.

2.4. Fundamentos de la segmentación

Otro esquema de administración de la memoria en el que no se exige que el proceso esté cargado en una zona contigua y que se puede utilizar en los sistemas de memoria virtual es la segmentación. En este caso el programa y los datos se dividen en **segmentos**, que pueden tener tamaños diferentes. Esta división de los procesos en segmentos se corresponde mejor que la paginación con la visión que tienen los usuarios de ellos. En la figura 2.2 se puede observar la visión que tiene el usuario de la imagen de un proceso. En ella los ficheros fuente que constituyen el programa se transforman en un espacio de direcciones lógicas constituido por diversos segmentos. En un sistema de segmentación, éstos se cargan en memoria en los huecos disponibles del tamaño adecuado y se crea una **tabla de segmentos** para su gestión.

Además de esta estructura, se necesita llevar el control de qué zonas de la memoria están libres y cuáles ocupadas. Esto se puede hacer mediante listas enlazadas o mapas de bits, al igual que en los sistemas de particiones variables (ver apartado 1.7.3).

2.4.1. Compartición y protección

Una ventaja de la segmentación es la posibilidad de compartir segmentos. Cuando dos o más procesos comparten segmentos, varias entradas de las tablas de segmentos de los procesos apuntarán a la misma dirección física. Para conseguir esta compartición, la tabla de segmentos debe tener información de cuántos procesos comparten cada segmento mediante un contador. Cuando los procesos van finalizando y liberando sus segmentos, los contadores correspondientes disminuyen en una unidad, y cuando llegan a cero se libera el segmento.

Aunque la compartición parece sencilla, implica una serie de consideraciones. Los segmentos de código suelen contener referencias a sí mismos (por ejemplo, las funciones recursivas), que consisten en un número de segmento y un desplazamiento. Por tanto, todos los procesos que comparten el segmento deberán asignarle el mismo número. Los segmentos de datos de sólo lectura, así como los de código que no se refieran a sí mismos pueden tener distintos números.

La segmentación permite realizar la protección de los segmentos fácilmente. Dado que los segmentos son porciones bien definidas de los programas, es normal encontrarse con segmentos de datos y segmentos de código. Puesto que las instrucciones no se pueden modificar a sí mismas, los segmentos de código suelen ser de sólo lectura. Por contra, los segmentos de datos pueden ser de lectura-escritura. Para conseguir esta protección se incorporan los bits correspondientes en la tabla de segmentos.

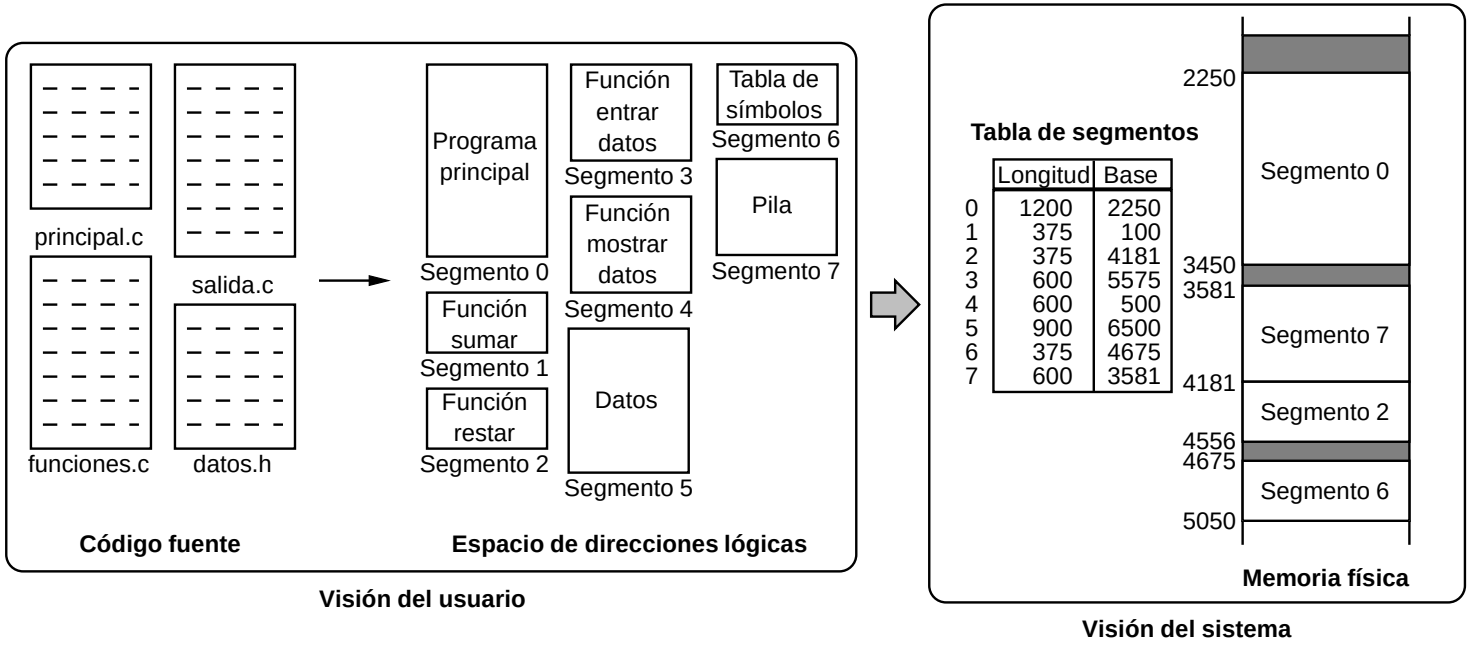

Figura 2.2: Esquema del sistema segmentado

2.4.2. Fragmentación y algoritmos de colocación

Un sistema de segmentación no presenta fragmentación interna, ya que cada segmento se aloja en una zona de memoria de su misma longitud. Sin embargo, sí puede presentar fragmentación externa si todos los bloques de memoria libre son demasiado pequeños para dar entrada a un nuevo segmento. Para reducir esta fragmentación se puede realizar compactación, al igual que en el sistema de particiones variables, pero con más facilidad al disponer de procesos reubicables.

Un aspecto relacionado con la fragmentación externa es la colocación de los segmentos en memoria. Dado que podemos encontrarnos con varios huecos en memoria y varios segmentos por ubicar, es necesario emplear un algoritmo de colocación de segmentos, como el mejor ajuste o el primer ajuste. Esta labor la realiza el planificador a largo plazo.

2.5. El área de intercambio

Un aspecto interesante en la memoria virtual es la gestión del área de intercambio. En los primeros sistemas de gestión de memoria basados en particiones, el área de intercambio desempeñaba una función bien distinta a la actual. Cuando en un sistema se encontraban varios procesos bloqueados, se podía suspender la ejecución de uno de ellos, descargándolo de memoria principal y pasándolo al área de intercambio. Posteriormente ese proceso volvería a memoria principal para continuar con su ejecución. Esta técnica de intercambio se empleaba en sistemas con una planificación de asignación por turnos, de forma que cuando terminaba el cuanto de un proceso, el administrador de memoria lo intercambiaba, para poder admitir un nuevo proceso al sistema.

Actualmente, esta técnica de intercambio de procesos se suele emplear en pocos sistemas. Así, en sistemas UNIX se emplea una modificación, que consiste en activarla solamente cuando la carga del sistema es elevada debido a la ocurrencia de numerosos fallos de página. Esto se estudiará con más detalle en el apartado 2.10.6.

A pesar de no emplearse la técnica de intercambio, el área de intercambio se sigue utilizando en la actualidad en los sistemas de memoria virtual. Existen diversos esquemas de utilización de esta área. En sistemas UNIX cuando se crea un proceso se asigna el espacio suficiente para alojar las páginas de código y datos en el área de intercambio. Cuando se inicia el proceso, las páginas de código se leen del sistema de ficheros y las de datos se traen también de éste o se crean (si no tienen valores iniciales). Cuando se realiza una sustitución para resolver un fallo de página, la página seleccionada se lleva al área de intercambio. De este modo, la primera vez que se referencia una página se lee del sistema de ficheros, y si es sustituida y vuelve a referenciarse lo hace desde el área de intercambio.

En otros sistemas como Windows NT o Solaris 2, cuando se hace referencia por primera vez a una página se trae del propio fichero ejecutable. Cuando una

página que ha sido modificada tiene que salir de memoria principal se escribe en el área de intercambio. De este modo, sólo se mantienen en ésta las páginas que han sido referenciadas y modificadas alguna vez. Las páginas de código que nunca se modifican, siempre se leen del propio fichero ejecutable; si tienen que salir de memoria se desechan.

Un aspecto relacionado con el área de intercambio es su ubicación. En algunos sistemas, como Windows NT, se utiliza un fichero grande dentro del propio sistema de ficheros para mantenerla. Esto permite aprovechar la propia estructura del sistema de ficheros (se verá en el tema 3) para gestionarla y que ésta presente un tamaño variable según las necesidades del sistema; sin embargo, las operaciones de E/S se realizan de forma más lenta. En otros, como UNIX, permiten que el área de intercambio pueda residir en una partición del disco; de este modo, al no necesitar ninguna estructura del sistema de ficheros, las operaciones de E/S se realizan de forma más rápida, pero requiere un gestor exclusivo del espacio del área de intercambio.

2.6. Estructuras hardware y de control

Para que los sistemas de memoria virtual sean prácticos y efectivos necesitan dos ingredientes. Por un lado, debe haber un soporte hardware para que se pueda emplear un esquema de paginación. Por otro, el sistema operativo debe incluir software adecuado para manejar el movimiento de páginas entre memoria secundaria y memoria principal. A continuación examinaremos los aspectos hardware, así como las estructuras de control necesarias que crea y mantiene el sistema operativo, pero que son utilizados por el hardware de manejo de memoria.

2.6.1. Tabla de marcos

Para asignar memoria a un proceso, el sistema operativo debe tener información sobre el número total de marcos y su estado (libre u ocupado). Ésta se mantiene en una estructura denominada tabla de marcos, que posee una entrada por cada marco de memoria física. El tamaño de la tabla de marcos es fijo y será igual al tamaño de la memoria principal dividido por el tamaño de un marco. Así,

$$TM = \frac{MF}{M}$$

donde TM es el número de entradas de la tabla de marcos, MF es el tamaño de la memoria física, y M es el tamaño del marco, que es igual al de la página.

Cada entrada de la tabla de marcos mantiene la siguiente información:

Bit de estado Indica si el marco está o no ocupado.

Página Número de página que lo ocupa.

Proceso Identificador del proceso propietario de la página.

Contador Número que procesos que comparten la página.

Bit de bloqueo Indica si el marco está o no bloqueado. Si lo está, la página que contiene no podrá ser sustituida. Un ejemplo de esto son los marcos que contienen al sistema operativo.

2.6.2. Tabla de páginas

En los sistemas de paginación existe una tabla de páginas por cada proceso activo, creándose durante la carga de éste en memoria. La tabla de páginas tiene tantas entradas como páginas pueda ocupar el proceso; cada una indica el número de marco donde está alojada.

La estructura de las entradas de las tablas de páginas depende del sistema, pero de forma general se suele encontrar en ellas la siguiente información:

Bit de presencia Dado que sólo algunas de las páginas del proceso pueden estar en memoria principal, se necesitará este bit para indicarlo. También se conoce como **bit de fallo**.

Número de marco Lugar de la memoria física donde se encuentra situada la página.

Bit de modificación Indica si el contenido de la página correspondiente ha sido alterado desde que fue cargada por última vez en memoria principal. Si no lo ha sido, no será necesario escribirla en disco cuando tenga que salir de la memoria.

Bit de referencia Señala si la página ha sido referenciada recientemente. Lo utilizan los algoritmos de sustitución de páginas.

Bits de protección Denotan las operaciones permitidas para la página.

Bit de compartición Muestra si la página puede o no ser compartida por varios procesos.

En los sistemas actuales, cuando la página no está presente en memoria (bit de presencia a 0), el resto de bits de la entrada se emplea para almacenar la dirección dentro de memoria secundaria donde se encuentra.

2.6.3. Tabla de segmentos

La segmentación emplea para la traducción de direcciones la tabla de segmentos. Cada proceso tiene su propia tabla, que contiene una entrada por cada segmento.

Éstas almacenan la dirección de comienzo del segmento en memoria física, así como su longitud, que se emplea para verificar que el proceso utilice direcciones válidas.

La tabla de segmentos puede contener además los bits de protección correspondientes, para controlar el modo de acceso al segmento.

2.7. Traducción de direcciones

La traducción de direcciones lógicas a físicas dependerá del sistema de memoria que dispongamos, paginado o segmentado. A continuación explicamos el proceso que se sigue en cada uno de dichos sistemas.

2.7.1. Traducción de direcciones en sistemas paginados

En un sistema paginado las direcciones lógicas tienen dos componentes, el número de página, p, y el desplazamiento dentro de ésta, d. El número de página se usa como índice para acceder a la tabla de páginas, que contiene el número de marco en el que reside la página. Este número de marco se combina con el desplazamiento para obtener la dirección física. La figura 2.3 representa este proceso de traducción. En ella aparece un registro hardware que apunta a la base de la tabla de páginas del proceso (PTBR, *Page-Table Base Register*) que está en ejecución. Cuando se produce un cambio de proceso es necesario cambiar el contenido de este registro para disponer de la nueva tabla de páginas.

El tamaño de las páginas y los marcos se establece en función del hardware disponible. Con objeto de facilitar el proceso de traducción se suele seleccionar una potencia de 2 para éste. Así, si el tamaño de una página es 2^s, el número de bits necesarios para direccionar una unidad de almacenamiento dentro de ésta es s. Por otro lado, si en nuestra máquina las direcciones lógicas son de t bits (el tamaño del espacio de direcciones lógicas es 2^t) y el tamaño de la página es 2^s, entonces los primeros $t-s$ bits designan el número de página, y los restantes s bits el desplazamiento dentro de ésta.

De forma análoga, la dirección física tiene la forma (m, d), donde m es el número de marco y d, el desplazamiento dentro de él. El número de bits de cada componente se determina a partir del tamaño del marco y del número de bits de la dirección física en nuestra máquina.

En el ejemplo que se muestra en la figura 2.3 partimos de una dirección lógica de 16 bits y un tamaño de página de 1 KiB (2^{10}). Por tanto, los 6 primeros bits constituyen el número de página y los 10 bits restantes, el desplazamiento. La dirección física del sistema es de 16 bits, por lo que presenta la misma estructura que la lógica. Supongamos la dirección lógica 0000110101111010, que corresponde a la página número 3 (000011) y a un desplazamiento de 378 (0101111010). Si ésta reside en el marco 2, la dirección física 0000100101111010 corresponde al marco número 2

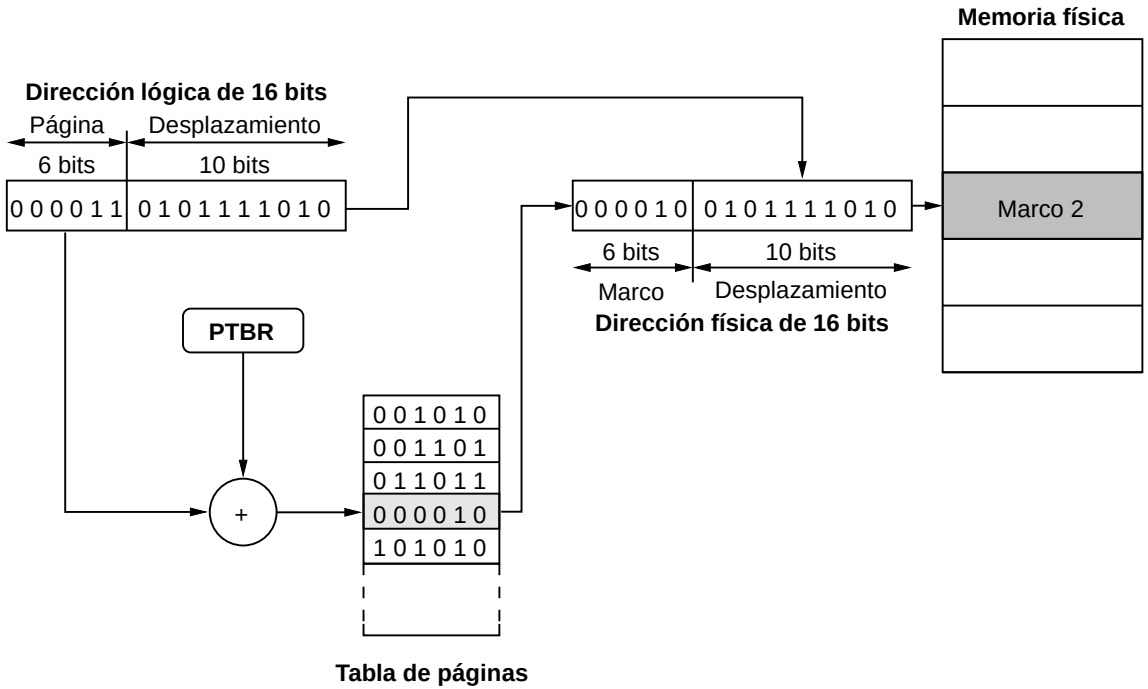

Figura 2.3: Traducción de direcciones en un sistema de paginación

(000010) y desplazamiento 378 (0101111010).

2.7.2. Fallos de página

Mientras el proceso se ejecute y referencie posiciones de memoria del conjunto residente no habrá ningún tipo de problema. Mediante la tabla de páginas, el sistema es capaz de determinar si las páginas referenciadas se encuentran o no en memoria. En este último caso, se genera una excepción que indica un fallo de acceso a memoria. Esto es lo que se conoce como **fallo de página**. Cuando se referencia una dirección virtual que no está presente en memoria física, tendremos que ir a la memoria secundaria para traer la página. La figura 2.4 muestra gráficamente los pasos a realizar:

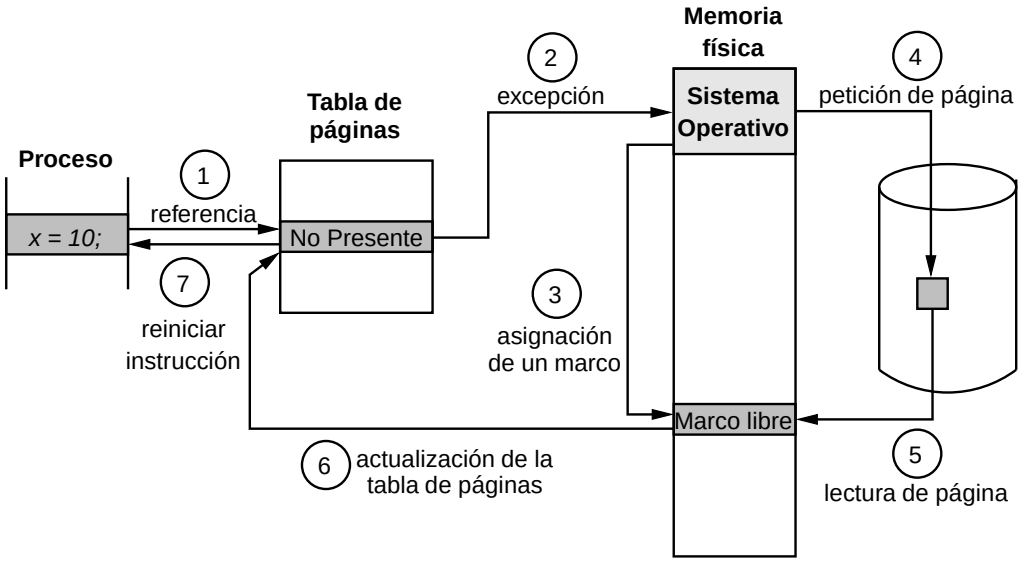

Figura 2.4: Pasos realizados durante un fallo de página

Paso 1 Cuando se produce una referencia a memoria durante la ejecución de un proceso, se consulta la tabla de páginas para realizar la traducción de direcciones.

Paso 2 Si la página no se encuentra en memoria se produce una excepción por parte del proceso y el sistema operativo toma el control. Éste lleva a cabo las siguientes acciones:

a) Realiza un cambio de contexto, es decir, se guarda la información que contienen los registros del procesador sobre el proceso.

b) Determina que la excepción producida se debe a un fallo de página.

c) Realiza las comprobaciones de protección, es decir, determina si la página es válida para el proceso y si el tipo de acceso es el adecuado. Si alguna de éstas falla, termina la ejecución del proceso.

Paso 3 Si la página es válida, el sistema intenta asignar un nuevo marco al proceso:

a) Si hay marcos libres, se le asigna uno y se marca como ocupado.

b) Si no los hay, tendrá que elegir uno de los ocupados para almacenar la nueva página (es lo que se conoce como **sustitución de página**). Si la página seleccionada ha sido modificada desde que fue introducida en memoria, tiene que ser reescrita en disco. El marco escogido se marca como ocupado.

Paso 4 Solicita al subsistema de E/S traer la nueva página del disco al marco libre anterior. El proceso que ocasionó el fallo de página pasa al estado de bloqueado a la espera de la página pedida, y el sistema selecciona un nuevo proceso para ejecutar. Es decir, se produce un cambio de proceso.

Paso 5 El dispositivo transfiere la página al marco seleccionado, produciendo una interrupción al finalizar. El sistema operativo atiende la interrupción y si se estaba ejecutando otro proceso habrá que hacer un nuevo cambio de contexto.

Paso 6 El sistema operativo actualiza la tabla de páginas para indicar que la página está en memoria. Cambia el estado al proceso que la esperaba, es decir, lo pasa del estado de bloqueado a listo.

Paso 7 El proceso espera a que el planificador a corto plazo le asigne de nuevo la CPU para continuar con la ejecución de la instrucción interrumpida.

Debido a que cuando se produce un fallo de página se interrumpe el proceso y se guarda su contexto, podemos reanudar su ejecución en la misma instrucción que produjo el fallo, con la diferencia de que la página solicitada se encuentra en memoria principal. De este modo, es posible ejecutar un proceso incluso si no dispone de páginas en la memoria física.

Durante la gestión de un fallo de página se pueden producir varias operaciones de lectura y/o escritura en disco. Debido a que en esos instantes el procesador permanecería ocioso se produce el cambio de proceso para mejorar el rendimiento.

Los fallos de página afectan de forma significativa al rendimiento del sistema, debido a que el tiempo de acceso efectivo a memoria aumenta, principalmente por las operaciones de E/S. Por lo tanto, un esquema de gestión de memoria virtual debe procurar que el número de fallos de páginas sea mínimo.

2.7.3. Traducción de direcciones en sistemas segmentados

Las direcciones lógicas en un sistema de segmentación vienen dadas por dos componentes: el número de segmento, s, y el desplazamiento, d. El primero se utiliza como índice para consultar la tabla de segmentos que almacena la dirección base de éste, que se combina con el desplazamiento para obtener la dirección física. Al

generarse las direcciones físicas en tiempo de ejecución, los procesos son reubicables. La figura 2.5 representa este proceso de traducción. En ella aparece el registro base de la tabla de segmentos (STBR, *Segment-Table Base Register*), y el registro de longitud de la tabla de segmentos (STLR, *Segment-Table Length Register*). El primero apunta a la tabla de segmentos del proceso en ejecución, y el segundo mantiene el número de segmentos de éste. Cuando se produce un cambio de proceso es necesario actualizarlos.

En los sistemas segmentados, como el tamaño de los segmentos es variable, se establece un tamaño máximo de éstos que determina el formato de la dirección lógica. En el ejemplo de la figura 2.5, la dirección lógica es de 16 bits, y se considera un tamaño máximo de segmento de 4 KiB; de este modo, los doce últimos bits indican el desplazamiento y los cuatro primeros el número de segmento. Así, la dirección lógica 0010001011100000 corresponde al segmento 2 y desplazamiento 736. Mediante la tabla de segmentos obtenemos la dirección de comienzo de éste en memoria física (0001000000010000), siendo la dirección física requerida la suma de ésta y el desplazamiento, 0001001011110000.

2.8. Segmentación paginada

Aparte de los sistemas de paginación y segmentación puros, existen sistemas que combinan ambas técnicas. Ejemplos de éstos son el sistema operativo MULTICS con la arquitectura GE 645 y el sistema OS/2 para Intel 386.

Éstos se caracterizan porque ofrecen al usuario una memoria lógica que particionan en segmentos, pero cada uno se almacena en un conjunto de páginas. De esta forma, se elimina la fragmentación externa y se resuelve el problema de la colocación. La figura 2.6[2] muestra el esquema de uno de estos sistemas. En ella, la dirección lógica consta de tres componentes, número de segmento, número de página y desplazamiento. El primero se usa como índice para acceder a la tabla de segmentos, cuya dirección base viene dada por el registro STBR, que proporciona la dirección base de la tabla de páginas. El número de página se usa como índice en esta tabla para obtener la dirección del marco. Finalmente, se concatena con el desplazamiento dando como resultado la dirección física deseada.

2.9. Aspectos de implementación de las tablas de páginas

En la implementación de las tablas de páginas, hemos de tener en cuenta dos aspectos, el aumento del tiempo de acceso a una dirección de memoria y el elevado número de entradas que pueden tener en los sistemas actuales.

[2]Para una mayor claridad, se ha omitido la comparación del número de segmento con la longitud de la tabla de segmentos, al igual que el desplazamiento con la longitud de éste.

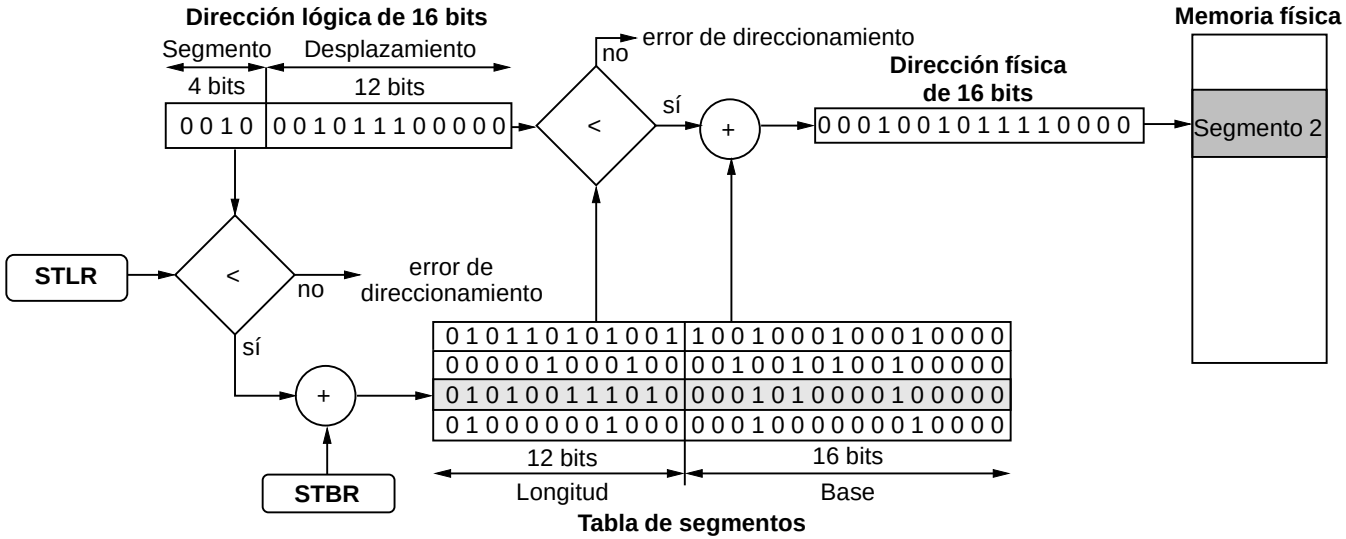

Figura 2.5: Traducción de direcciones en un sistema de segmentación

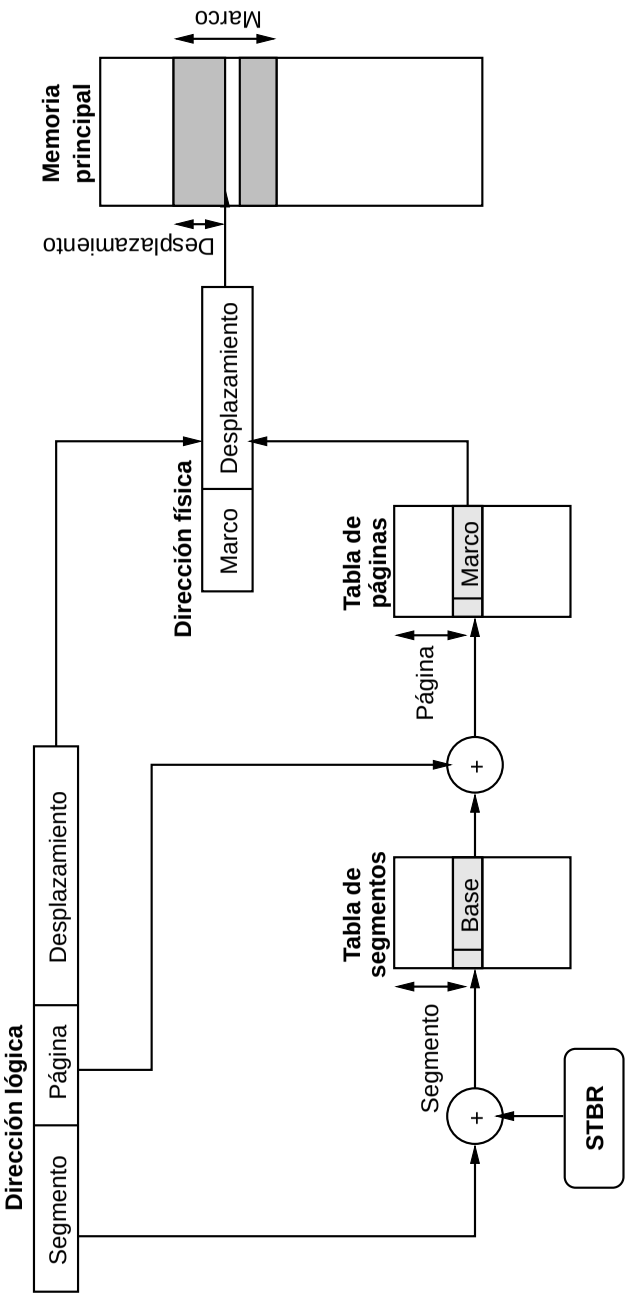

Figura 2.6: Traducción de direcciones en un sistema segmentado paginado

2.9.1. Reducción del tiempo de acceso

La tabla de páginas es un elemento vital para la traducción de direcciones en un sistema de paginación ya que establece la correspondencia entre la dirección lógica y la física. Este proceso se realiza en tiempo de ejecución, por lo que cada acceso a memoria supone acceder primero a la tabla de páginas y posteriormente a la dirección pedida.

Para disminuir el tiempo medio de acceso a memoria la tabla de página se ha implementado de diversas formas, en registros, o utilizando ***buffers* de traducción adelantada** (TLB, *Translation Lookaside Buffer*).

El uso de registros para mantener la tabla de páginas hace que el acceso a ésta sea muy rápido, pero tiene el inconveniente de que sólo puede usarse cuando el número de entradas de ésta es pequeño, de ahí que no se utilice en la actualidad.

Otra forma posible de acelerar el acceso a la tabla de páginas es mantenerla en memoria física y disponer de algunas de sus entradas en una TLB. Se trata de una memoria caché de traducciones que contiene aquellas entradas de la tabla de páginas usadas más recientemente. Este esquema se utiliza en los sistemas actuales.

Si se dispone de TLB el proceso de traducción de direcciones se modifica tal como aparece en la figura 2.7. Cuando se va a traducir una dirección lógica a física, se comprueba si la entrada de la tabla de páginas correspondiente se encuentra en la TLB. Si es así, no es necesario acceder a la tabla de páginas; en caso contrario habrá que acceder a ella.

Puesto que la TLB sólo contiene algunas de las entradas de la tabla de páginas, no se puede indexar por el número de página, sino que cada entrada deberá contener el número de página, además de la entrada de la tabla de páginas completa. Cuando se accede a la TLB se consultan todas sus entradas simultáneamente, acelerándose el proceso de búsqueda.

Como ocurre con la paginación, en los sistemas segmentados la tabla de segmentos se mantiene en memoria, por lo que se requieren dos accesos a ésta para acceder a cada dirección, uno a la tabla y otro a la dirección deseada. Con objeto de reducir este doble acceso se suele emplear un conjunto de registros asociativos para guardar las entradas de la tabla de segmentos que se han usado más recientemente, de forma similar a la TLB.

2.9.2. Reducción del tamaño de las tablas de páginas

El segundo aspecto a considerar es el elevado número de entradas de las tablas de páginas. En los sistemas de memoria virtual el tamaño de la tabla de páginas puede ser muy grande dado que se puede tener procesos de mayor tamaño que la memoria física, aprovechando de este modo todo el espacio de direcciones que permite la arquitectura. Hoy en día, la mayoría de los sistemas de computación soportan un espacio de direcciones lógicas muy grande (2^{32} o 2^{64}). En un ambiente de este tipo las

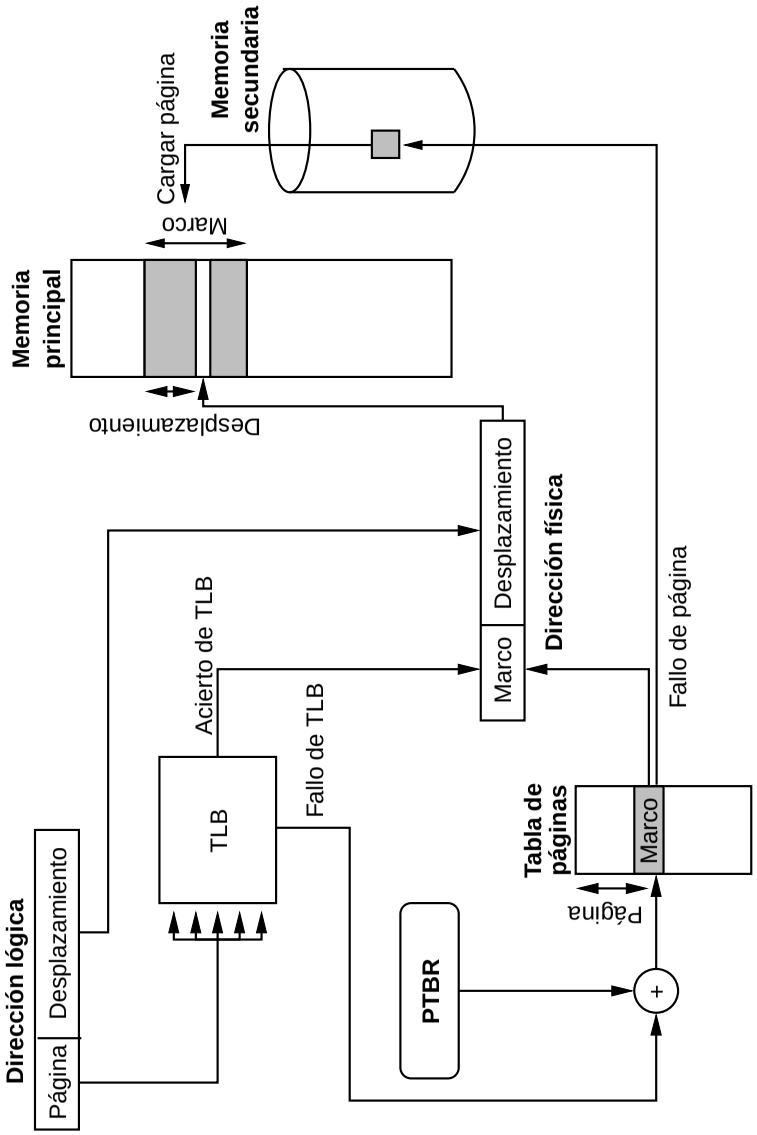

Figura 2.7: Traducción en un sistema de memoria virtual con TLB

tablas de páginas serían demasiado grandes. Por ejemplo, consideremos un sistema con un espacio de direcciones lógicas de 32 bits. Si el tamaño de la página es de 4 KiB, son necesarias 2^{20} entradas en la tabla de páginas, algo más de un millón de entradas. Si cada una tiene 4 bytes, se necesitan unos 4 MiB de memoria para almacenarlas. Además, hay que recordar que cada proceso necesita su propia tabla de páginas.

Se han propuesto diversos métodos para almacenar las tablas de páginas sin que sea necesario gastar tanta memoria. A continuación se verán dos de ellos, las **tablas de páginas multinivel**, y las **tablas de páginas invertidas**.

2.9.2.1. Tabla de páginas multinivel

Una solución sencilla para no tener que asignar un espacio de memoria contiguo tan grande a la tabla de páginas, consiste en dividirla en fragmentos más pequeños, es decir, paginar la tabla de páginas. La figura 2.8 muestra la diferencia entre utilizar una tabla de páginas convencional con un solo nivel, y una tabla de páginas de dos niveles. En este esquema hay una tabla de páginas de primer nivel, en la que cada entrada apunta a una tabla de páginas. Normalmente se establece una longitud máxima para la tabla de páginas, que suele ser igual al tamaño de una página.

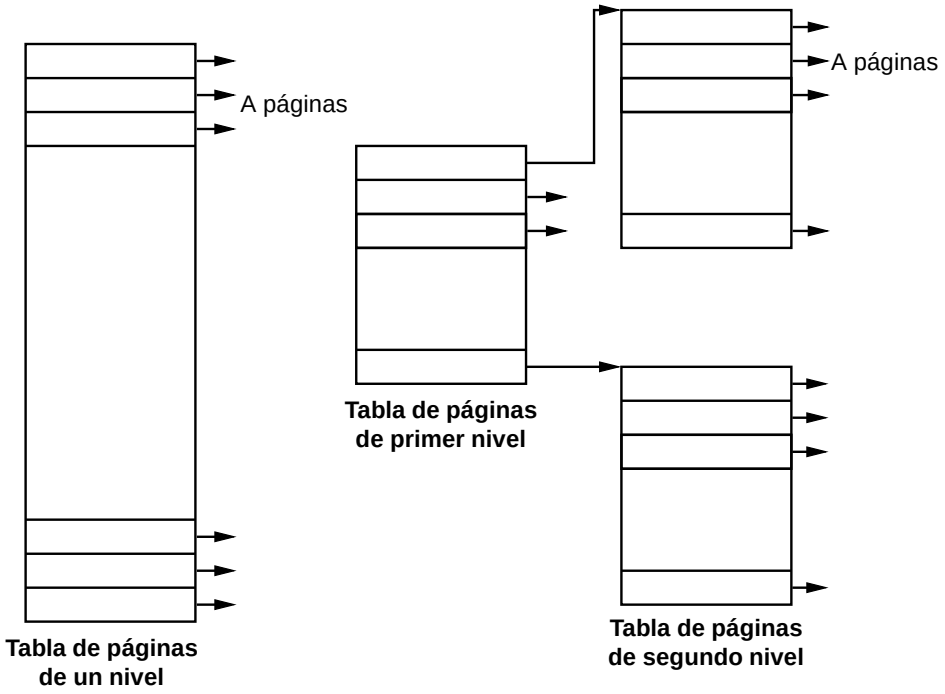

Figura 2.8: Tabla de páginas de un nivel y de dos niveles

Para ilustrar esta técnica usaremos un ejemplo con una máquina de 32 bits con páginas de 4 KiB y entradas de 4 bytes. Una dirección lógica está dividida en un número de página que consta de 20 bits y un desplazamiento de página que consta

de 12. Puesto que vamos a utilizar una tabla de páginas de dos niveles, el número de página deberá estar a su vez dividido en dos componentes. De esta forma las direcciones tendrían la siguiente estructura:

- Un campo, p_1, de 10 bits para el primer nivel, lo que supone una tabla de 1024 entradas, para que ocupe como mucho una página.
- Un campo, p_2, de 10 bits para el segundo nivel, que supone otras 1024 entradas.
- Un campo, d, de 12 bits para el desplazamiento puesto que la página es de 4 KiB.

El esquema de la traducción de direcciones para esta arquitectura se puede ver en la figura 2.9. Dada una dirección lógica, empleamos el primer campo para buscar en una tabla de primer nivel de 1024 entradas. Ésta a su vez apunta a otra tabla de 1024 entradas, utilizándose el segundo campo para escoger una entrada de ella, que es la que contiene la dirección base del marco buscado. A esta dirección se le suma el tercer campo y se obtiene la dirección física buscada. De este modo, en este sistema un proceso podrá ocupar $1024 * 1024$ páginas de 4 KiB cada una, empleando 1025 tablas para indicar dónde se encuentra almacenado (1 de primer nivel, y 1024 de segundo) cada una con 1024 entradas.

La ventaja de este esquema es que no se necesita mantener en memoria todas estas tablas a la vez, sino sólo unas pocas, manteniéndose el resto en memoria virtual. Además, si el proceso no ocupa todas las páginas disponibles, se necesitarán menos tablas ya que sólo se necesitan tantas de segundo nivel como entradas de las de primer nivel haya ocupadas.

Para un sistema con un espacio de direcciones lógicas de 64 bits, el esquema de paginación de 2 niveles ya no es apropiado. Para ilustrar esto, supongamos que el tamaño de página en tal sistema es de 4 KiB. En este caso, la tabla de páginas constará de 2^{52} entradas. Si usamos un esquema de paginación de dos niveles, entonces la tabla de páginas de primer nivel constará de 2^{26} entradas. Claramente, no podemos asignar la tabla de páginas de primer nivel de forma contigua en memoria (suponiendo 4 bytes por entrada, se requerirían 256 MiB de memoria física para almacenarla). La solución obvia es dividir la tabla de páginas de primer nivel en piezas más pequeñas, es decir, realizar un mayor número de niveles. Esta solución se usa también en algunos procesadores de 32 bits para obtener mayor flexibilidad y eficiencia. Por ejemplo, las arquitecturas SPARC y Motorola 68030, ambas de 32 bits, proporcionan un esquema de paginación de 3 y 4 niveles, respectivamente. Sin embargo, un mayor número de niveles implica más complejidad en el cálculo de la dirección física, y un mayor tiempo de acceso a la memoria.

2.9.2.2. Tabla de páginas invertida

Un enfoque alternativo es el empleo de las tablas de páginas invertidas. En este caso, sólo existe una en el sistema, y posee una entrada por cada marco de memo-

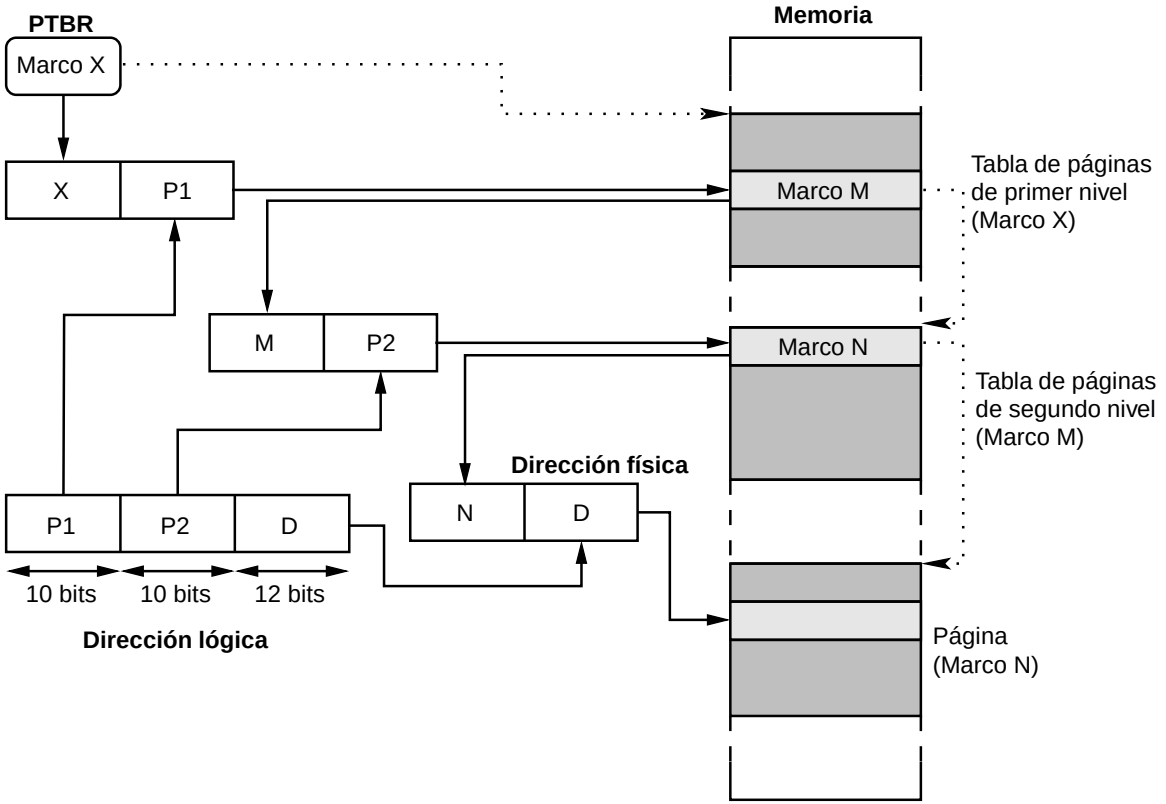

Figura 2.9: Tabla de páginas de dos niveles

ria. La estructura de las entradas de esta tabla de páginas difiere de la comentada anteriormente, ya que ahora debe mantener, además de la información anterior, el número de página y el identificador del proceso al que pertenece.

La figura 2.10 muestra la traducción de direcciones con una tabla de páginas invertida. En ella las direcciones lógicas consisten en una tripleta (id_proceso, página, desplazamiento). Cada entrada en la tabla de páginas es en realidad un par (id_proceso, página). Cuando se realiza una referencia a memoria, parte de la dirección lógica, en concreto (id_proceso, página), se presenta al subsistema de memoria y se busca una concordancia en la tabla de páginas. Si coincide en la entrada i, entonces la dirección física es (i, desplazamiento). Si no hay concordancia, entonces se produce un fallo de página.

Este esquema reduce la cantidad de memoria necesaria para almacenar la tabla de páginas. Sin embargo, aumenta el tiempo de búsqueda de una entrada, debido a que la tabla de páginas invertida está ordenada por direcciones físicas, y las búsquedas se realizan por direcciones lógicas. En algunos casos puede ser necesario recorrer toda la tabla para encontrar una coincidencia. Esto origina que la búsqueda sea demasiada larga.

Para aliviar el problema se utiliza una **tabla de dispersión**[3], para limitar la búsqueda a una o unas pocas entradas de la tabla, tal como se observa en la figura 2.10. Por supuesto, cada acceso a la tabla de dispersión añade una referencia más a memoria. De este modo una referencia a memoria virtual requiere al menos dos accesos a memoria real, una para la entrada de la tabla de dispersión y otra para la de páginas. Para mejorar el rendimiento, normalmente se emplean los registros asociativos o TLB. De este modo, si la página buscada se encuentra en estos registros no es necesario acudir a la tabla de páginas invertida.

La tabla de páginas invertida reduce considerablemente el tamaño de las tablas de páginas, pero sólo mantiene información de las páginas alojadas en memoria física. Sin embargo, para mantener la información completa se necesita una tabla de páginas externa por proceso. Éstas se utilizan solamente cuando la página referenciada no se encuentra actualmente en memoria. Su estructura es similar a las tradicionales, conteniendo información acerca de donde está localizada cada página virtual.

Estas tablas de páginas externas se mantienen en memoria secundaria, por lo que cuando se produzca un fallo de página, se necesitará un acceso adicional a memoria secundaria. Por tanto, existe una demora extra en el procesamiento de búsqueda de una página.

Ejemplos de sistemas que utilizan este esquema son el IBM System/38, el IBM RISC System 6000, IBM RT, IBM AS/400 y las estaciones de trabajo Hewlett-Packard Spectrum.

[3]También denominada tabla *hash*.

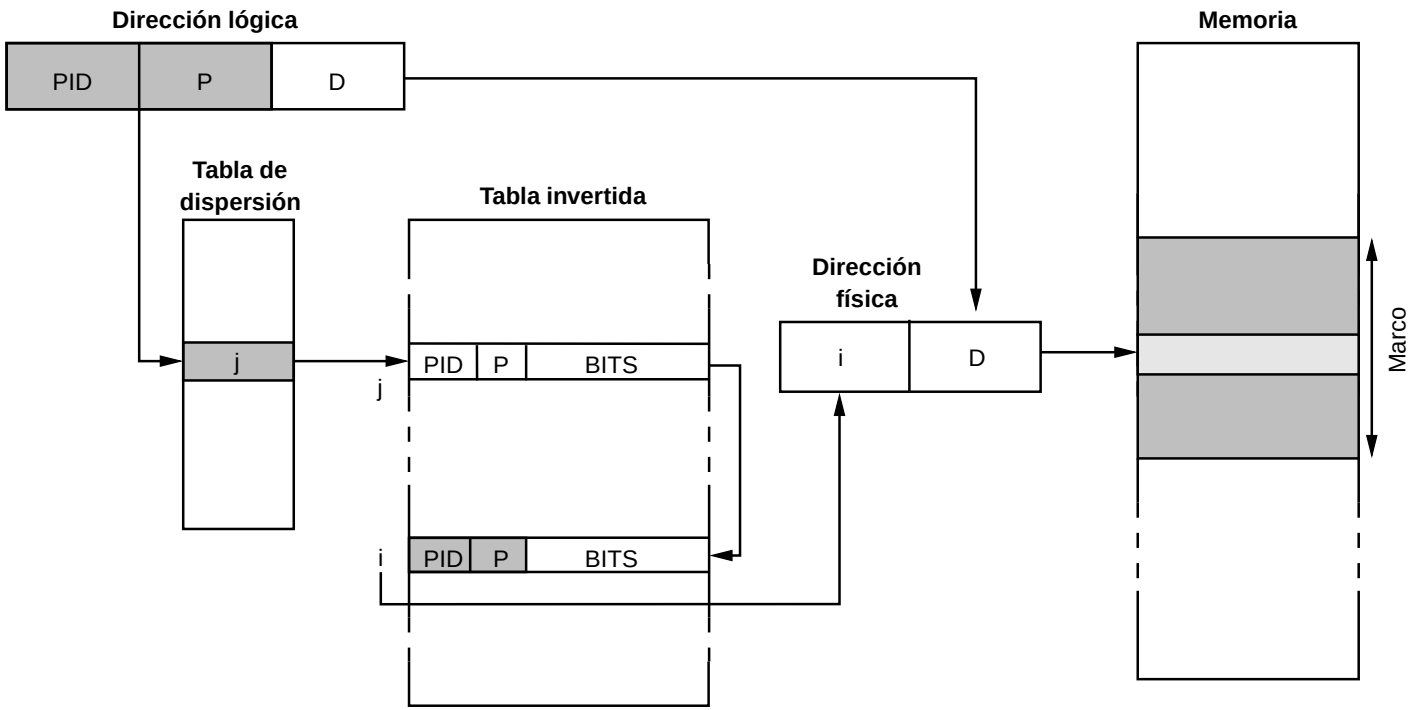

Figura 2.10: Tabla de páginas invertida

2.10. Funciones del gestor de memoria virtual

El gestor de memoria virtual está compuesto de diversos elementos software, que aparecen en la tabla 2.1. Todos tienen como objetivo obtener un buen rendimiento del sistema para lo cual intentan minimizar el porcentaje de fallos de página. Ya que como se ha comentado anteriormente, éstos suponen una considerable sobrecarga de trabajo.

Política de lectura	Demanda
	Prepaginación
Política de colocación	
Política de sustitución	Algoritmos básicos
	Buffering de páginas
Gestión del conjunto residente	Tamaño del conjunto residente
	Alcance del reemplazamiento
Política de limpieza	Demanda
	Limpieza previa
Control de la carga	

Cuadro 2.1: Funciones del gestor de memoria virtual

2.10.1. Política de lectura

También se denomina **política de carga**. Es la encargada de determinar qué elemento traer y cuándo se incorpora desde el almacenamiento secundario a la memoria principal, es decir, cuándo se debe cargar una página en memoria principal. Existen dos alternativas, la **paginación por demanda** y la **prepaginación** o **paginación previa**.

2.10.1.1. Paginación por demanda

Las páginas son traídas a memoria principal cuando se produce el fallo de página. Con esta política, cuando comienza la ejecución de un proceso no dispone de ninguna página en memoria, por lo que producirá un elevado número de fallos de página. Según el principio de localidad, a medida que se vayan cargando páginas en memoria el número de fallos disminuirá. Pasado un cierto tiempo, el proceso empezará a hacer referencia a direcciones que no se encuentran en las páginas cargadas, por lo que se producirá de nuevo un incremento en el número de fallos de página. Esto irá ocurriendo sucesivas veces hasta la terminación del proceso.

2.10.1.2. Prepaginación

Con esta técnica se llevan a memoria otras páginas, además de las demandadas por los fallos de página. De este modo, el sistema se adelanta a las necesidades

futuras, es decir, intenta tener el elemento antes de que sea necesario. La eficacia de esta estrategia depende de si las páginas que se traen van a ser referenciadas en un futuro próximo.

El fundamento de esta estrategia se encuentra en las características de los dispositivos de almacenamiento secundario. Dado que traer una página a memoria principal requiere de un tiempo de acceso al dispositivo más un tiempo de transferencia, es más eficiente traer varias páginas contiguas en el almacenamiento secundario, cuando se produce un fallo de página, en lugar de traerlas una a una. Sin embargo, la utilidad de la prepaginación no ha sido demostrada.

2.10.2. Política de colocación

Determina dónde se va a situar una porción de un proceso en memoria principal. En los sistemas de paginación pura o segmentación paginada, la ubicación es inmediata y no presenta ninguna relevancia esta política, pues cualquier marco libre es adecuado para alojar una página.

Sin embargo, en el caso de un sistema de segmentación, la política de ubicación sí cobra importancia en el diseño del gestor de memoria virtual. En este caso se necesitará un algoritmo de colocación de los segmentos, tales como el mejor ajuste, primer ajuste, etc., vistos en el tema 7 de la asignatura Sistemas Operativos I.

2.10.3. Política de sustitución

Cuando un proceso produce un fallo de página no puede continuar hasta que no se incorpore a memoria principal la página ausente. Puede ocurrir que no existan marcos libres de memoria para asignárselo a la página solicitada. En este caso, habrá que sacar una de las que están en memoria para atender la demanda de la página que ocasionó el fallo. Para ello, tenemos que elegir una página víctima mediante un algoritmo de sustitución, para que su lugar lo ocupe la página solicitada.

A la hora de abordar la sustitución de páginas hay que tener en cuenta varios aspectos:

- El número de marcos asignados a cada proceso activo.
- El conjunto de páginas candidatas para la sustitución, sólo las del proceso que ocasiona el fallo o todas las que residen en memoria.
- La elección de la página a ser sustituida.

El estudio conjunto de los dos primeros aspectos se conoce como **gestión del conjunto residente**, mientras que el último es la **política de sustitución**.

2.10.3.1. Tasa de fallos de página

Existen numerosos algoritmos de sustitución, para compararlos utilizaremos la **tasa media de fallos de páginas**, ya que es ésta la que influye directamente en el rendimiento del sistema. Para mejorarlo, los algoritmos deberían seleccionar aquella página que tenga la menor probabilidad de volver a ser referenciada próximamente. El principio de localidad establece una relación entre el comportamiento actual y en un futuro cercano del proceso. De ahí que los distintos algoritmos de sustitución intenten predecir el comportamiento futuro en función del pasado reciente.

La tasa de fallos de página se ve influida por varios factores. Por un lado, va a variar según el número de marcos de memoria de los que dispone el proceso. Estudios experimentales han comprobado que el comportamiento de los procesos es el que se muestra en la figura 2.11. En ella se observa que cuando el número de marcos del que dispone el proceso es muy pequeño la tasa de fallos es muy alta; a medida que aumenta la memoria asignada al proceso, la tasa disminuye. Sin embargo esa disminución no es lineal, a partir de un cierto valor la asignación de memoria adicional no repercute de forma significativa en la tasa de fallos de página. Obviamente, si a un proceso se le asigna un número de marcos suficiente para alojar todas sus páginas su tasa de fallos será cero.

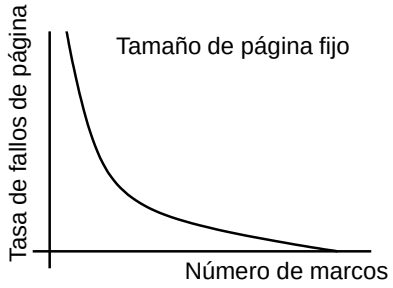

Figura 2.11: Relación entre el n.º de marcos y la tasa de fallos de página

Otro factor que influye sobre la tasa de fallos de página es el tamaño de ésta. La figura 2.12 muestra cómo varía la tasa de fallos frente al tamaño de la página para una cantidad de memoria fija. El comportamiento puede explicarse teniendo en cuenta el principio de localidad. Cuando el tamaño de la página es pequeño se observa que la tasa de fallos es baja, debido a que el proceso dispone de un elevado número de páginas en memoria y, por tanto, es probable que toda su localidad esté residente. Cuando aumenta el tamaño de la página se observa que también lo hace la tasa de fallos. Esto es debido a que el número de páginas de las que dispone el proceso en memoria es más reducido y a que éstas contienen un mayor número de direcciones que no pertenecen a la localidad actual. Obviamente, cuando el tamaño de la página coincide con el del proceso la tasa de fallos de página es cero (punto P).

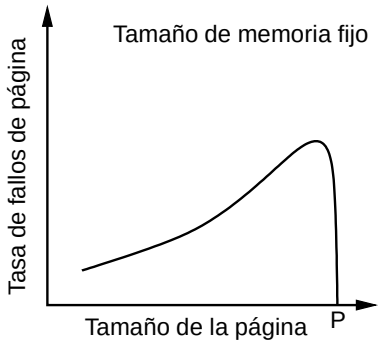

Figura 2.12: Relación entre el tamaño de la página y la tasa de fallos de página

2.10.3.2. Algoritmos de sustitución

Se han propuesto diversos algoritmos de sustitución de páginas; aquí vamos a considerar algunos de ellos: óptimo, FIFO, LRU y 2 variantes del algoritmo del reloj. Es importante tener en cuenta que cuanto más compleja sea la política de sustitución, mayores serán sus requisitos hardware para implementarla, y mayor sobrecarga producirá.

Con objeto de compararlos consideraremos un proceso que dispone de cuatro marcos para cargar sus páginas y que realiza la siguiente cadena de referencias a páginas:

3 1 4 2 3 1 5 3 1 4

Algoritmo óptimo

Selecciona para su sustitución la página que tardará más tiempo en volver a ser referenciada, es decir, aquella que no se usará durante un período de tiempo más largo. Se debe a Belady, y es la política que garantiza la tasa de fallo más baja para un número fijo de marcos.

Un ejemplo de utilización de este algoritmo aparece en la tabla 2.2. En ella podemos ver la página que referencia el proceso en cada instante, así como el contenido de los marcos asignados, resaltándose en negrita la última página que entra. Se indican también, las páginas que entran y salen cuando se produce un fallo de página. En este ejemplo se producen 5 fallos de página.

Este algoritmo resulta imposible de implementar porque requiere que el sistema operativo tenga un conocimiento exacto de las páginas que se van a referenciar en un futuro. Sin embargo, posee un interés teórico, al servir como estándar para comparar el resto de algoritmos.

Ref.	3	1	4	2	3	1	5	3	1	4
Marco 0	**3**	3	3	3	3	3	3	3	3	3
Marco 1	-	**1**	1	1	1	1	1	1	1	1
Marco 2	-	-	**4**	4	4	4	4	4	4	4
Marco 3	-	-	-	**2**	2	2	**5**	5	5	5
Fallo	✔	✔	✔	✔			✔			
Entra	3	1	4	2			5			
Sale	-	-	-	-			2			

Cuadro 2.2: Comportamiento del algoritmo óptimo

Algoritmo FIFO

Como su nombre indica saca de la memoria aquella página que llegó en primer lugar, es decir, la página más antigua en memoria. Un ejemplo de funcionamiento del algoritmo aparece en la tabla 2.3. En ella el símbolo ➔ denota la página más antigua, que es la candidata a salir. En este caso se producen 8 fallos de página.

Ref.	3	1	4	2	3	1	5	3	1	4
Marco 0	➔**3**	➔3	➔3	➔3	➔3	➔3	**5**	5	5	➔5
Marco 1	-	**1**	1	1	1	1	➔1	**3**	3	3
Marco 2	-	-	**4**	4	4	4	4	➔4	**1**	1
Marco 3	-	-	-	**2**	2	2	2	2	➔2	**4**
Fallo	✔	✔	✔	✔			✔	✔	✔	✔
Entra	3	1	4	2			5	3	1	4
Sale	-	-	-	-			3	1	4	2

Cuadro 2.3: Comportamiento del algoritmo FIFO

Su ventaja principal es su facilidad de implementación. Sólo necesita una lista de M elementos (tantos como marcos de memoria), con un puntero que apunte a la primera página (la más antigua) y otro a la última (la más reciente). Cuando se produce un fallo de página, se elimina la primera y se añade la nueva al final de la lista.

Un modo alternativo de implementarlo es con una cola circular y un solo puntero que apunte al principio de ésta. Cuando se produce un fallo de página, se elimina de memoria la página apuntada por el puntero, y se sitúa en su lugar el nuevo elemento. En ese momento, el puntero se incrementa, apuntando al elemento más antiguo.

El principio en el que se basa este algoritmo es que una página que lleve mucho tiempo en memoria puede que ya no se necesite. Sin embargo, este razonamiento es incorrecto, porque hay páginas que son usadas durante todo el tiempo de ejecución del proceso. De este modo pueden salir de memoria páginas que todavía se necesitan.

Como se vió en la figura 2.11 al aumentar el número de marcos asignados a un proceso debería disminuir su tasa de fallos de página. Sin embargo, algunos algoritmos presentan lo que se conoce como **anomalía de Belady**, por la cual la

Ref.	2	3	4	5	2	3	6	2	3	4	5	6
Marco 0	→**2**	→2	→2	**5**	5	→5	**6**	6	6	6	→6	→6
Marco 1	-	**3**	3	→3	**2**	2	→2	→2	→2	**4**	4	4
Marco 2	-	-	**4**	4	→4	**3**	3	3	3	→3	**5**	5
Fallo	✔	✔	✔	✔	✔	✔	✔			✔	✔	
Entra	2	3	4	5	2	3	6			4	5	
Sale	-	-	-	2	3	4	5			2	3	

(a) Comportamiento del algoritmo FIFO con 3 marcos

Ref.	2	3	4	5	2	3	6	2	3	4	5	6
Marco 0	→**2**	→2	→2	→2	→2	→2	**6**	6	6	→6	**5**	5
Marco 1	-	**3**	3	3	3	3	→3	**2**	2	2	→2	**6**
Marco 2	-	-	**4**	4	4	4	4	→4	**3**	3	3	→3
Marco 3	-	-	-	**5**	5	5	5	5	→5	**4**	4	4
Fallo	✔	✔	✔	✔			✔	✔	✔	✔	✔	✔
Entra	2	3	4	5			6	2	3	4	5	6
Sale	-	-	-	-			2	3	4	5	6	2

(b) Comportamiento del algoritmo FIFO con 4 marcos

Cuadro 2.4: Anomalía de Belady

tasa de fallos de página puede aumentar al hacerlo el número de marcos asignados. El algoritmo FIFO presenta esta anomalía. En la tabla 2.4(a) podemos observar que cuando el proceso dispone de 3 marcos se producen 9 fallos de páginas, mientras que cuando se le asignan 4 marcos (tabla 2.4(b)) el número de fallos sube a 10.

Algoritmo LRU

El algoritmo LRU[4] reemplaza la página que hace más tiempo que no se utiliza, es decir, aquella que no ha sido referenciada desde hace más tiempo. Se basa en el principio de localidad, ya que la página que hace más tiempo que no se utiliza es la que tiene menor probabilidad de volver a ser referenciada. En la práctica el comportamiento de este algoritmo se aproxima mucho al óptimo.

Forma parte de los denominados **algoritmos de pila**, que se caracterizan porque no presentan la anomalía de Belady. Para estos algoritmos se cumple que el conjunto de páginas en memoria para N marcos es siempre un subconjunto de las páginas que estarían en memoria con $N+1$ marcos. Para LRU con N marcos, el conjunto de páginas en memoria serían las N páginas referenciadas más recientemente. Si el número de marcos es $N+1$, tendríamos en memoria la últimas $N+1$ páginas referenciadas.

Un ejemplo se aprecia en la tabla 2.5, en la que se muestra una pila en la que se mantienen en orden las páginas referenciadas que residen en memoria, situándose en la cima la página referenciada más recientemente y en la base la página que hace

[4]Siglas correspondientes a su denominación inglesa *Less Recently Used*.

Ref.	3	1	4	2	3	1	5	3	1	4
Marco 0	**3**	3	3	3	3	3	3	3	3	3
Marco 1	-	**1**	1	1	1	1	1	1	1	1
Marco 2	-	-	**4**	4	4	4	**5**	5	5	5
Marco 3	-	-	-	**2**	2	2	2	2	2	**4**
Fallo	✔	✔	✔	✔			✔			✔
Entra	3	1	4	2			5			4
Sale	-	-	-	-			4			2
Pila	3	1	4	2	3	1	5	3	1	4
	-	3	1	4	2	3	1	5	3	1
	-	-	3	1	4	2	3	1	5	3
	-	-	-	3	1	4	2	2	2	5

Cuadro 2.5: Comportamiento del algoritmo LRU

más tiempo que no se utiliza. Cuando se hace referencia a una página que reside en memoria, se saca de la pila y se coloca en la cima de ésta. Cuando se produce un fallo de página, ésta se sitúa en la cima y se saca la que está en la base. El número de fallos de página que se produce es 6. Este algoritmo da resultados mucho mejores que FIFO y parecidos al óptimo.

El problema que presenta este algoritmo es la dificultad de implementación de la pila. La mejor forma de implementarla es utilizar una lista doblemente enlazada con punteros a la cima y a la base. Así, en el peor de los casos habrá que modificar seis punteros para sacar una página y colocarla en la cima de la pila. Las actualizaciones son costosas, pero no es necesario buscar la página a reemplazar, puesto que el puntero a la base de la pila apunta a ella.

Este algoritmo se comporta como un FIFO en el caso de que cada página sea referenciada sólo una vez, pues no van a alterar sus posiciones dentro de la pila.

Algoritmo del reloj

Dada la sobrecarga que produce el mantenimiento de la pila en el algoritmo LRU, pese a sus buenos resultados, y el bajo rendimiento del algoritmo FIFO, se han desarrollado nuevos algoritmos. Éstos intentan obtener buenos rendimientos con poca sobrecarga.

Algunos de estos algoritmos hacen uso del soporte hardware para facilitar su implementación y producir menos sobrecarga. Los más conocidos son el **algoritmo del reloj** o **de la segunda oportunidad** y algunas de sus variantes.

El algoritmo del reloj hace uso de un bit que se asocia a cada marco y que se denomina bit de referencia o de uso. Éste se utiliza para saber si una página ha sido referenciada recientemente. Cuando la página se carga en memoria, se pone a uno. Si posteriormente se hace referencia a ella, también se pone a uno (si no lo está ya). A la hora de elegir una página, se seleccionará una con el bit a cero, con objeto de deshacerse de las páginas de uso menos frecuente.

Algoritmo 2.1 Algoritmo del reloj

```
mientras no seleccione página a ser reemplazada
hacer
  si el puntero está en una página con bit de referencia a 0
  entonces
    La página es seleccionada para ser reemplazada
    Avanzar puntero
  si no
    Poner el bit de referencia a 0
    Avanzar puntero
  fin si
fin mientras
```

El funcionamiento de este algoritmo es similar al FIFO, pero le da a cada página una segunda oportunidad. Considera el conjunto de páginas candidatas a la sustitución como una lista circular y cuando hay que sustituir una, recorre la lista hasta encontrar una página con su bit de referencia a cero. Cada vez que pasa por una con el bit a uno, lo pone a cero. En el caso de que todas las páginas tengan su bit a uno, se dará una vuelta completa a toda la lista poniendo los bits a cero, parándose en la posición original y escogiendo esa página para su sustitución. En este caso, se comporta como FIFO. El algoritmo 2.1 muestra este comportamiento.

Un ejemplo se ve en la tabla 2.6, donde aparece el bit de referencia como superíndice de la página. En este caso se producen 8 fallos de página.

Ref.	3	1	4	2	3	1	5	3	1	4
Marco 0	→**3**^{1}	→3^1	→3^1	→3^1	→3^1	→3^1	**5**^{1}	5^1	5^1	→5^1
Marco 1	-	**1**^{1}	1^1	1^1	1^1	1^1	→1^0	**3**^{1}	3^1	3^1
Marco 2	-	-	**4**^{1}	4^1	4^1	4^1	4^0	→4^0	**1**^{1}	1^1
Marco 3	-	-	-	**2**^{1}	2^1	2^1	2^0	2^0	→2^0	**4**^{1}
Fallo	✔	✔	✔	✔			✔	✔	✔	✔
Entra	3	1	4	2			5	3	1	4
Sale	-	-	-	-			3	1	4	2

Cuadro 2.6: Comportamiento del algoritmo del reloj

Algoritmo del reloj mejorado

Si se aumenta el número de bits empleados se obtienen distintas variantes del algoritmo del reloj que pueden obtener mejores resultados. Una posibilidad consiste en hacer uso del bit de modificación que se asocia a cada página en memoria en los sistemas de paginación. El empleo de éste junto con el de referencia permite clasificar las páginas en cuatro tipos:

Página (0,0) Es una página que no ha sido referenciada recientemente y que su contenido no se ha modificado desde que se cargó en memoria.

Página (0,1) No ha sido referenciada recientemente pero su contenido se ha modificado durante su estancia en memoria.

Página (1,0) Se ha referenciado recientemente pero no ha sido modificada.

Página (1,1) Aquella que ha sido referenciada y modificada.

El algoritmo del reloj mejorado, que vamos a describir a continuación, hace uso de estos dos bits de la siguiente forma:

Paso 1 En primer lugar busca páginas de tipo (0,0) y si encuentra alguna la sustituye.

Paso 2 Si no ha encontrado ninguna, recorre de nuevo la lista circular buscando páginas (0,1), seleccionando la primera que encuentre. Durante esta búsqueda, modifica el bit de referencia de aquellas páginas que lo tienen a uno.

Paso 3 Si todavía no ha encontrado ninguna, vuelve al paso 1, ya que todas las páginas habrán cambiado de categoría.

El objetivo que persigue este algoritmo es doble, seleccionar aquella página que tenga menos posibilidades de ser referenciada de nuevo y, además, que su sustitución no conlleve una operación de escritura en el área de intercambio. Si no encuentra ninguna, volvería a rastrear la lista circular descartando el segundo objetivo. Esto implica que se pueden dar varias vueltas a la lista hasta encontrar la página a sustituir.

Un ejemplo se aprecia en la tabla 2.7, en la que aparece como superíndice el bit de referencia y como subíndice el bit de modificación. El número de fallos de página que se produce es 6, siendo menor que en la versión original del algoritmo.

Ref.	3^w	1^w	4	2	3	1	5	3	1	4
Marco 0	→$\mathbf{3}_1^1$	→3_1^1	→3_1^1	→3_1^1	→3_1^1	→3_1^1	3_1^0	3_1^1	3_1^1	→3_1^1
Marco 1	-	$\mathbf{1}_1^1$	1_1^1	1_1^1	1_1^1	1_1^1	1_1^0	1_1^0	1_1^1	1_1^1
Marco 2	-	-	$\mathbf{4}_0^1$	4_0^1	4_0^1	4_0^1	$\mathbf{5}_0^1$	5_0^1	5_0^1	5_0^1
Marco 3	-	-	-	$\mathbf{2}_0^1$	2_0^1	2_0^1	→2_0^0	→2_0^0	→2_0^0	$\mathbf{4}_0^1$
Fallo	✔	✔	✔	✔			✔			✔
Entra	3	1	4	2			5			4
Sale	-	-	-	-			4			2

Cuadro 2.7: Comportamiento del reloj mejorado

Otros algoritmos

Además de los vistos anteriormente, existen otros como el LFU[5] y el MFU[6], que consisten en seleccionar la página que menos y más veces se ha usado, respectivamente. Estos algoritmos emplean un contador por cada página, que indica el número de veces que ha sido referenciada. En cada caso, se escogerá la que tenga el menor y el mayor valor, respectivamente. La política LFU se basa en que si la página presenta un valor grande significa que se usa mucho por lo que no debe reemplazarse. Por otro lado, el algoritmo MFU supone que una página con un valor pequeño significa que tiene que ser referenciada en un futuro próximo, por lo que no debe descargarse.

2.10.3.3. Almacenamiento intermedio de páginas

Los algoritmos de sustitución introducen una sobrecarga grande intentando escoger la página más adecuada. Otro enfoque posible sería la utilización de algoritmos simples, que provoquen menos sobrecarga, haciendo que una mala elección de la página a sustituir no sea tan decisiva para el sistema. Esto es lo que se intenta con el **almacenamiento intermedio de páginas**.

Éste establece una zona de memoria que se utiliza para mantener temporalmente las páginas seleccionadas por el algoritmo de sustitución. De este modo, cuando se selecciona una página, ésta no sale realmente de memoria, sino que se mantiene durante un tiempo en este almacenamiento intermedio y se modifica su entrada en la tabla de páginas. Así, si la página vuelve a ser referenciada no tiene que ser traída a memoria de nuevo, puesto que no ha salido de ella.

El sistema Mach nos puede servir como ejemplo. Éste utiliza el algoritmo FIFO y para mejorar el rendimiento, las páginas que van a ser reemplazadas, no se pierden, sino que se mantienen en una lista de páginas libres.

La figura 2.13 muestra el funcionamiento de este esquema en un sistema con seis marcos, donde el almacenamiento intermedio está formado por dos marcos. En los pasos (b) y (c) las páginas sustituidas pasan al almacenamiento intermedio. En el paso (d) se vuelve a referenciar una página sustituida, pero que todavía se encuentra en el almacenamiento intermedio. De este modo es devuelta al conjunto residente con un coste bajo, puesto que no hay que acceder al disco. En el paso (e) hay que introducir una nueva página, por lo que sale de la memoria una que se encontraba en el almacenamiento intermedio, y la página seleccionada por el algoritmo de sustitución pasa a éste.

El sistema LINUX también hace uso de esta técnica; su estudio lo haremos en el apartado 2.11.7. También la emplea el sistema VAX/VMS, pero empleando dos listas, una para las páginas que no han sido modificadas y otra para las que sí lo han sido. El uso de estas listas permite escribir las páginas modificadas en bloques y sólo cuando sea necesario. Esto reduce significativamente el número de operaciones

[5]Siglas correspondientes a su denominación inglesa *Less Frequently Used*.
[6]Siglas correspondientes a su denominación inglesa *Most Frequently Used*.

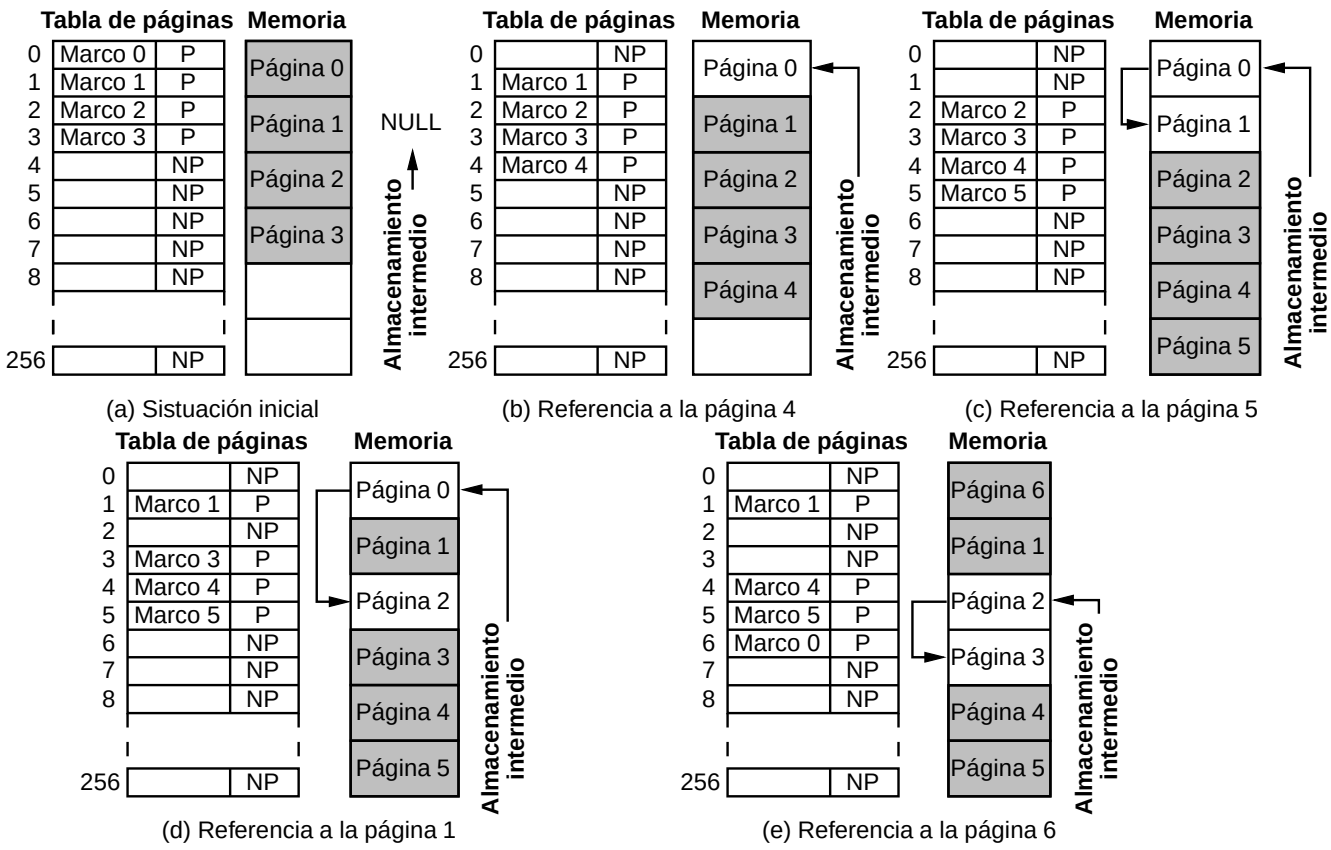

Figura 2.13: Funcionamiento del almacenamiento intermedio de páginas

de E/S y, por tanto, la cantidad de tiempo de acceso al disco.

2.10.4. Gestión del conjunto residente

En el apartado 2.10.3 se comentó que en la sustitución de páginas se encuentran involucrados varios aspectos interrelacionados, el tamaño del conjunto residente, el alcance de la sustitución y la estrategia de sustitución. Esta sección la vamos a dedicar al estudio conjunto de los dos primeros aspectos, conocido como gestión del conjunto residente.

2.10.4.1. Tamaño del conjunto residente

Uno de los aspectos de diseño del gestor de memoria virtual es decidir el número de marcos que se asignará a cada proceso, es decir, el tamaño del conjunto residente. Existen dos alternativas, la **asignación fija** y la **asignación variable**.

Con la primera, el sistema proporciona a cada proceso un número fijo de marcos para su ejecución. Este número se establece en el momento de la creación del proceso, según el tipo, prioridad, tamaño, etc. Se ha de tener en cuenta que cuanto menor sea la cantidad de memoria que se le asigne al proceso, mayor será el grado de multiprogramación del sistema. Sin embargo, según el principio de localidad, si el tamaño del conjunto residente de un proceso es demasiado pequeño su tasa de fallos de página puede aumentar considerablemente, pudiendo provocar una situación conocida como **hiperpaginación**. Ésta se caracteriza por una caída del rendimiento del sistema. Este concepto se estudiará más detenidamente en el apartado 2.10.6. Por otro lado, estudios del comportamiento de los procesos han permitido comprobar que la asignación de más memoria a un proceso por encima de un determinado valor, no tiene un efecto significativo sobre su tasa de fallos de página (véase la figura 2.11).

La estrategia de asignación variable establece conjuntos residentes de tamaño variable, que se adaptan a las necesidades de ejecución de los procesos. De este modo, un proceso que produce muchos fallos de página verá incrementado el número de marcos que tiene asignados. Mientras que aquellos que producen pocos fallos de página podrán reducir el tamaño de su conjunto residente sin incidir de forma negativa en su rendimiento.

Según las afirmaciones anteriores, la política de asignación variable parece ser la más flexible y adecuada. Sin embargo, requiere que el sistema operativo valore las necesidades de los procesos produciendo una sobrecarga en el sistema.

2.10.4.2. Alcance de la sustitución

Otro de los aspectos de diseño del gestor de memoria virtual es decidir cuál va a ser el conjunto de páginas candidatas a ser sustituidas cuando se produce un

fallo de página y no hay marcos libres. Esto es lo que se conoce como alcance de la sustitución. Éste puede ser **global** o **local**.

Una estrategia de alcance local sólo elige entre las páginas residentes del proceso que provoca el fallo de página, mientras que una política de alcance global puede elegir entre todas las que se encuentran en memoria, independientemente del proceso al que pertenezcan.

Una restricción aplicable a estas políticas es la existencia de páginas que no pueden ser sacadas de memoria ya que contienen una información especial. La mayor parte del núcleo del sistema operativo, así como estructuras de control claves para el funcionamiento del sistema, *buffers* de E/S, etc., son ejemplos de éstas. Estas páginas se encuentran en marcos bloqueados (ver apartado 2.6.1).

El alcance de la sustitución y el tamaño del conjunto residente son conceptos que están relacionados, dando lugar a diferentes esquemas. A continuación veremos las combinaciones posibles.

Asignación fija, alcance local

Cuando se emplea una política de asignación fija, cada vez que se necesita aplicar un algoritmo de sustitución, el conjunto de páginas candidatas se reduce a las del propio proceso. En este sistema es necesario establecer tanto el algoritmo de sustitución que se va a emplear como el tamaño del conjunto residente de cada proceso.

Asignación variable, alcance global

En estos sistemas los tamaños de los conjuntos residentes de los procesos se van adaptando a sus necesidades. Así, cuando un proceso produce un fallo de página, si existen marcos libres se le añade a su conjunto residente; si no los hay, se aplicará el algoritmo de sustitución a todas las páginas residentes en memoria que no estén bloqueadas. De este modo, un proceso que produzca muchos fallos de página aumentará el tamaño de su conjunto residente a costa de disminuir el de otros procesos.

Este esquema es uno de los más empleados en los sistemas actuales debido a su facilidad de implementación, ya que sólo necesita establecer el algoritmo de sustitución. Sin embargo, el inconveniente que plantea es que podría seleccionarse para su sustitución una página de un proceso al que no conviene reducir su conjunto residente. Esto puede provocar que el sistema pase a una situación de hiperpaginación.

Una forma de contrarrestar este problema sería la utilización del almacenamiento intermedio de páginas (ver apartado 2.10.3.3).

Asignación variable, alcance local

Esta estrategia también intenta adaptar el tamaño del conjunto residente a las necesidades del proceso, pero sin incurrir en los problemas de hiperpaginación que puede plantear el alcance global. Para ello, se comporta como una estrategia de asignación fija y alcance local a intervalos, es decir, inicialmente asigna un número

de marcos a los procesos y durante un cierto período de tiempo emplea alcance local. Pasado un tiempo ajusta el tamaño del conjunto residente a las necesidades del proceso, según haya sido su comportamiento en el período anterior. Así, si un proceso ha sufrido muchos fallos de página aumentará su conjunto residente y si ha sufrido muy pocos, éste podrá disminuir. Esta evaluación periódica hace que esta estrategia sea más compleja que la de alcance global, pero ofrece un mejor rendimiento.

En este sistema hay que determinar el tamaño del conjunto residente de cada proceso, los intervalos de cambio y los criterios para modificar los tamaños de los conjuntos residentes. A continuación se estudiarán dos estrategias de este tipo, el **modelo del conjunto de trabajo** y la **frecuencia de fallos de página**.

Modelo del conjunto de trabajo Esta estrategia, introducida por Denning, ha recibido mucha atención en la bibliografía. Se basa en el principio de localidad, según el cual los procesos, durante períodos cortos de tiempo, hacen referencia a un conjunto de direcciones agrupadas. Sin embargo, a lo largo de períodos de tiempo largos estos conjuntos de direcciones cambian. Es decir, los procesos, durante su ejecución, se mueven de una localidad a otra.

La filosofía subyacente en el modelo del conjunto de trabajo es que, en un momento dado, la cantidad de almacenamiento que requiere un proceso puede estimarse basándose en su comportamiento en el pasado reciente.

Se puede definir el conjunto de trabajo de un proceso como el conjunto de páginas a las que hace referencia activamente el proceso; según el principio de localidad antes citado, este conjunto varía a lo largo del tiempo.

Más formalmente podemos decir que el conjunto de trabajo (W) es función del tiempo y de un parámetro Δ que se denomina **ventana del conjunto de trabajo**; así, $W(t, \Delta)$ es el conjunto de páginas que ha referenciado el proceso en las últimas Δ unidades de tiempo. Puesto que Δ es una medida de tiempo, y estamos considerando sistemas de tiempo compartido, Δ debe medir el tiempo de ejecución del proceso en vez del tiempo real, por lo que se habla de tiempo virtual. Podemos decir que una unidad de tiempo virtual es el tiempo que tarda una instrucción en ejecutarse.

Pasamos a considerar ahora las dos variables de las que depende W. La variable Δ es una ventana de tiempo durante la cual se observa el comportamiento del proceso, por tanto, el tamaño del conjunto de trabajo es una función no decreciente de Δ. De forma matemática:

$$W(t, \Delta) \subseteq W(t, \Delta + 1)$$

En la figura 2.14 podemos ver un ejemplo del funcionamiento de esta estrategia para la ejecución de un proceso con distintos valores de Δ. En cada celda se muestra el conjunto de trabajo del proceso, estando ordenadas las páginas por el instante de su última referencia. Así, la página situada más a la derecha es la última referenciada.

De este modo, cuando se produce un fallo de página, indicado con el símbolo ✔, la página que se sustituye es la que se encuentra situada a la izquierda.

Secuencia de referencias a páginas	Tamaño de la ventana, Δ			
	2	**3**	**4**	**5**
12	12 ✔	12 ✔	12 ✔	12 ✔
27	12 27 ✔	12 27 ✔	12 27 ✔	12 27 ✔
33	27 33 ✔	12 27 33 ✔	12 27 33 ✔	12 27 33 ✔
16	33 16 ✔	27 33 16 ✔	12 27 33 16 ✔	12 27 33 16 ✔
12	16 12 ✔	33 16 12 ✔	27 33 16 12	27 33 16 12
①44	12 44 ✔	16 12 44 ✔	33 16 12 44 ✔	27 33 16 12 44 ✔
33	44 33 ✔	12 44 33 ✔	16 12 44 33	16 12 44 33
12	33 12 ✔	44 33 12	44 33 12	16 44 33 12
33	12 33	12 33	44 12 33	44 12 33
44	33 44 ✔	12 33 44	12 33 44	12 33 44
44	44	33 44	12 33 44	12 33 44
②44	44	44	33 44	12 33 44
12	44 12 ✔	44 12 ✔	44 12 ✔	33 44 12
27	12 27 ✔	44 12 27 ✔	44 12 27 ✔	44 12 27 ✔
12	27 12	27 12	44 27 12	44 27 12
33	12 33 ✔	27 12 33 ✔	27 12 33 ✔	44 27 12 33 ✔

Figura 2.14: Conjuntos de trabajo de un proceso para distintos Δ

Se puede observar que cuanto mayor es el tamaño de la ventana, mayor es también el conjunto de trabajo. Así, cuando se referencia la página 44 por primera vez (indicado como ①) alcanza el tamaño máximo de la ventana.

Hay que destacar que la elección del tamaño de la ventana influye en la tasa de fallos de página. A medida que aumenta Δ, el número de fallos de página disminuye.

La otra variable de la que depende el conjunto de trabajo es el tiempo. En la figura 2.14 se puede observar que para un valor de Δ fijo, el conjunto de trabajo va variando no sólo de contenido, sino también de tamaño. De este modo, cuando un proceso hace referencia a páginas que se encuentran dentro del conjunto su tamaño puede mantenerse o, incluso, disminuir. Así, en la última referencia a la página 44 (indicado como ②), el conjunto de trabajo sólo contiene a ésta para $\Delta = 2$ y $\Delta = 3$. De forma matemática esta relación se puede expresar:

$$1 \leq |W(t, \Delta)| \leq \text{mín}\{\Delta, N\}$$

donde N es el número de páginas del proceso.

Esta relación se puede observar en la figura 2.15 en la que se muestra cómo varía el tamaño del conjunto de trabajo a lo largo del tiempo para un valor fijo de Δ. En ella se alternan períodos donde el tamaño del conjunto de trabajo es relativamente estable con otros donde se producen cambios rápidos. Cuando un proceso empieza a ejecutarse, el tamaño del conjunto de trabajo aumenta rápidamente hasta alcanzar la localidad de referencias, donde se estabiliza. Pasado cierto tiempo, se produce un aumento brusco del tamaño del conjunto de trabajo seguido de un descenso rápido.

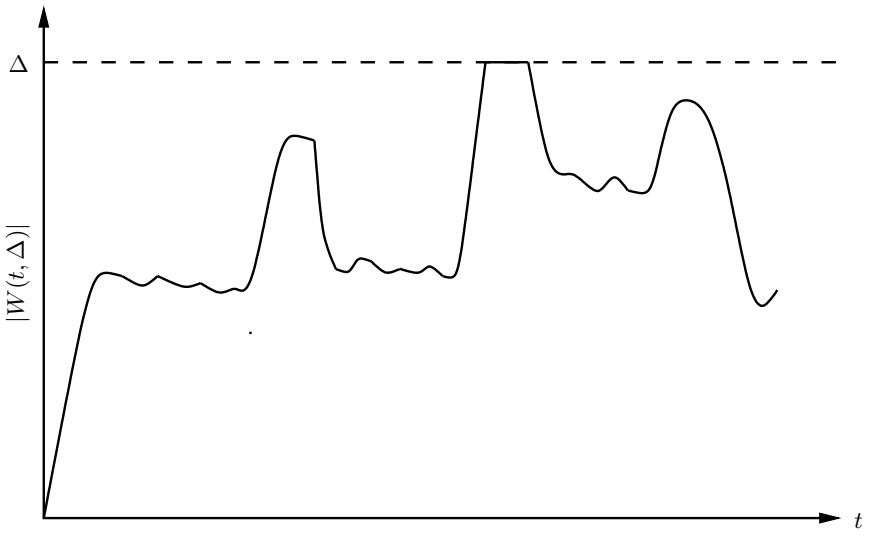

Figura 2.15: Evolución del conjunto de trabajo con el tiempo

Esto se debe a que el proceso cambia de localidad, por lo que su conjunto de trabajo irá admitiendo inicialmente páginas de la nueva localidad para posteriormente liberar las de la antigua.

La importancia de los conjuntos de trabajo es debida a que relacionan la administración de la memoria y de los procesos a través del **Principio del conjunto de trabajo**, que además se puede utilizar como una estrategia guía para la sustitución de páginas. Este principio dice:

1. Un proceso debe ejecutarse sólo si su conjunto de trabajo reside en memoria principal; es decir, si su conjunto residente incluye a su conjunto de trabajo.

2. Una página no debe ser retirada de memoria principal si es miembro del conjunto de trabajo de un proceso en ejecución.

Por tanto, con respecto al tamaño del conjunto residente, el número de páginas asignadas a cada proceso debería coincidir con el tamaño del conjunto de trabajo, es decir, el número de páginas distintas contenidas en éste. De esta forma evita la hiperpaginación, porque no permitiría ejecutar un proceso que no contara con su conjunto de trabajo en memoria. Con respecto a la sustitución de páginas, nos indica que una página debería ser sustituida sólo si no pertenece al conjunto de trabajo del proceso, ya que en caso contrario es probable que la tasa de fallos de página del proceso aumente, puesto que se está retirando una página con grandes posibilidades de ser referenciada en un futuro cercano.

Frecuencia de fallos de página Debido a los problemas que plantea la implementación de la estrategia del conjunto de trabajo, han aparecido otras que intentan

aproximarse a ella. Una de las más conocidas es la de **frecuencia de fallos de página** que utiliza la tasa de fallos de página para controlar el tamaño del conjunto residente. Este algoritmo utiliza el bit de referencia de la tabla de páginas. Se establece un umbral de tiempo F, que indica el tiempo[7] que debería de transcurrir entre dos fallos de página. Este umbral es el inverso de la tasa de fallos de página.

El razonamiento que sigue esta estrategia es el siguiente, si la tasa de fallos de página de un proceso está por debajo de un valor dado, se le podría disminuir el tamaño de su conjunto residente sin incidir notablemente en su tasa de fallos de página. Por el contrario, si la tasa de fallos de página de un proceso está por encima de un valor umbral, se debería aumentar el número de marcos asignados para disminuir su tasa de fallos de página. De este modo, la política de frecuencia de fallos de página garantiza que el conjunto residente crece cuando los fallos de página son frecuentes, y disminuye cuando la tasa de fallos de página baja.

La principal ventaja que presenta el algoritmo de frecuencia de fallos de página frente al modelo del conjunto de trabajo es que el conjunto residente sólo se ajusta cuando se produce un fallo de página en vez de en cada referencia.

La figura 2.16 muestra un ejemplo del comportamiento de este algoritmo con un umbral $F = 2$. Podemos ver como va aumentando inicialmente el tamaño del conjunto residente del proceso. Cuando llega a un periodo estable donde referencia las páginas cargadas no se producen fallos. Posteriormente, cuando se producen fallos, no sólo incorpora las nuevas páginas a memoria, sino que aquellas que no han sido referenciadas se descargan.

Ref. a páginas	Conjunto residente	Fallo de página	Entra	Sale
1	1^1	✔	1	-
4	$1^0\ 4^1$	✔	4	-
5	$1^0\ 4^0\ 5^1$	✔	5	-
3	$1^0\ 4^0\ 5^0\ 3^1$	✔	3	-
3	$1^0\ 4^0\ 5^0\ 3^1$			
3	$1^0\ 4^0\ 5^0\ 3^1$			
4	$1^0\ 4^1\ 5^0\ 3^1$			
2	$4^0\ 3^0\ 2^1$	✔	2	1 5
3	$4^0\ 3^1\ 2^1$			
5	$4^0\ 3^0\ 2^0\ 5^1$	✔	5	-
3	$4^0\ 3^1\ 2^0\ 5^1$			
5	$4^0\ 3^1\ 2^0\ 5^1$			
1	$3^0\ 5^0\ 1^1$	✔	1	4 2
4	$3^0\ 5^0\ 1^0\ 4^1$	✔	4	-

Figura 2.16: Comportamiento de la frecuencia de fallos de página

Esta estrategia se puede refinar mediante el uso de dos umbrales (figura 2.17). Si el intervalo de tiempo entre dos fallos de página está por encima de un umbral

[7]Puesto que estamos considerando un sistema de tiempo compartido, éste sería un tiempo virtual, es decir, cada unidad de tiempo equivale a la ejecución de una instrucción.

superior, aumentará el tamaño del conjunto residente; por contra, si está por debajo del umbral inferior disminuirá su tamaño.

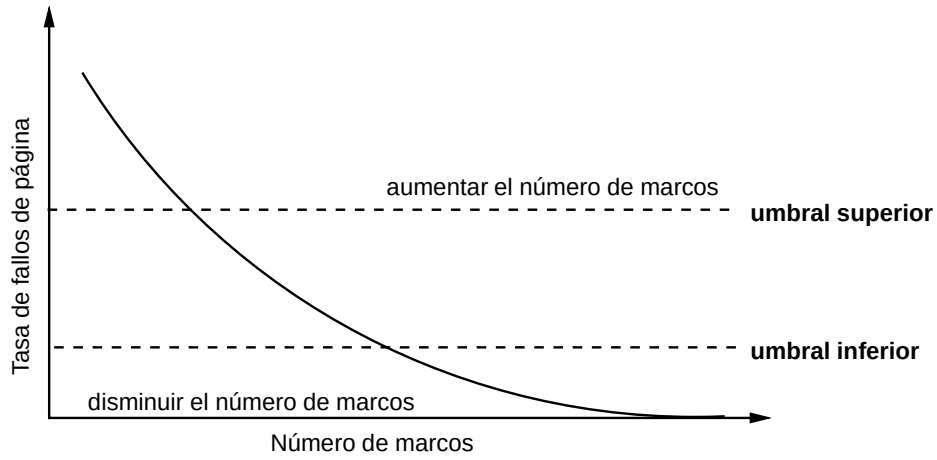

Figura 2.17: Frecuencia de fallos de página

2.10.5. Política de limpieza

La **política de limpieza** decide cuando debe ser escrita en el área de intercambio una página que ha sido modificada. Existen varias alternativas. Una es la denominada **limpieza previa** que escribe las páginas modificadas cada cierto tiempo, independientemente de que vayan a salir o no de la memoria. La ventaja de esta técnica es que permite realizar muchas operaciones de escritura aprovechando las características de los dispositivos de almacenamiento secundario. Sin embargo, el problema que plantea es que las páginas que se escriben permanecen en memoria principal pudiendo ser modificadas nuevamente, por lo que si tienen que ser reemplazadas deberán escribirse de nuevo.

La utilización del almacenamiento intermedio (ver apartado 2.10.3.3) puede mejorar esta estrategia. Si se limpian sólo aquellas páginas que son reemplazables, es decir, las que se encuentran en el almacenamiento intermedio, disminuiríamos el número de operaciones de escritura innecesarias.

Otra alternativa, conocida como **limpieza por demanda**, consiste en escribir una página modificada en el área de intercambio sólo cuando sea seleccionada por el algoritmo de sustitución. De este modo, la escritura de una página modificada va asociada y precede a la lectura de una nueva página. La ventaja de esta estrategia es que permite minimizar el número de operaciones de escritura en el área de intercambio. Pero presenta el inconveniente de que algunos fallos de página provoquen dos operaciones de E/S, la escritura de la página a sustituir y la lectura de la nueva.

2.10.6. Control de la carga y la hiperpaginación

Uno de los principales problemas de un sistema de gestión de memoria virtual es mantener un equilibrio entre el grado de multiprogramación y la tasa de fallos de página. Esta es una de las labores del gestor de memoria virtual y se conoce como **control de la carga**.

La figura 2.18 muestra cómo varía la utilización del procesador en función del grado de multiprogramación. Como se puede observar, a medida que éste crece a partir de un valor pequeño, la utilización del procesador también aumenta. Sin embargo, si aumenta por encima de un cierto valor, se observa una disminución drástica del rendimiento. Esto se debe a que al tener muchos procesos en memoria el tamaño de sus conjuntos residentes es demasiado pequeño lo que provoca una tasa muy alta de fallos de página. Esta situación se conoce como hiperpaginación.

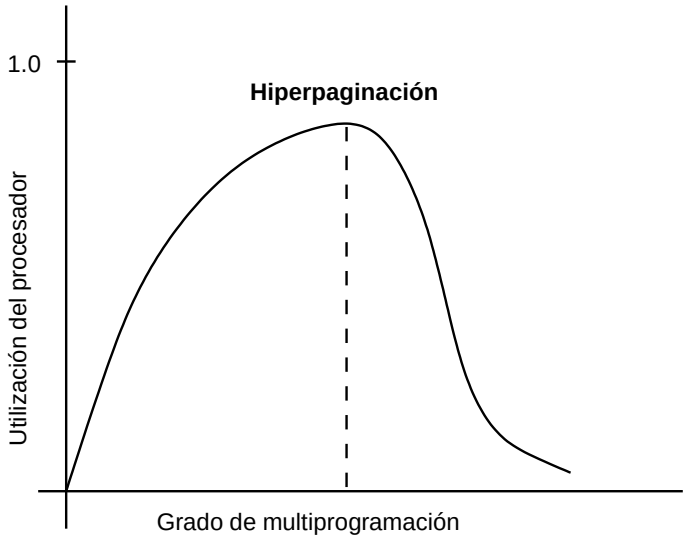

Figura 2.18: Hiperpaginación

Cuando se emplea una estrategia de gestión del conjunto residente de asignación variable y alcance local, el control de la carga viene determinado por ésta. Por ejemplo, si empleamos el modelo del conjunto de trabajo, sólo podrán ejecutarse aquellos procesos que tengan su conjunto de trabajo en memoria, de esta forma se determina automáticamente el grado de multiprogramación. De forma matemática, la demanda total de marcos en un instante dado, vendrá dada por:

$$D = \Sigma W(t, \Delta)$$

De este modo, si la demanda total es mayor que el número total de marcos disponibles en memoria se produciría hiperpaginación. Para evitarla, el sistema sólo crea un nuevo proceso si existen marcos suficientes para albergar su conjunto de trabajo. Dado que el tamaño de los conjuntos va variando a lo largo del tiempo, si

en un instante la suma de todos los conjuntos de trabajo es superior al número de marcos del sistema, habrá que suspender algún proceso, descargando sus páginas y asignándoselas a otros.

En los otros esquemas de gestión del conjunto residente hay que añadir un mecanismo de control de la carga. A la hora de implementarlo habrá que tener en cuenta dos aspectos:

1. Criterio para modificar el grado de multiprogramación.
2. Elección del proceso a intercambiar, si fuese necesario.

En cuanto al primer aspecto, existen varias posibilidades:

Criterio L=S ajusta el grado de multiprogramación de forma que se cumpla que el tiempo medio entre fallos de página (L) sea igual al tiempo medio requerido para procesar un fallo de página (S). Los estudios de rendimiento indican que éste es el punto en el cual el uso del procesador alcanza un máximo.

Criterio del 50 % intenta mantener un grado de multiprogramación, tal que el dispositivo de paginación esté ocupado el 50 % del tiempo. De nuevo, los estudios de rendimiento indican que éste es un punto de uso máximo del procesador.

Algoritmo del reloj adaptado es una modificación del algoritmo del reloj que consiste en supervisar la velocidad con la que el puntero completa una vuelta a la lista circular de marcos. Así, si se detecta que la velocidad es menor que un valor umbral, puede deberse a que se están generando pocos fallos de página o a que se encuentra fácilmente una página para su sustitución cuando se produce un fallo de página. En ambos casos, puede aumentar el nivel de multiprogramación sin degradar el rendimiento del sistema. Si la velocidad es mayor que un cierto umbral, puede estar motivado por una alta tasa de fallos de página o por la dificultad de encontrar una página para sustituir. Esta técnica se describe para un sistema de alcance global.

Respecto a la decisión de intercambiar un proceso para reducir el grado de multiprogramación, también existen diversos criterios:

- Prioridad del proceso.
- Instante de carga del proceso en memoria, si un proceso lleva poco tiempo cargado en memoria es poco probable que tenga en ella su conjunto de trabajo.
- Número de fallos de página. Un proceso con un número elevado de fallos de página es poco probable que tenga su conjunto de trabajo en memoria.
- Tamaño del proceso. Cuanto mayor sea el proceso que se intercambia mayor número de marcos libres se obtienen.

- Tamaño del conjunto residente. Si se escoge un proceso con un tamaño de conjunto residente pequeño el esfuerzo para cargar de nuevo su conjunto de trabajo será menor.

2.11. Gestión de memoria en LINUX

LINUX utiliza un esquema de memoria virtual bajo paginación por demanda. El tamaño de la página dependerá de la arquitectura, así en los sistemas Alpha AXP la página es de 8 KiB, mientras que en los Intel x86 es de 4 KiB.

LINUX presenta dos modos de direccionamiento, el virtual y el físico. El primero es el que utilizan los procesos de usuario y sigue un esquema de paginación. El segundo lo utiliza el núcleo y se caracteriza porque no emplea tablas de páginas ni traducción de direcciones de ningún tipo. No es conveniente mantener al núcleo del sistema operativo en memoria virtual ya que esto supondría una disminución del rendimiento del sistema.

Cada proceso dispone de un espacio de direcciones virtuales cuyo tamaño dependerá de la arquitectura. Éste se divide en dos segmentos, uno para el código y los datos del proceso de usuario que emplea direccionamiento virtual, y otro para el núcleo con direccionamiento físico. Así, en la familia Intel x86 este espacio es de 4 GiB (2^{32}), que se divide en un segmento de usuario de 3 GiB y un segmento de núcleo de 1 GiB.

2.11.1. Regiones de memoria virtual

El administrador de la memoria en LINUX mantiene dos visiones del espacio de direcciones virtuales de un proceso, como un conjunto de regiones separadas y como un conjunto de páginas.

La primera concuerda con la visión lógica que tiene el usuario del espacio de direcciones del proceso. Cada región consiste en un subconjunto de páginas contiguas del espacio de direcciones. El sistema mantiene información sobre el rango de direcciones virtuales que ocupa la región, sus modos de acceso y los ficheros asociados. El bloque de control del proceso apunta a las distintas estructuras que representan las regiones virtuales.

Cada región puede representar una parte de la imagen ejecutable, el código, los datos inicializados, los datos no inicializados, etc. Dependiendo del tipo de región, estará almacenada en un fichero o no. Por ejemplo, en el caso del código y los datos inicializados estarán en el fichero ejecutable; los datos no inicializados no tendrán un fichero asociado.

La segunda visión se corresponde con la estructura física del espacio de direcciones del proceso dividido en páginas.

2.11.2. Traducción de direcciones

LINUX define un modelo de memoria independiente de la arquitectura. En él una dirección virtual se traduce a una dirección física mediante un esquema de paginación de tres niveles.

En la figura 2.19 se puede observar este proceso de traducción. La dirección virtual se divide en 4 componentes, los tres primeros hacen referencia a las tablas de páginas de primer, segundo y tercer nivel, mientras que el último es el desplazamiento.

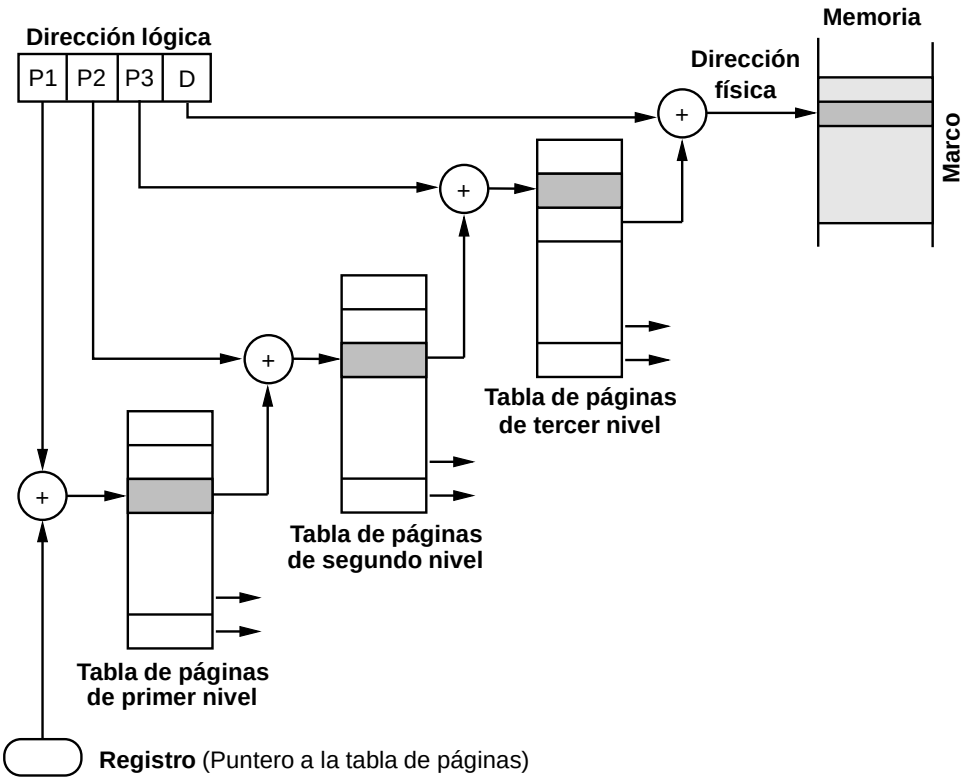

Figura 2.19: Traducción de direcciones en LINUX

Este esquema general se adapta a las características de las arquitecturas para las que se implementa. Así, en el caso de la familia Intel x86 emplea sólo dos niveles, mientras que el Alpha AXP emplea los tres.

2.11.3. Estructura de la tabla de páginas

Las entradas de la tabla de páginas son iguales independientemente del nivel. La estructura y tamaño de éstas dependen de la arquitectura. Para ilustrarlo se mostrarán dos ejemplos.

En el caso de la familia Intel x86 las entradas son de 32 bits y almacenan la siguiente información (figura 2.20):

Bit de protección (R/W) Controla el modo de acceso a la página.

Bit de tipo de página (U/S) Indica si se trata de una página de usuario o del núcleo.

Bit de presencia (P) Indica si la página está cargada en memoria o no.

Bits para soportar la paginación por demanda

Bit de modificación (M) Se activa cuando se modifica la página, desactivándose cuando se actualiza en memoria secundaria. Este bit no posee significado en la tabla de páginas de primer nivel.

Bit de referencia (R) Se activa cuando se hace referencia a la página.

Bits del sistema operativo (SO) Es un conjunto de bits que pueden ser empleados por el sistema operativo para implementar la política de sustitución de páginas.

Dirección del marco Dependiendo de si es de primer o segundo nivel corresponderá a una tabla de páginas o a una página del proceso, en el caso de que se encuentre en memoria principal.

31 - 12	11 10 9	8	7	6	5	4	3	2	1	0
DIRECCIÓN	SO			M	R			U/S	R/W	P

Figura 2.20: Entrada de la tabla de páginas en Intel x86

Cuando una página se descarga de memoria se almacena en los bits 1 a 31 su posición en memoria secundaria. El bit de presencia indica que la página no está en memoria principal.

Las entradas de las tablas de páginas en la arquitectura Alpha AXP tienen 64 bits y mantienen la siguiente información (figura 2.21):

Bit de presencia (P)

Bits de fallo de página

Bit de fallo en lectura (FR) Se activa cuando se produce un fallo de página al intentar leer una página que no reside en memoria.

Bit de fallo en escritura (FW) Se activa cuando se produce un fallo de página al intentar escribir en una página que no reside en memoria.

Bit de fallo en ejecución (FX) Se activa cuando se produce un fallo de página al intentar ejecutar una instrucción de una página que no reside en memoria.

Bits para el manejo de TLB (TLB) Conjunto de bits utilizados para la gestión de la TLB.

Bits de protección Se utilizan cuatro bits de protección que permiten indicar si una página puede ser leída en modo usuario (RU) o supervisor (RS), o bien, escrita en modo usuario (WU) o supervisor (WS).

Bits para soportar la paginación por demanda

Bit de modificación (M)

Bit de referencia (R)

Dirección del marco Dependiendo del nivel corresponderá a una tabla de páginas o a una página del proceso. Cuando la página no está en memoria, se almacena la dirección donde se encuentra en el área de intercambio.

11 10	9	8	7	6 5 4	3	2	1	0
	RU	RS		TLB	FX	FW	FR	P

63 - 32	31 - 23	22	21	20	19 - 14	13	12
DIRECCIÓN		R		M		WU	WS

Figura 2.21: Entrada de la tabla de páginas en Alpha AXP

2.11.4. Estructura de la tabla de marcos

La información más relevante que se suele almacenar en una entrada de la tabla de marcos es la siguiente:

Bit de bloqueo Indica si la página se encuentra implicada en una operación de E/S.

Bit de actualización Indica si el contenido de la página ha sido traído desde memoria secundaria con éxito.

Bit de error Indica si una operación de E/S ha finalizado con un error.

Bit de referencia Indica si la página ha sido referenciada. Lo utilizan los algoritmos de sustitución.

Bit de modificación Indica si la página ha sido modificada.

Contador de uso Muestra el número de procesos que comparten una página.

Dirección de la página en memoria secundaria

2.11.5. Gestión del espacio libre

La gestión de los marcos libres se lleva a cabo mediante una tabla. Cada elemento de ésta contiene información sobre bloques de páginas consecutivos en potencia de dos. Así, el primer elemento contiene información sobre los marcos individuales libres, el siguiente sobre bloques de dos marcos consecutivos, y así sucesivamente. Estos elementos contienen punteros a las entradas de la tabla de marcos correspondientes. Además, posee un mapa de bits por cada conjunto de bloques de marcos para seguir la pista de los marcos libres. De este modo, el bit n-ésimo valdrá 1 si el bloque n-ésimo de marcos está libre.

La figura 2.22 muestra la estructura de esta tabla. El elemento 0 tiene un marco libre (marco 0), y el elemento 2 tiene dos bloques libres (cada uno de cuatro marcos), el primero empezando en el marco 4 y el segundo en el marco 56.

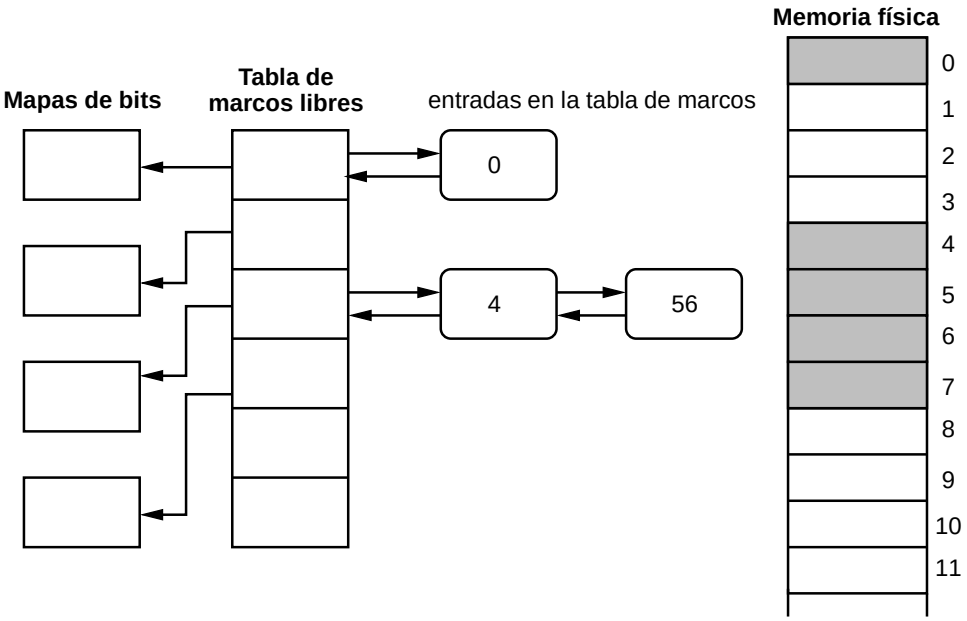

Figura 2.22: Estructura de la lista de marcos libres

2.11.5.1. Asignación de páginas

La gestión del espacio libre comentada anteriormente, permite que LINUX emplee el algoritmo compañero para la asignación de páginas. Las solicitudes pueden ser de marcos individuales o de conjuntos de marcos. Mediante el empleo de paginación por demanda, la mayoría de las solicitudes son de marcos individuales. Sin embargo, hay ocasiones en las que un proceso necesita más de un marco de forma simultánea, como por ejemplo, cuando se crea un proceso hijo. En este caso, inicialmente se comparten las páginas de código y de datos entre padre e hijo, pero cuando se realiza una modificación, hay que mantener imágenes distintas. También cuando un

proceso quiere crear un segmento de memoria compartido con otros, puede necesitar más de un marco. Igualmente, cuando un proceso en estado de suspendido vuelve a activarse hay que devolverle la memoria que tenía asignada en ese momento.

Cuando un proceso solicita un conjunto de marcos, se busca en la tabla de marcos libres un bloque libre del tamaño requerido. Si no existen bloques de páginas del tamaño requerido, se busca en los bloques del siguiente tamaño. Este proceso continúa hasta encontrar un bloque de marcos libre. Si el bloque asignado es mayor que el solicitado, los marcos sobrantes se introducen en la tabla de bloques libres de acuerdo a su nuevo tamaño.

Así, en la figura 2.22 si se solicita un bloque de dos páginas, el primer bloque de 4 marcos libres (empieza en el marco 4) se divide en dos bloques de dos marcos. El primero, que empieza en el marco 4, se asignará al proceso que lo solicitó, y el segundo bloque, que empieza en el marco 6, se introducirá en la lista de bloques de marcos libres de tamaño 2.

Una consecuencia de este algoritmo es que el proceso puede ocupar, siempre que sea posible, posiciones contiguas en la memoria.

2.11.5.2. Liberación de páginas

Cuando se libera una página, se comprueba que el bloque adyacente (bloque compañero) del mismo tamaño está libre. En ese caso, se combinan los dos formando un nuevo bloque libre. Cada vez que se unen dos bloques, el algoritmo intenta fusionarlo con bloques adyacentes para formar uno más grande. De este modo, se busca conseguir que los bloques de marcos libres sean tan grande como lo permita la memoria utilizada.

Así, en la figura 2.22 si el marco número 1 se libera, se combinará con el marco 0, creando un bloque de dos marcos libres que comienza en el marco 0.

2.11.6. Fallos de página

El sistema puede incurrir en dos tipos de fallos de página:

Fallo de validez La página referenciada no se encuentra en memoria.

Fallo de protección Se intenta acceder a una página en un modo no permitido o bien a una página que no pertenece al proceso.

La forma de indicar el tipo de fallo depende de la arquitectura. En el caso de la familia Intel se utiliza el registro CR2 para indicar la dirección causante del fallo de página. De este registro de 16 bits sólo se utilizan los tres primeros, como se muestra en la tabla 2.8.

Bit	Significado con 0	Significado con 1
0	Página no presente	Nivel de protección
1	Fallo debido a lectura	Fallo debido a escritura
2	Modo supervisor	Modo usuario

Cuadro 2.8: Códigos de error de un fallo de página

En la arquitectura Alpha AXP se emplean los bits de la tabla de páginas FR, FW y FX.

Cuando se produce un fallo de página, el sistema comprueba en primer lugar si la página pertenece al proceso y si se intenta acceder a ella de un modo adecuado. Esta comprobación se realiza buscando la región de memoria virtual asociada. Si la página y el modo son correctos se inicia la lectura de ésta.

La página que provoca el fallo puede encontrarse en el área de intercambio o bien, en el fichero ejecutable. En el primer caso, su dirección está almacenada en la entrada correspondiente de la tabla de páginas. Mientras que en el segundo, se encuentra en la estructura asociada a la región de memoria virtual correspondiente.

2.11.7. El demonio de paginación

El demonio de paginación es un hilo del núcleo que se ejecuta en segundo plano y que se activa una vez cada segundo. Su función es elegir los marcos que tienen que liberarse para poder alojar nuevas páginas, aplicando el algoritmo del reloj.

Cuando se activa comprueba el número de marcos libres que hay en el sistema, si está por debajo de un valor umbral empieza a liberar memoria. Dispone de tres umbrales que modifican su forma de actuación, es decir, el intervalo de tiempo para su activación.

Si una página seleccionada por el demonio de paginación contiene código del proceso, simplemente se descarta, debido a que puede volver a ser leída de nuevo del fichero ejecutable, ya que nunca se modifican. Por el contrario, si la página seleccionada ha sido modificada desde que se cargó en memoria principal, ésta se escribirá en el área de intercambio para liberar memoria. La entrada en la tabla de páginas se marcará como no válida, y se almacenará la dirección de la página en el área de intercambio.

Una vez que se han liberado los marcos, se termina el trabajo actualizando la tabla de marcos y las tablas de páginas. Las páginas que contienen al núcleo, así como las tablas de páginas de primer nivel, son páginas reservadas que nunca se descargarán de memoria.

2.11.8. Gestión del área de intercambio

LINUX puede utilizar como área de intercambio tanto una partición como un fichero. Si se utiliza una partición las operaciones con esta área son más rápidas. Sin embargo, la utilización de un fichero permite cambiar de forma fácil su tamaño, ya que no se necesita reparticionar el disco. Se pueden utilizar varias de estas áreas de forma simultánea.

Para llevar el control del espacio libre en las áreas de intercambio se utiliza un mapa de bits con un bit por página. En esta estructura no se indica el proceso propietario de la página, esta información se mantiene en su tabla de páginas.

La utilidad del área de intercambio es el almacenamiento de aquellas páginas que no pueden ser leídas del fichero ejecutable. Así, las páginas de código siempre se lee del propio programa, mientras que las de datos, pila, así como las tablas de páginas de segundo nivel pueden ser intercambiadas y descargadas a dicha área.

2.11.9. Llamadas al sistema

LINUX, al igual que su progenitor UNIX, tampoco especifica llamadas al sistema para la administración de la memoria. Esto se consideró como demasiado dependiente de la máquina como para lograr su estandarización. No obstante posee unas pocas llamadas al sistema relacionadas con la gestión de memoria, que se pueden ver en la tabla 2.9.

Llamada	Descripción
brk	Especificar el tamaño del segmento de datos.
malloc	Solicitar memoria.
free	Liberar memoria.
swapon	Activar área de intercambio.
swapoff	Desactivar área de intercambio.

Cuadro 2.9: Llamadas al sistema relacionadas con la gestión de memoria en LINUX

2.12. Resumen

Un sistema de memoria virtual es capaz de ejecutar procesos sin que la imagen de éstos se encuentre por completo en memoria principal. Estos sistemas se implementan basándose en un sistema paginado, segmentado, o una combinación de ambos. En este tema se ha realizado un estudio de los sistemas de memoria virtual paginados.

Una de las características de los sistemas de memoria virtual es que un proceso puede tener un tamaño muy superior a la memoria física disponible. Esto produce un

aumento considerable en el tamaño de las tablas de páginas, que se pueden reducir mediante el empleo de tablas de páginas multinivel o tablas de páginas invertidas.

En estos sistemas, cuando un proceso realiza una referencia que no se encuentra en memoria principal se produce un fallo de página. La existencia de éstos afecta de forma significativa al rendimiento del sistema, debido a las operaciones de E/S que se encuentran implicadas en su resolución. Por este motivo, uno de los objetivos que debe perseguir el diseño de un gestor de memoria virtual es la minimización de los fallos de página.

Un gestor de memoria virtual incorpora un conjunto de políticas para asegurar el buen funcionamiento del sistema, tales como la de lectura, colocación, sustitución, gestión del conjunto residente, política de limpieza y el control de la carga.

La política de lectura es la encargada de determinar qué página traer y cuándo se incorpora desde el almacenamiento secundario a la memoria principal. En este sentido, se estudian las técnicas de paginación por demanda y prepaginación. La política de colocación es la responsable de ubicar la página en memoria principal.

La política de sustitución decide qué página tiene que salir de memoria principal cuando ésta se encuentra llena y hay que cargar una nueva. Se estudian los algoritmos óptimo, LRU, FIFO, etc. Esta política está íntimamente relacionada con la gestión del conjunto residente, que se encarga de determinar su tamaño, fijo o variable, así como el alcance de la sustitución, local o global.

La política de limpieza realiza la escritura de las páginas que han sido modificadas en memoria principal. Se puede realizar en el momento en que la página sale de memoria principal, conocido como limpieza por demanda, o bien antes de salir, denominándose limpieza previa.

El control de la carga hace referencia al control que lleva el sistema para la evitación del problema de hiperpaginación. Esta situación se caracteriza porque produce una disminución del rendimiento del sistema.

Apéndice A

Prefijos para los múltiplos binarios

Este libro adopta las recomendaciones aprobadas en Diciembre de 1998 por el IEC (*International Electrotechnical Commission*) [?] respecto a los nombres y símbolos para los prefijos de los múltiplos binarios que se utilizan en los campos del procesamiento y transmisión de datos. Los prefijos son los que aparecen en la tabla A.1.

Factor	Nombre	Símbolo	Origen	Derivación
2^{10}	kibi	Ki	kilobinario:$(2^{10})^1$	kilo:$(10^3)^1$
2^{20}	mebi	Mi	megabinario:$(2^{10})^2$	mega:$(10^3)^2$
2^{30}	gibi	Gi	gigabinario:$(2^{10})^3$	giga:$(10^3)^3$
2^{40}	tebi	Ti	terabinario:$(2^{10})^4$	tera:$(10^3)^4$
2^{50}	pebi	Pi	petabinario:$(2^{10})^5$	peta:$(10^3)^5$
2^{60}	exbi	Ei	exabinario:$(2^{10})^6$	exa:$(10^3)^6$

Cuadro A.1: Prefijo para múltiplos binarios

Estos nuevos prefijos para los múltiplos binarios no son parte del Sistema Internacional de Unidades (SI), pero para facilitar su comprensión y memorización derivan de los prefijos del SI para las potencias positivas de diez. Como puede verse en la tabla A.1, el nombre de cada prefijo deriva del nombre de éste en el SI, se toman las dos primeras letras de éste, y se le añade "bi", que recuerda a la palabra binario. De forma similar, el símbolo nuevo deriva del correspondiente del sistema

internacional añadiéndole la letra "i", para recordar la palabra binario de nuevo.

A continuación se muestran algunos ejemplos y se comparan con los prefijos del SI:

1 kibibit	1 Kibit=2^{10} bit=1024 bit
1 kilobit	1 kbit=10^{3} bit=1000 bit
1 mebibyte	1 MiB=2^{20} B=1.048.576 B
1 megabyte	1 MB=10^{6} B=1.000.000 B
1 gibibyte	1 GiB=2^{30} B=1.073.741.824 B
1 gigabyte	1 GB=10^{9} B=1.000.000.000 B