Administración de Memoria

Una memoria debe ser en extremo rápida, de gran tamaño y muy económica. Actualmente, no existe ninguna tecnología que cumpla con todos esos objetivos. Por lo que se adoptó un sistema de **jerarquía de memoria**:

- **Registros**: Registros internos de la CPU
- **Cache**: Muy rápida, costosa y volátil
- **RAM**: Mediana velocidad, a precio mediano y volátil
- **Discos**: Lento, económico y no volátil
- **Discos removibles**: Almacenamiento removible, como los DVDs y las memorias USB

Administrador de memoria es la **parte del S.O que administra parte de la jerarquía de memoria**. Su trabajo es administrar la memoria con eficiencia, es decir:

- Llevar el registro de cuáles partes de la memoria están en uso
- Asignar memoria a los procesos cuando la necesiten
- Desasignarla cuando terminen

Sin Abstracción de Memoria

La abstracción más simple de memoria es ninguna abstracción.
Las primeras computadoras mainframe no tenían abstracción de memoria. **Cada programa veía simplemente la memoria física**.
Al ejecutar 2 programas en simultáneo, uno podía ocupar la dirección de almacenamiento del otro, por lo tanto, **no era posible la multiprogramación**.

Incluso cuando el modelo de memoria consiste en sólo la memoria física hay varias opciones posibles de organizar la memoria con un S.O y un programa de usuario:

1. El S.O puede estar en la parte inferior de la memoria en la RAM
    a. Se utilizó antes en las mainframe y minicomputadoras, pero actualmente no se utiliza
    b. Un error en el programa de usuario puede borrar el sistema operativo
    c. Se puede ejecutar sólo un proceso a la vez
2. El S.O puede estar en la ROM en la parte superior de la memoria
    a. Se utiliza en algunas computadoras de bolsillo y sistemas integrados
3. Los controladores de dispositivos pueden estar en la parte superior de la memoria en una ROM y el resto del sistema en RAM más abajo
    a. Utilizado por las primeras computadoras personales donde la porción del sistema en la ROM se conoce como BIOS
    b. Un error en el programa de usuario puede borrar el sistema operativo
    c. Se puede ejecutar sólo un proceso a la vez

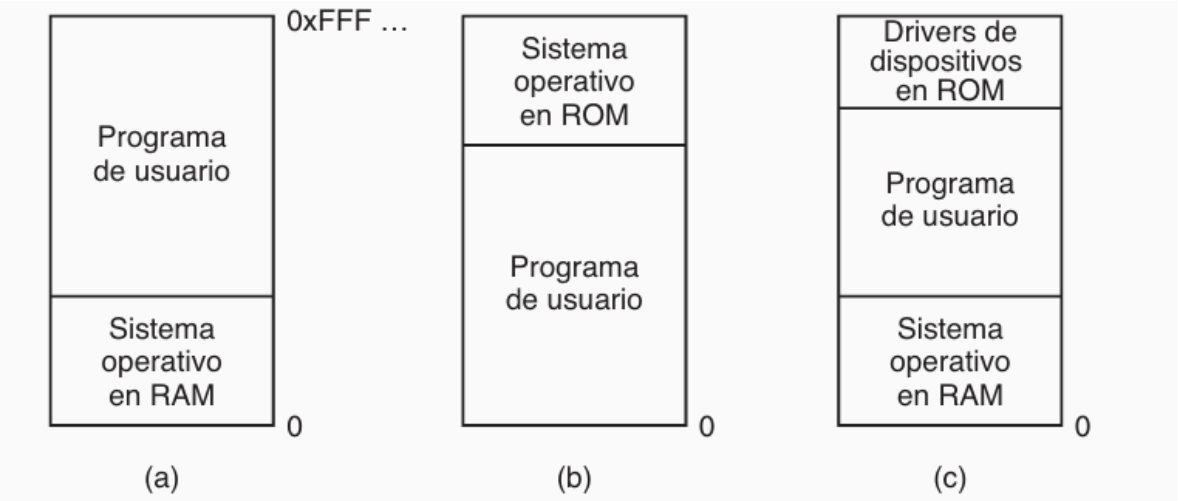

Figura 3-1. Tres formas simples de organizar la memoria con un sistema operativo y un proceso de usuario. También existen otras posibilidades.

Una forma de obtener cierto grado de **paralelismo** en un sistema, sin abstracción de memoria, es **programar con múltiples hilos**. Aunque esta idea funciona, es de uso limitado ya que lo que las personas desean a menudo es que los programas no relacionados se ejecuten al mismo tiempo, algo que la abstracción de los hilos no provee

Ejecución de múltiple programas sin una abstracción de memoria

Aún **sin abstracción de memoria es posible ejecutar varios programas al mismo tiempo**.

Lo que el S.O debe hacer es **guardar todo el contenido de la memoria en un archivo en disco, para después traer y ejecutar el siguiente programa**, este concepto se conoce como **intercambio**. Mientras sólo haya un programa a la vez en la memoria no hay conflictos.

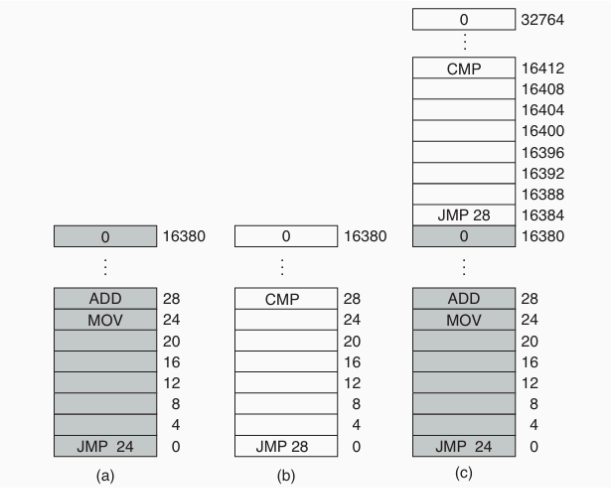

Figura 3-2. Ilustración del problema de reubicación. (a) Un programa de 16 KB. (b) Otro programa de 16 KB. (c) Los dos programas cargados consecutivamente en la memoria.

El problema central aquí es que los dos programas hacen referencia a la memoria física absoluta.

Para solucionar esto aparece la **reubicación estática**, que al cargar un programa en la dirección 'x' se suma el valor constante 'x' a todas las direcciones del programa durante el proceso de carga.

Aunque este mecanismo funciona si se lleva a cabo en la forma correcta, no es una solución muy general y reduce la velocidad de la carga.

La **falta de una abstracción de memoria** sigue siendo **común** en los **sistemas integrados** y de **tarjeta inteligente**. Esto funciona debido a que **todos los programas se conocen de antemano**, y los usuarios no tienen la libertad de ejecutar su propio software en su tostador.

UNA ABSTRACCIÓN DE MEMORIA: ESPACIOS DE DIRECCIONES

Exponer la memoria física a los procesos tiene varias **desventajas**:

- Si los programas de usuario pueden direccionar cada byte de memoria, pueden estropear el sistema operativo con facilidad.
- Es difícil tener varios programas en ejecución a la vez (tomando turnos, si sólo hay una CPU).

Hay que resolver dos problemas para permitir que haya varias aplicaciones en memoria al mismo tiempo sin que interfieran entre sí: **protección** y **reubicación**.
De manera primitiva, el primero puede solucionarse etiquetando trozos de memoria con una llave de protección, también, el segundo podría solucionarse mediante la reubicación de los programas al momento de cargarlos, igualmente estas soluciones son poco efectivas.

Una mejor solución es inventar una nueva **abstracción para la memoria**: el espacio de direcciones.
Un **espacio de direcciones** es el **conjunto de direcciones que puede utilizar un proceso para direccionar la memoria**. Cada proceso tiene su propio espacio de direcciones, independiente de los que pertenecen a otros procesos (exceptuando aquellos que deseen compartirlo). Estos espacios de direcciones pueden ser numéricos o de caracteres.
Algo un poco más difícil es proporcionar a cada programa su propio espacio de direcciones, de manera que la dirección 28 en un programa indique una ubicación física distinta de la dirección 28 en otro programa.

La solución sencilla utiliza una versión muy simple de la **reubicación dinámica**. Consiste en asociar el espacio de direcciones de cada proceso sobre una parte distinta de la memoria física.
La solución clásica es equipar cada CPU con dos registros de hardware especiales, conocidos comúnmente como los registros **base** y **límite**.

- **Base**: Dirección física donde empieza el programa en memoria
- **Limite**: Longitud del programa

A cada dirección de memoria que se genera en forma automática se le suma el contenido del registro base antes de enviarla a memoria. Al mismo tiempo comprueba si la dirección ofrecida es igual o mayor que el valor resultante de sumar los valores de los registros límite y base, en cuyo caso se genera un fallo y se aborta el acceso.

Ej: Primer programa: base 0 y límite 16,384
Segundo programa: base 16,384 y límite 16,384
Tercer programa: base 32,768 y límite 16,384

Una **desventaja** de la reubicación usando los registros base y límite es la n**ecesidad de realizar una suma y una comparación en cada referencia a memoria**.

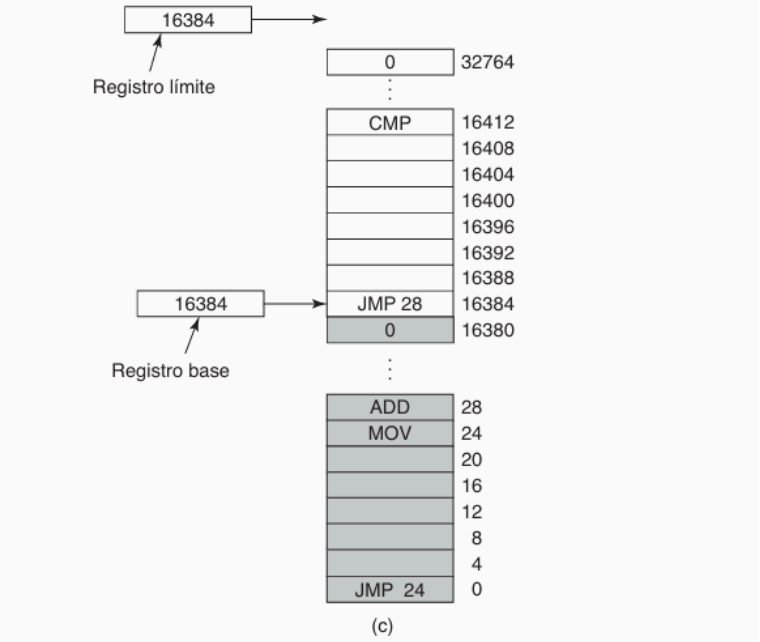

Figura 3-3. Se pueden utilizar registros base y límite para dar a cada proceso un espacio de direcciones separado.

Intercambio

A través de los años se han desarrollado dos esquemas generales para lidiar con la sobrecarga de memoria principal, el **intercambio** y la **memoria virtual**:

Intercambio: Consiste en llevar cada proceso completo a memoria, ejecutarlo durante cierto tiempo y después regresarlo al disco.

Los procesos inactivos mayormente son almacenados en disco, de tal manera que no ocupan memoria cuando no se están ejecutando

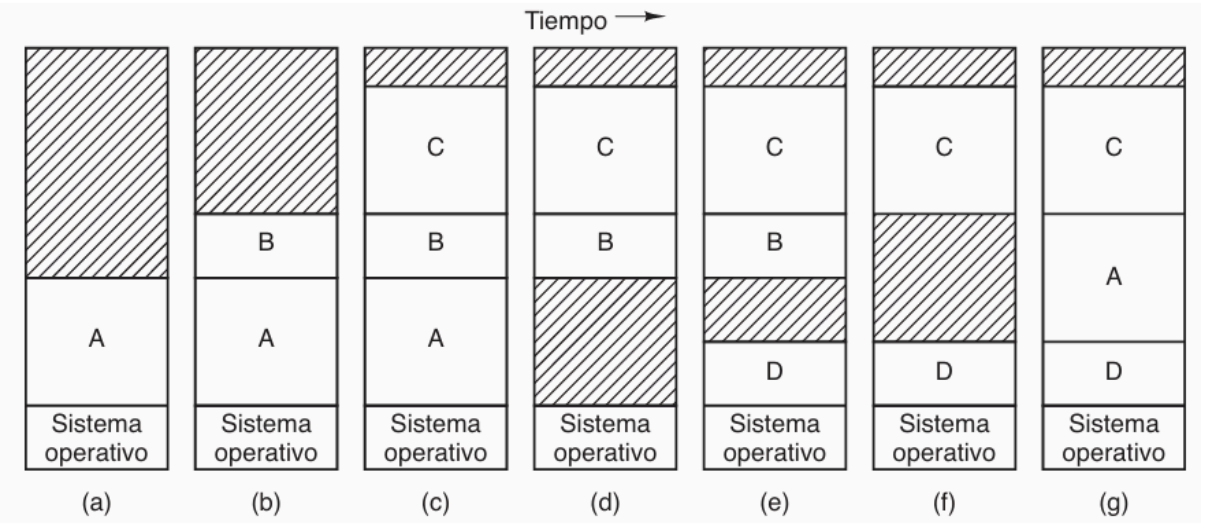

Figura 3-4. La asignación de la memoria cambia a medida que llegan procesos a la memoria y salen de ésta. Las regiones sombreadas son la memoria sin usar.

Cuando el intercambio crea varios huecos en la memoria, es posible combinarlos todos en uno grande desplazando los procesos lo más hacia abajo que sea posible. Esta técnica se conoce como **compactación de memoria**.

Cantidad de memoria que debe **asignarse** a un proceso cuando se crea o intercambia:

- **Proceso de tamaño fijo**:
  - Asignar exactamente lo necesario
- **Proceso de tamaño variable**:
  - Si hay un hueco adyacente al proceso, puede asignarse y se permite al proceso crecer en el hueco
  - Si el proceso está adyacente a otro proceso, el proceso en crecimiento tendrá que moverse a un hueco en memoria que sea lo bastante grande como para alojarlo, o habrá que intercambiar otros procesos para crear un hueco con el tamaño suficiente.
  - Es conveniente asignar un poco de memoria adicional cada vez que se intercambia o se mueve un proceso
  - Si un proceso no puede crecer en memoria y el área de intercambio en el disco está llena, el proceso tendrá que suspenderse hasta que se libere algo de espacio

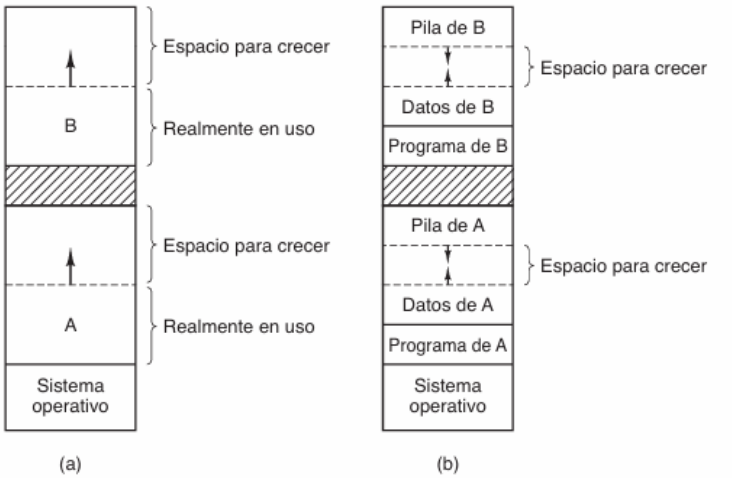

Figura 3-5. (a) Asignación de espacio para un segmento de datos en crecimiento. (b) Asignación de espacio para una pila en crecimiento y un segmento de datos en crecimiento.

Administración de Memoria Libre

Cuando la memoria se asigna en forma dinámica, el sistema operativo debe administrarla.
Hay 2 formas de llevar el registro del uso de memoria, **mapas de bits** y **listas libres/enlazadas**:

- **Mapas de bits**

Con un mapa de bits, la memoria se divide en unidades de asignación. Para cada unidad de asignación hay un bit correspondiente en el mapa de bits, que es 0 si la unidad está libre y 1 si está ocupada (o viceversa).

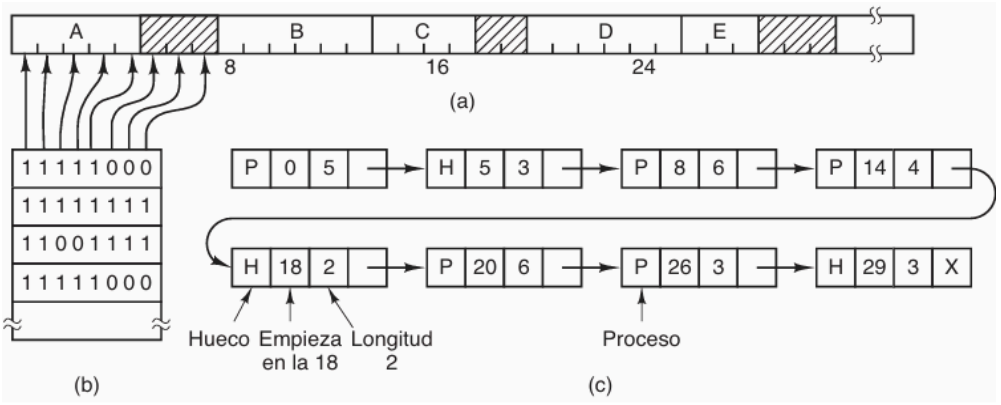

Figura 3-6. (a) Una parte de la memoria con cinco procesos y tres huecos. Las marcas de graduación muestran las unidades de asignación de memoria. Las regiones sombreadas (0 en el mapa de bits) están libres. (b) El mapa de bits correspondiente. (c) La misma información en forma de lista.

El **tamaño de la unidad de asignación** es una importante cuestión de diseño. Entre más **pequeña** sea la unidad de asignación, **mayor será el mapa de bits**.
Si la unidad de asignación se elige de manera que sea **grande**, el **mapa de bits será más pequeño** pero se puede **desperdiciar una cantidad considerable de memoria**.
El **problema principal** del mapa de bits es que el proceso de buscar en un mapa de bits una serie de cierta longitud es una operación lenta.

- **Listas Enlazadas**

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

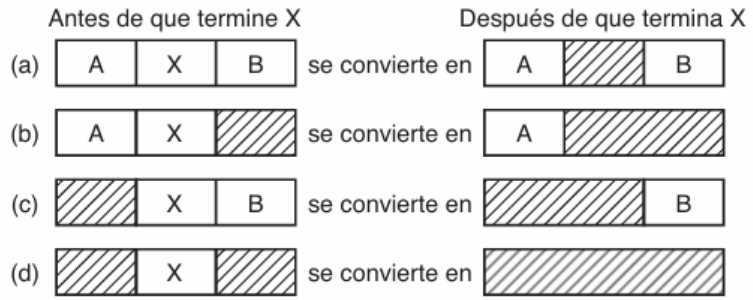

Figura 3-7. Cuatro combinaciones de los vecinos para el proceso en terminación, *X*.

Se pueden utilizar varios **algoritmos para asignar memoria** a un proceso creado (o a un proceso existente que se intercambie del disco):

- **Primer ajuste:** El administrador de memoria explora la lista de segmentos hasta encontrar un hueco que sea lo bastante grande. Después el hueco se divide en dos partes, una para el proceso y otra para la memoria sin utilizar
- **Siguiente ajuste:** Funciona de la misma manera que el primer ajuste, excepto porque lleva un registro de dónde se encuentra cada vez que descubre un hueco adecuado. La siguiente vez que es llamado para buscar un hueco, empieza a buscar en la lista desde el lugar en el que se quedó la última vez. Tiene un rendimiento ligeramente peor que el de primer ajuste
- **Mejor ajuste:** Busca en toda la lista, de principio a fin y toma el hueco más pequeño que sea adecuado. Es más lento que el del primer ajuste, ya que debe buscar en toda la lista cada vez que se le llama
- **Peor ajuste:** Tomar siempre el hueco más grande disponible, de manera que el nuevo hueco sea lo bastante grande como para ser útil

Los cuatro algoritmos pueden ser acelerados manteniendo listas separadas para los procesos y los huecos. Si se mantienen distintas listas para los procesos y los huecos, la lista de huecos se puede mantener ordenada por el tamaño, para que el algoritmo del mejor ajuste sea más rápido.

- **Ajuste rápido:** Mantiene listas separadas para algunos de los tamaños más comunes solicitados.

Memoria Virtual

El problema de que los programas sean más grandes que la memoria ha estado presente desde los inicios de la computación.
Una solución que se adoptó en la década de 1960 fue dividir los programas en pequeñas partes, conocidas como **sobrepuestos** (**overlays**). Al empezar un programa, se ejecutaba el sobrepuesto 0, al finalizar se cargaba el 1 encima del 0 o sobrepuesto en el mismo, de acuerdo a si había espacio, y así sucesivamente. Sin embargo, esto era muy complejo para los programadores, por lo que se ideó una solución mejor: **memoria virtual**, esta permite que los programas se ejecuten incluso cuando solo se encuentran en forma parcial en la memoria.

La idea básica detrás de la **memoria virtual** es que **cada programa tiene su propio espacio de direcciones**, el cual **se divide en trozos llamados páginas**. Cada página es un rango contiguo de direcciones. Estas **páginas se asocian a la memoria física**, **pero no todas tienen que estar en la memoria física** para poder ejecutar el programa.
Cuando el programa hace referencia a una parte de su espacio de direcciones que no está en la memoria física, el sistema operativo recibe una alerta para buscar la parte faltante y volver a ejecutar la instrucción que falló.

Paginación

Los programas hacen referencia a un conjunto de direcciones de memoria. Hay diferentes formas de generar direcciones. Estas direcciones generadas por el programa se conocen como **direcciones virtuales** y forman el **espacio de direcciones virtuale**s. Al usar memoria virtual, las direcciones virtuales van a una MMU (Unidad de administración de memoria) que asocia direcciones virtuales a las direcciones de memoria físicas.
La mayor parte de los sistemas de memoria virtual utilizan una técnica llamada **paginación**.
Direcciones virtuales: Son direcciones generadas por el programa, forman el **espacio de direcciones virtuales**.

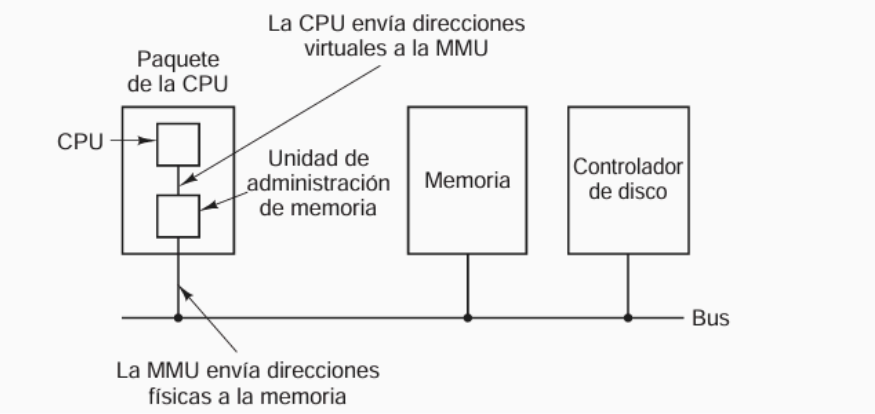

Figura 3-8. La posición y función de la MMU. Aquí la MMU se muestra como parte del chip de CPU, debido a que es común esta configuración en la actualidad. Sin embargo, lógicamente podría ser un chip separado y lo era hace años.

El espacio de direcciones virtuales se divide en unidades de tamaño fijo llamadas **páginas**. Las unidades correspondientes en la memoria física se llaman **marcos de página**. Las páginas y los marcos de página por lo general son del mismo tamaño.

- Las transferencias entre la RAM y el disco siempre son en páginas completas.
- La memoria no sabe nada acerca de la MMU y sólo ve una petición para leer o escribir en la dirección.
- En el hardware real, un bit de presente/ausente lleva el registro de cuáles páginas están físicamente presentes en la memoria.

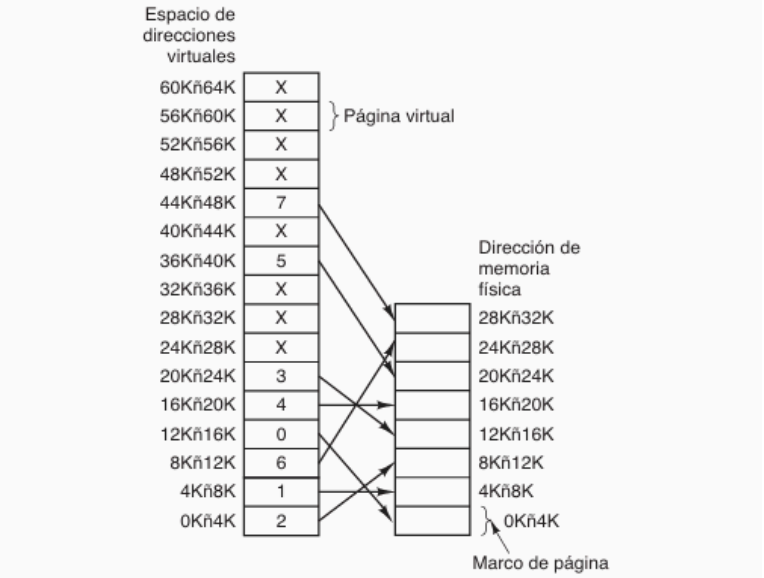

Figura 3-9. La relación entre las direcciones virtuales y las direcciones de memoria física está dada por la tabla de páginas. Cada página empieza en un múltiplo de 4096 y termina 4095 direcciones más arriba, por lo que de 4 K a 8 K en realidad significa de 4096 a 8191 y de 8 K a 12 K significa de 8192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

Ejemplo de cómo funciona la **asociación**

Si el programa hace referencia a direcciones no asociadas la MMU detecta que la página no está asociada y hace que la CPU haga un trap al sistema operativo. A este trap se le llama **fallo de página**. El S.O selecciona un marco de página que se utilice poco y escribe su contenido de vuelta al disco, después obtiene la página que se acaba de referenciar en el marco de página que se acaba de liberar, cambia la asociación y reinicia la instrucción que originó el trap.

Tablas de páginas

Una **tabla de páginas** es una estructura de datos utilizada por el sistema operativo para mapear las direcciones virtuales (o lógicas) a direcciones físicas en la memoria.
Cuando un programa accede a una dirección virtual, la unidad de administración de memoria (MMU) busca en la tabla de páginas para obtener la correspondiente dirección física. Si la página está presente en la memoria principal, se devuelve la dirección física, lo que permite acceder a la memoria. Si no está presente, se produce un fallo de página y el sistema operativo debe cargar la página desde el disco al espacio de memoria física

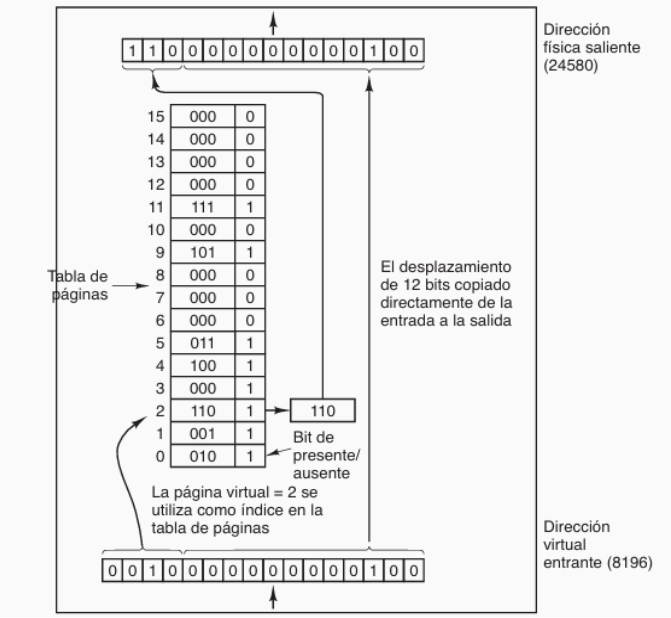

Figura 3-10. La operación interna de la MMU con 16 páginas de 4 KB.

Estructura de una entrada en la tabla de páginas

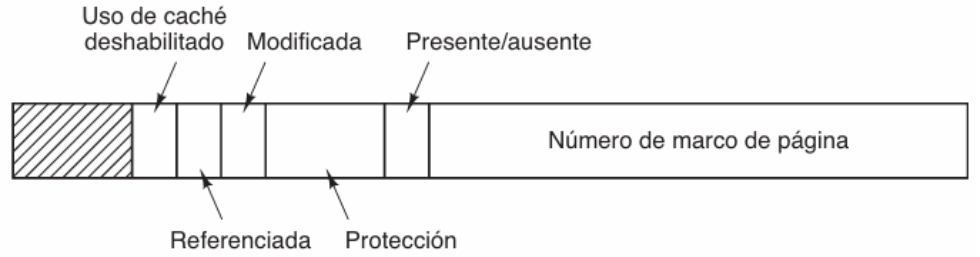

Figura 3-11. Una típica entrada en la tabla de páginas.

- **Número de marco de página:**
  - Este campo es el más importante en la entrada de la tabla de páginas.
  - Representa la dirección física en la memoria principal donde se encuentra la página correspondiente.
  - El objetivo principal de la tabla de páginas es mapear direcciones virtuales a estas direcciones físicas.
- **Bit de presente/ausente:**
  - Si este bit es 1, la entrada es válida y la página está actualmente en la memoria principal.
  - Si es 0, la página virtual a la que pertenece la entrada no está en la memoria y se produce un **fallo de página** al acceder a ella.
- **Bits de protección:**
  - Estos bits indican qué tipo de acceso está permitido a la página.
  - En su forma más simple, este campo contiene 1 bit:
    - 0 para lectura/escritura.
    - 1 para solo lectura.
  - En un arreglo más sofisticado, se pueden utilizar 3 bits separados:
    - Uno para habilitar la lectura.
    - Otro para la escritura.
    - El tercero para ejecutar la página (permisos de ejecución).
- **Bits de modificada y referenciada:**

- ➢ Estos bits llevan el registro del uso de las páginas.
- ➢ El **bit de modificada**:
  - Indica si la página ha sido modificada (escrita) desde que se cargó en la memoria.
  - A veces se le llama "bit sucio" porque refleja el estado de la página.
- ➢ El **bit de referenciada**:
  - Se establece cada vez que se accede a la página (ya sea para lectura o escritura).
  - Ayuda al sistema operativo a decidir qué página desalojar cuando ocurre un fallo de página.

❖ **Uso de caché deshabilitado:**
- ➢ Bit que permite deshabilitar el uso de caché para la página.

La tabla de páginas sólo guarda la información que el hardware necesita para traducir una dirección virtual en una dirección física. Es por esto que la dirección de disco utilizado para guardar la página cuando no está en memoria no forma parte de la tabla de páginas. La información que necesita el S.O para manejar los fallos de páginas se mantienen en tablas de software dentro del S.O.

Aceleración de la paginación

En cualquier sistema de paginación hay que abordar dos cuestiones principales:
1. La asociación de una dirección virtual a una dirección física debe ser rápida.
2. Si el espacio de direcciones virtuales es grande, la tabla de páginas será grande.

El primer punto es una consecuencia del hecho de que la asociación virtual a física debe realizarse en cada referencia de memoria.
El segundo punto se deriva del hecho de que si hay muchos procesos en ejecución simultáneamente, la tabla de páginas puede volverse muy grande.

Búferes de traducción adelantada

El punto inicial de la mayor parte de las técnicas de optimización es que la tabla de páginas está en la memoria. Potencialmente, este diseño tiene un enorme impacto sobre el rendimiento.
Se ha ideado una solución que está basada en la observación de que la mayor parte de los programas tienden a hacer un gran número de referencias a un pequeño número de páginas y no viceversa.

El **TBL** o **memoria asociativa** es un pequeño dispositivo de hardware equipado en las computadoras que sirve para asociar direcciones virtuales a direcciones físicas sin pasar por la tabla de páginas. Este se encuentra dentro de la MMU y consiste en un pequeño número de entradas, cada una de estas contiene:

- ❖ Numero de pagina virtual
- ❖ Un bit que se establece cuando se modifica la página
- ❖ Código de protección (permisos de lectura/escritura/ejecución)
- ❖ Marco de página físico en el que se encuentra la página.
- ❖ Bit indica si la entrada es válida (si está en uso) o no

Estos campos tienen una correspondencia de uno a uno con los campos en la tabla de páginas, excepto por el número de página virtual, que no se necesita en la tabla de páginas.

Válida	Página virtual	Modificada	Protección	Marco de página
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

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

Funcionamiento del TLB:
Cuando se presenta una dirección virtual a la MMU para que la traduzca, el hardware primero comprueba si su número de página virtual está presente en el TLB:

- Si se encuentra una coincidencia válida y el acceso no viola los bits de protección, el marco de página se toma directamente del TLB, sin pasar por la tabla de páginas.
- Si el número de página virtual está presente en el TLB, pero la instrucción está tratando de escribir en una página de sólo lectura, se genera un fallo por protección.
- Si el número de página virtual no está en el TLB, la MMU realizará una búsqueda ordinaria en la tabla de páginas, después desaloja una entrada en la TLB y la reemplaza con la entrada en la tabla de páginas que acaba de buscar. De esta forma, si esa página se utiliza pronto otra vez, la segunda vez se producirá un acierto en el TLB en vez de un fracaso.

Administración del TLB mediante software

En el pasado, la administración y el manejo de fallas del TLB se realizaban por completo mediante el hardware de la MMU. Las traps, o trampas, para el sistema operativo ocurren sólo cuando una página no se encuentra en memoria.

Actualmente, muchas máquinas RISC modernas hacen casi toda esta administración de páginas mediante software.
En estas máquinas, cuando no se encuentra una coincidencia en el TLB, la MMU genera un fallo del TLB y pasa el problema al sistema operativo. El sistema debe buscar la página, eliminar una entrada del TLB, introducir la nueva página y reiniciar la instrucción que originó el fallo.

Cuando se utiliza la administración del TLB mediante software, es esencial comprender la diferencia entre dos **tipos de fallos**:

- **Fallo suave**: Ocurre cuando la página referenciada no está en el TLB, sino en memoria. Todo lo que se necesita aquí es que el TLB se actualice. No se necesita E/S de disco.
- **Fallo duró**: Ocurre cuando la misma página no está en memoria (y desde luego, tampoco en el TLB). Se requiere un acceso al disco para traer la página.

Tablas de páginas para memorias extensas

Estas ayudan a lidiar con el problema de espacios de direcciones virtuales muy extensos. Hay dos maneras de hacerlo:

Tablas de páginas multinivel

En un sistema de **tablas de páginas multinivel**, las tablas de páginas convencionales se dividen en niveles.

Cada nivel de la tabla de páginas multinivel se encarga de una parte específica del espacio de direcciones virtuales, además, cada una tiene su propia tabla de páginas, y las entradas en cada tabla apuntan a otras tablas o marcos de página físicos.

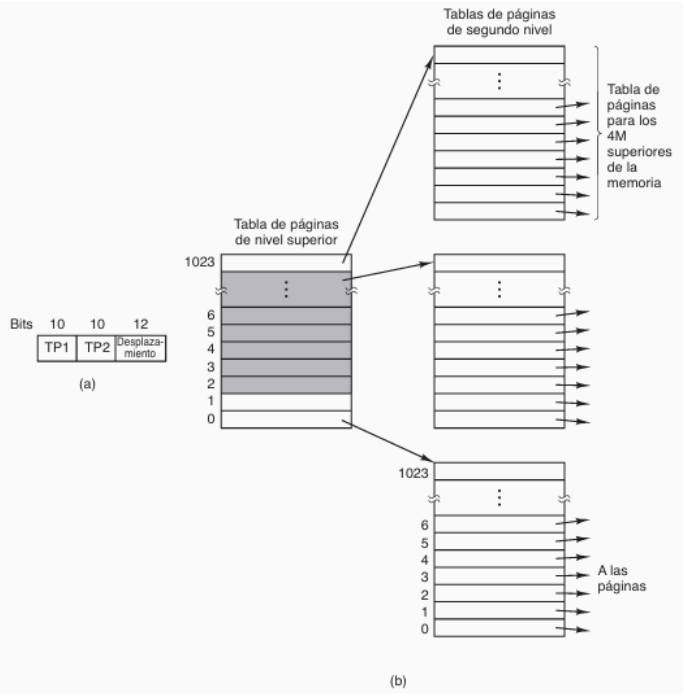

Figura 3-13. (a) Una dirección de 32 bits con dos campos de tablas de páginas. (b) Tablas de páginas de dos niveles.

Tablas de páginas invertidas

Para los espacios de direcciones virtuales de 32 bits, la tabla de páginas multinivel funciona bastante bien. Para los espacios de direcciones virtuales paginados de 64 bits se necesita una solución diferente.

Una de esas soluciones es la tabla de **páginas invertida**. En este diseño hay una entrada por cada marco de página en la memoria real, en vez de tener una entrada por página de espacio de direcciones virtuales, permitiendo ahorrar grandes cantidades de espacio.

Cuando el espacio de direcciones virtuales es mucho mayor que la memoria física, tienen una seria **desventaja**: la traducción de dirección virtual a dirección física se hace mucho más difícil.

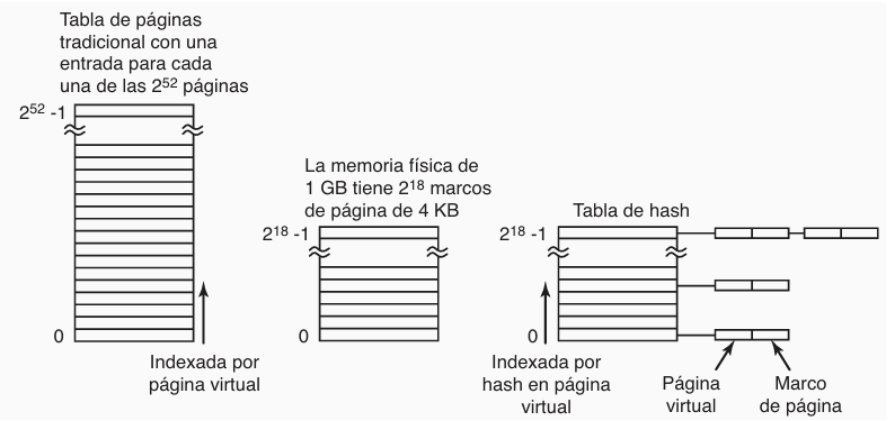

Figura 3-14. Comparación de una tabla de páginas tradicional con una tabla de páginas invertida.

ALGORITMOS DE REEMPLAZO DE PÁGINAS

Cuando ocurre un fallo de página, el sistema operativo tiene que elegir una página para desalojarla (eliminarla de memoria) y hacer espacio para la página entrante.
Aunque sería posible elegir una página al azar para desalojarla en cada fallo de página, el rendimiento del sistema es mucho mayor si se selecciona una página que no sea de uso frecuente.
Existen diferentes algoritmos de reemplazo de páginas:

- **Algoritmo de Reemplazo de Páginas: Óptimo:**

Cada página se puede etiquetar con el número de instrucciones que se ejecutarán antes de que se haga referencia por primera vez a esa página. El algoritmo óptimo de reemplazo de páginas establece que la página con la etiqueta más alta debe eliminarse.
El único problema con este algoritmo es que **no se puede realizar**. Al momento del fallo de página, el sistema operativo no tiene forma de saber cuándo será la próxima referencia a cada una de las páginas

- **Algoritmo de Reemplazo de Páginas: No Usadas Recientemente (NRU)**

Para permitir que el sistema operativo recolecte estadísticas útiles sobre el uso de páginas, las computadores con memoria virtual tienen 2 bits:

- **Bit R**: Se establece cada vez que se hace referencia a la página (lectura o escritura)
- **Bit M**: Se establece cuando se escribe en la página (es decir, se modifica).

Los bits R y M se pueden utilizar para construir un algoritmo simple de paginación de la si siguiente manera.
Cuando se inicia un proceso, ambos bits de página para todas sus páginas se establecen en 0 mediante el sistema operativo. El bit R se borra en forma periódica (en cada interrupción de reloj) para diferenciar las páginas a las que no se ha hecho referencia recientemente de las que sí se han referenciado.
Cuando ocurre un fallo de página, el sistema operativo inspecciona todas las páginas y las divide en 4 categorías con base en los valores actuales de sus bits R y M:

- **Clase 0**: no ha sido referenciada, no ha sido modificada.
- **Clase 1**: no ha sido referenciada, ha sido modificada.
- **Clase 2**: ha sido referenciada, no ha sido modificada.
- **Clase 3**: ha sido referenciada, ha sido modificada.

El algoritmo NRU elimina una página al azar de la clase de menor numeración que no esté vacía, está implícita la idea de que es **mejor eliminar una página modificada a la que no se haya hecho referencia en al menos un pulso de reloj que una página limpia de uso frecuente.**

- **Algoritmo de Reemplazo de Páginas: Primera en entrar, primera en salir (FIFO)**

El sistema operativo mantiene una lista de todas las páginas actualmente en memoria, en donde la llegada más reciente está en la parte final y la menos reciente en la parte frontal. En un fallo de página, se elimina la página que está en la parte frontal y la nueva página se agrega a la parte final de la lista.
Este algoritmo puede eliminar páginas importantes, por lo tanto es raro que se utilice FIFO en su forma pura.

- **Algoritmo de Reemplazo de Páginas: Segunda oportunidad**

Una modificación simple al algoritmo FIFO que evita el problema de descartar una página de uso frecuente es **inspeccionar el bit R** de la página más antigua.

- Si es 0, la página es antigua y no se ha utilizado, por lo que se sustituye de inmediato.
- Si es 1, el bit se borra, la página se pone al final de la lista de páginas y su tiempo de carga se actualiza, como si acabara de llegar a la memoria.

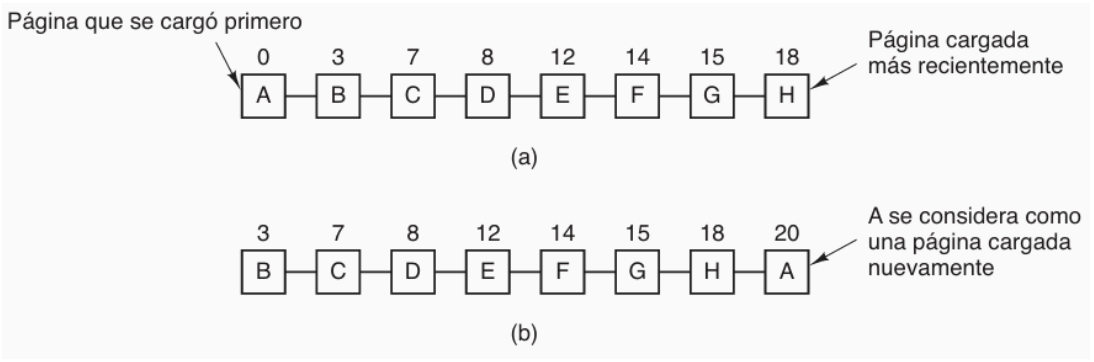

Figura 3-15. Operación del algoritmo de la segunda oportunidad. (a) Páginas ordenadas con base en FIFO. (b) Lista de las páginas si ocurre un fallo de página en el tiempo 20 y *A* tiene su bit *R* activado. Los números encima de las páginas son sus tiempos de carga.

Lo que el algoritmo de segunda oportunidad está buscando es una página antigua a la que no se haya hecho referencia en el intervalo de reloj más reciente.
Si se ha hecho referencia a todas las páginas, el algoritmo segunda oportunidad se degenera y se convierte en FIFO puro.

- **Algoritmo de Reemplazo de Páginas: Reloj**

En este los marcos de página se mantienen una lista circular en forma de reloj, donde la manecilla apunta a la página más antigua.

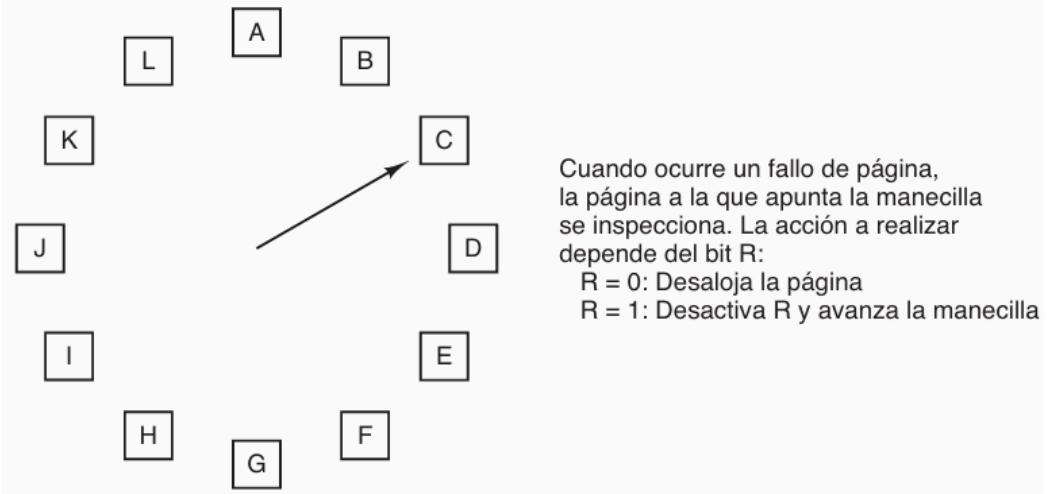

Figura 3-16. El algoritmo de reemplazo de páginas en reloj.

- **Algoritmo de Reemplazo de Páginas: Menos usadas recientemente (LRU)**

Esta estrategia plantea que cuando ocurra un fallo de página, hay que descartar la página que no se haya utilizado durante la mayor longitud de tiempo.
Hay otras formas de implementar el algoritmo LRU con hardware especial. Este método requiere equipar el hardware con un contador el cual se incrementa de manera automática después de cada instrucción.

Cuando ocurre un fallo de página, el sistema operativo examina todos los contadores en la tabla de páginas para encontrar el menor. Esta página es la de uso menos reciente.

Ahora veamos un segundo algoritmo LRU mediante hardware. Para una máquina con n marcos de página, el hardware del LRU puede mantener una matriz de nxn bits (inicialmente, todos son 0). Cada vez que se hace referencia a la página k, el hardware primero establece todos los bits de la fila ken 1 y después todos los bits de la columna k en 0. En cualquier instante, la fila cuyo valor binario sea menor es la de uso menos reciente, la fila cuyo valor sea el siguiente más bajo es la del siguiente uso menos reciente, y así en lo sucesivo.

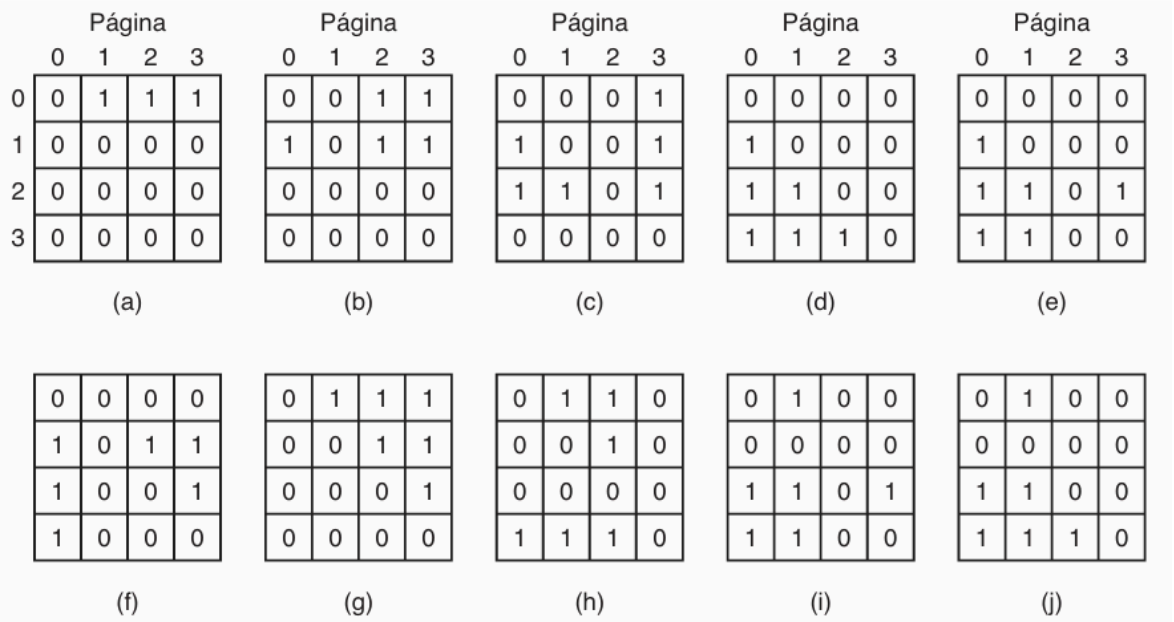

Figura 3-17. LRU usando una matriz cuando se hace referencia a las páginas en el orden 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

Simulación de LRU en software (NFU)

Este algoritmo requiere un contador de software asociado con cada página, que al principio es cero. En cada interrupción de reloj, el S.O explora todas las páginas en memoria. Para cada página se agrega el bit R, que es 0 o 1, al contador. Los contadores llevan la cuenta aproximada de la frecuencia con que se hace referencia a cada página. Cuando ocurre un fallo de página, se selecciona la página con el contador que sea menor para sustituirla.

Resumen de los algoritmos de reemplazo de páginas

Algoritmo	Comentario
Óptimo	No se puede implementar, pero es útil como punto de comparación
NRU (No usadas recientemente)	Una aproximación muy burda del LRU
FIFO (primera en entrar, primera en salir)	Podría descartar páginas importantes
Segunda oportunidad	Gran mejora sobre FIFO
Reloj	Realista
LRU (menos usadas recientemente)	Excelente, pero difícil de implementar con exactitud
NFU (no utilizadas frecuentemente)	Aproximación a LRU bastante burda
Envejecimiento	Algoritmo eficiente que se aproxima bien a LRU
Conjunto de trabajo	Muy costoso de implementar
WSClock	Algoritmo eficientemente bueno

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

CUESTIONES DE DISEÑO PARA LOS SISTEMAS DE PAGINACIÓN

Políticas de asignación local contra las de asignación global

Una cuestión importante asociada con la selección de qué página sustituir al ocurrir un fallo es **cómo se debe asignar la memoria entre los procesos ejecutables en competencia**.
Existen 2 tipos de algoritmos para asignar esta memoria:

- **Algoritmos Locales**: Corresponden de manera efectiva a asignar a cada proceso una fracción fija de la memoria.
  - Si se utiliza un algoritmo local y el conjunto de trabajo crece, se producirá una **sobrepaginación**, aun cuando haya muchos marcos de página libres.
  - Si el conjunto de trabajo se reduce, los algoritmos locales desperdician memoria.
- **Algoritmos Globales**: Asignan marcos de página de manera dinámica entre los procesos ejecutables. Así, el número de marcos de página asignados a cada proceso varía en el tiempo.
  - Funcionan mejor cuando el tamaño del conjunto de trabajo puede variar durante el tiempo de vida de un proceso.
  - El sistema debe decidir en forma continua cuántos marcos de página asignar a cada proceso.

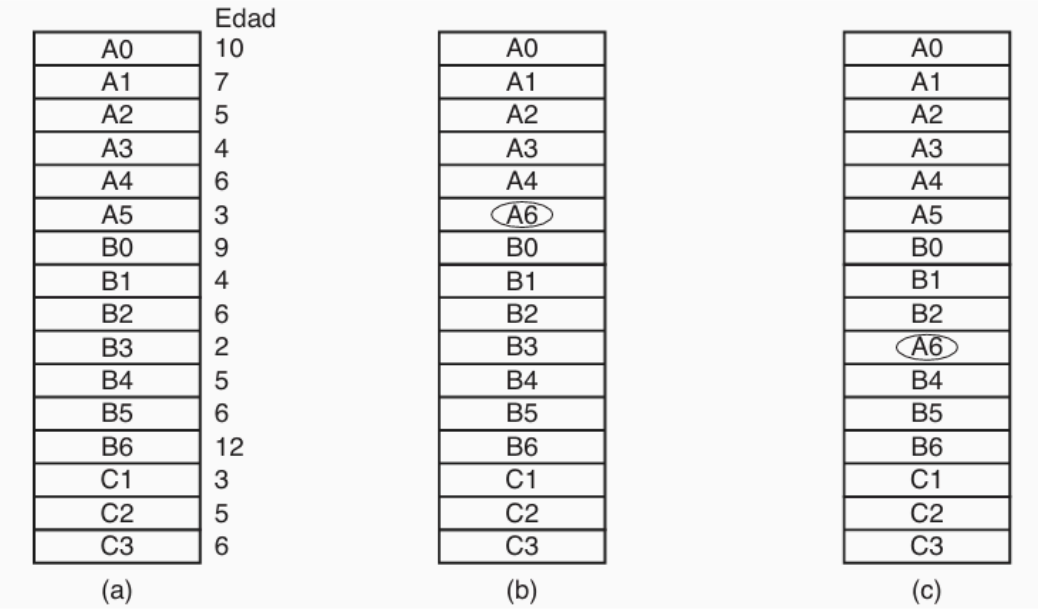

Figura 3-23. Comparación entre el reemplazo de páginas local y el global. (a) Configuración original. (b) Reemplazo de páginas local. (c) Reemplazo de páginas global.

Existen distintas maneras de decidir el tamaño del conjunto de trabajo en los algoritmos globales:

- Tener un algoritmo para asignar marcos de página a los procesos. Una manera es determinar periódicamente el número de procesos en ejecución y asignar a cada proceso una parte igual.
- Es posible empezar cada proceso con cierto número de páginas proporcional al tamaño del proceso, pero la **asignación se tiene que actualizar** dinámicamente a medida que se ejecuten los procesos

Una manera de **administrar la asignación** es utilizando el **algoritmo PFF** (Frecuencia de fallo de páginas). Este algoritmo indica cuándo se debe incrementar o decrementar la asignación de páginas a

un proceso, pero no dice nada acerca de cuál página se debe sustituir en un fallo. Sólo controla el tamaño del conjunto de asignación.

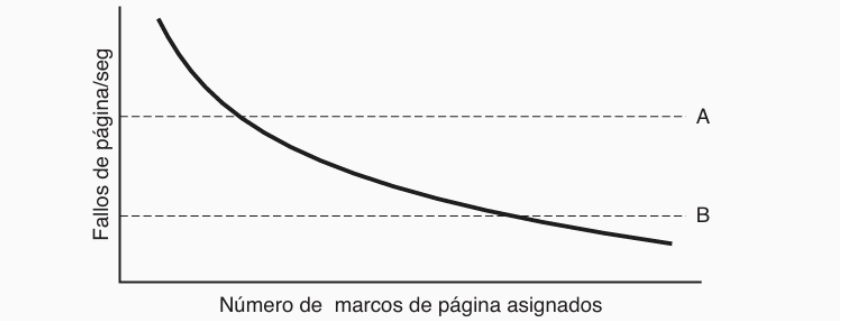

Figura 3-24. Proporción de fallos de página como una función del número de marcos de página asignados.

Control de Cargas

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

Cada vez que los conjuntos de trabajo combinados de todos los procesos exceden a la capacidad de la memoria, se puede esperar la sobrepaginación.
La única solución real es deshacerse temporalmente de algunos procesos.

Una buena forma de reducir el número de procesos que compiten por la memoria es **intercambiar** algunos de ellos **enviándolos al disco** y liberar todas las páginas que ellos mantienen.

Otro factor a considerar es el **grado de multiprogramación**. Cuando el número de procesos en la memoria principal es demasiado bajo, la CPU puede estar inactiva durante largos periodos.

Tamaño de las Páginas

El tamaño de página es un parámetro que a menudo **el sistema operativo puede elegir**.
Para determinar el mejor tamaño de página se requiere balancear varios factores competitivos, es decir, no hay un tamaño óptimo en general.
Un tamaño de **página grande** hará que haya una parte más grande no utilizada del programa que un tamaño de página pequeño.
Por otro lado, tener **páginas pequeñas** implica que los programas necesitarán muchas páginas, lo que sugiere la necesidad de una tabla de páginas grande.

La **fragmentación interna** es el espacio adicional que hay en la página, el cual se desperdicia.

Espacios separados de instrucciones y de datos

La mayor parte de las computadoras tienen un solo espacio de direcciones que contiene tanto programas como datos. Si este espacio de direcciones es lo bastante grande, todo funciona bien, pero de lo contrario, no.
Una solución es tener espacios de direcciones separados para las instrucciones y los datos, llamados **espacio I** y **espacio D**. Estos espacios:

- Se pueden paginar de manera independiente
- Cada uno tiene su propia tabla de páginas, con su propia asignación de páginas virtuales a marcos de páginas físicas.
- Cuando el hardware desea obtener una instrucción, sabe qué espacio utilizar

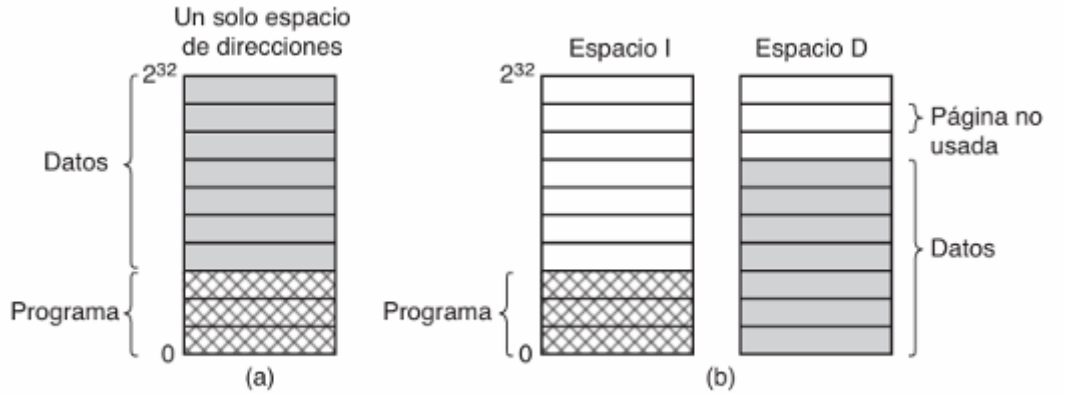

Figura 3-25. (a) Un espacio de direcciones. (b) Espacios I y D separados.

Páginas compartidas

Otra cuestión de diseño es la compartición. En un sistema de multiprogramación grande, es común que varios usuarios ejecuten el mismo programa a la vez es más eficiente compartir las páginas para evitar tener dos copias de la misma página en memoria al mismo tiempo. Un problema es que no todas las páginas se pueden compartir (solo se pueden compartir páginas de lectura y no de datos).

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

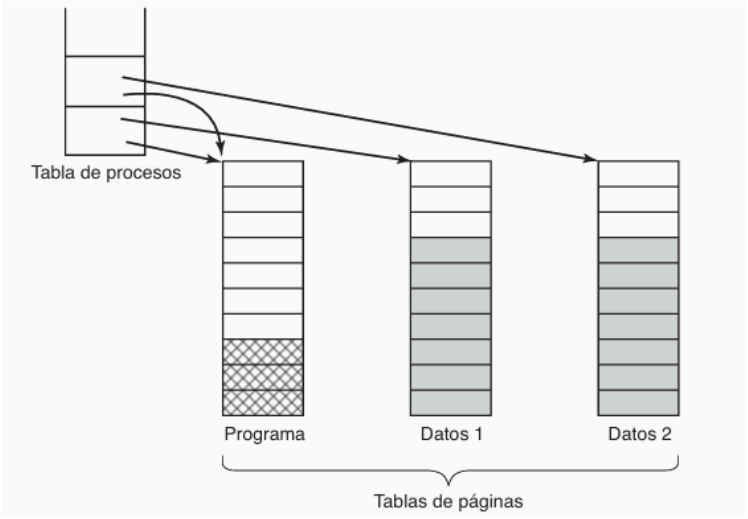

Figura 3-26. Dos procesos comparten el mismo programa compartiendo su tabla de páginas.

El método **copiar en escritura**, nos permite mejorar el rendimiento al reducir el copiado. Consiste en dar a cada proceso su propia tabla de páginas y hacer que ambos apunten al mismo conjunto de páginas, teniendo las páginas de datos en SOLO LECTURA.

Tan pronto como cualquiera de los procesos actualiza una palabra de memoria, la violación de la protección de sólo lectura produce un trap al sistema operativo. Después se hace una copia de la página ofensora, para que cada proceso tenga ahora su propia **copia privada**.

Bibliotecas compartidas

Cuando un programa se vincula con bibliotecas compartidas, en vez de incluir la función a la que se llamó, el vinculador incluye una pequeña rutina auxiliar que se enlaza a la función llamada en tiempo de ejecución. Las bibliotecas compartidas se cargan cuando se carga el programa o cuando las funciones en ellas se llaman por primera vez.

Cuando se carga o utiliza una biblioteca compartida, no se lee toda la biblioteca en memoria de un solo golpe. Se página una página a la vez según sea necesario, de manera que las funciones que no sean llamadas no se carguen en la **RAM**.

Las **ventajas** de esto es que reduce el tamaño de los archivos ejecutables, ahorra espacio de memoria y si una función en una biblioteca compartida se actualiza para eliminar un error, no es necesario recompilar los programas que la llaman, pues los antiguos binarios siguen funcionando.

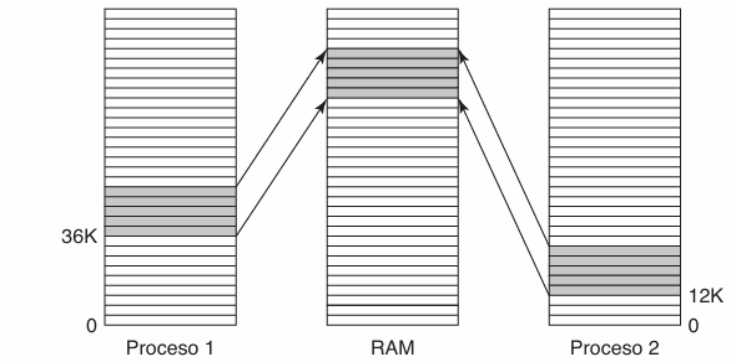

Figura 3-27. Una biblioteca compartida utilizada por dos procesos.

Política de limpieza

La paginación funciona mejor cuando hay muchos marcos de página libres que se pueden reclamar al momento en que ocurran fallos de página.
Demonios de paginación → Es un proceso en segundo plano que está inactivo la mayor parte del tiempo, pero se despierta en forma periódica para inspeccionar el estado de la memoria, se usa con el fin de asegurar una provisión abundante de marcos de páginas libres.

Si hay muy pocos marcos de página libres, el demonio de paginación empieza a seleccionar páginas para desalojarlas mediante cierto algoritmo de reemplazo de página. En cualquier caso, se recuerda el contenido anterior de la página.

Al mantener una provisión de marcos de página a la mano se obtiene un mejor rendimiento que al utilizar toda la memoria y después tratar de encontrar un marco al momento de necesitarlo.

Interfaz de memoria virtual

En algunos sistemas avanzados, los programadores tienen cierto control sobre el mapa de memoria. Una razón por la que se otorga a los programadores el control sobre su mapa de memoria es para permitir que dos o más procesos compartan la misma memoria. un proceso escribe en la memoria compartida y otro proceso lee de ella.
Otra técnica más de administración avanzada de memoria es la **memoria compartida distribuida**. La idea aquí es permitir que varios procesos compartan a través de la red un conjunto de páginas, posiblemente (pero no es necesario) como un solo espacio de direcciones lineal compartido.

Segmentación

Tener dos o más espacios de direcciones virtuales separados puede ser mucho mejor que tener uno sólo. Para lograrlo se tendría que asignar trozos contiguos de espacios de direcciones virtuales a cada tabla.
La solución simple es proporcionar la máquina con muchos espacios de direcciones por completo independientes, llamados **segmentos**.
Cada **segmento** consiste en una secuencia lineal de direcciones, desde 0 hasta un valor máximo.
La longitud de cada segmento no siempre es igual en cada uno de ellos y puede variar durante la ejecución.
Debido a que cada segmento constituye un espacio de direcciones separado, los distintos **segmentos pueden crecer o reducirse** de manera independiente, sin afectar unos a otros.

Para especificar una dirección en esta memoria segmentada el programa debe suministrar una dirección en dos partes, un número de segmento y una dirección dentro del segmento.

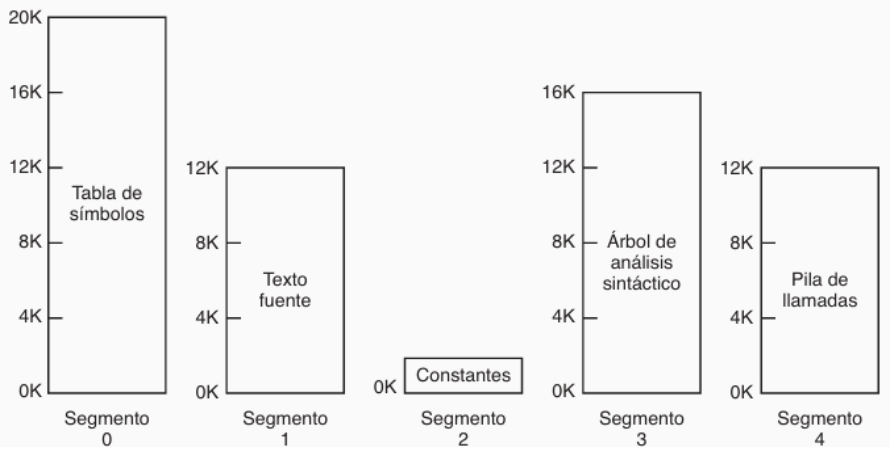

Figura 3-32. Una memoria segmentada permite que cada tabla crezca o se reduzca de manera independiente a las otras tablas.

Implementación de la Segmentación

La implementación de segmentación difiere de la paginación de una manera esencial: las páginas tienen un tamaño fijo y los segmentos no.

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

Figura 3-33. Comparación de la paginación y la segmentación.

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

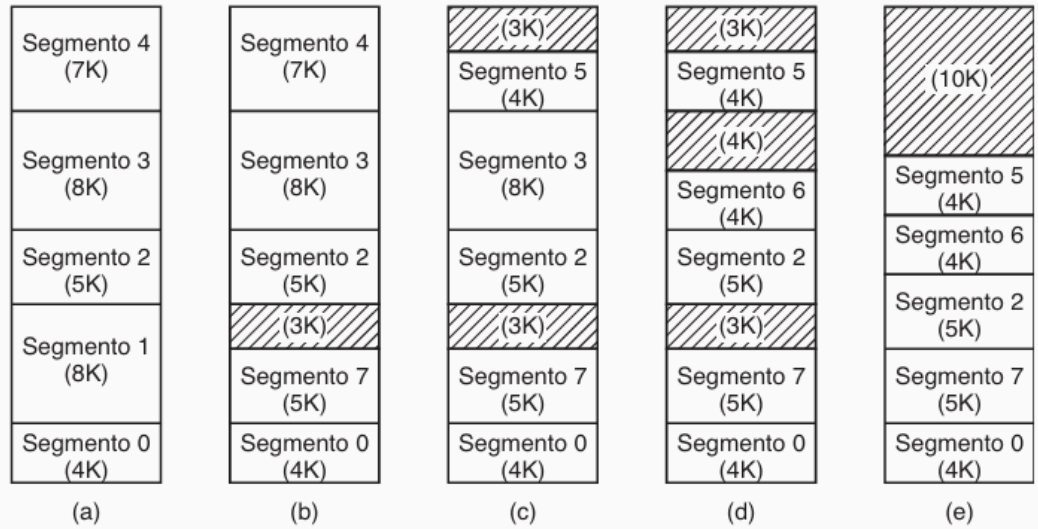

Figura 3-34. (a)-(d) Desarrollo del efecto de tablero de ajedrez. (e) Eliminación del efecto de tablero de ajedrez mediante la compactación.

La figura 3-34(a) muestra un ejemplo de una memoria física que al principio contiene cinco segmentos. Ahora considere lo que ocurre si el segmento 1 se desaloja y el segmento 7, que es más pequeño, se coloca en su lugar. Obtendremos la configuración de memoria de la figura 3-34(b). Entre el segmento 7 y el 2 hay un área sin uso; es decir, un hueco. Después el segmento 4 se reemplaza por el segmento 5, como en la figura 3-34(c), y el segmento 3 se reemplaza por el segmento 6, como en la figura 3-34(d)

Segmentación con Paginación

Esto consiste en paginar los segmentos, de manera que sólo las páginas que realmente se necesiten tengan que estar presente. Esta idea surge de que cuando los segmentos son extensos, puede ser inconveniente (o incluso imposible) mantenerlos completos en memoria.
Hay varios sistemas importantes que han soportado segmentos de páginas, podemos nombrar a **MULTICS** y al más reciente el **Intel Pentium**

Sistemas de Archivos

Todas las aplicaciones de computadoras requieren almacenar y recuperar información.

Problemáticas:

- **Espacio de direcciones limitado**, la capacidad de almacenamiento está restringida por el tamaño del espacio de direcciones virtuales.
- **Persistencia**, cuando el proceso termina, la información se pierde. Es inaceptable que la información se desvanezca cuando el proceso que la utiliza termina (por sí solo o por una falla en la computadora).
- **Concurrencia**, es necesario que varios procesos accedan a la información al mismo tiempo.

Tenemos tres requerimientos esenciales para el almacenamiento de información a largo plazo:

1. Debe ser posible almacenar una cantidad muy grande de información.
2. La información debe sobrevivir a la terminación del proceso que la utilice.
3. Múltiples procesos deben ser capaces de acceder a la información concurrentemente.

El **sistema de archivos** es la parte del S.O que trata con los archivos.

Los **archivos** son unidades lógicas de información creada por los **procesos**, estos procesos **pueden leer los archivos existentes y crear otros si es necesario**.
La información que se almacena en los archivos es **persistente** y debe desaparecer sólo cuando su propietario lo remueve de manera explícita.
Estos archivos son **administrados por el sistema operativo**.

ARCHIVOS

Los archivos son un mecanismo de abstracción. Proporcionan una manera de almacenar información en el disco y leerla después

Nomenclatura de archivos

Hace referencia a cómo se nombran y clasifican los S.A.
Las reglas exactas para denominar archivos varían un poco de un sistema a otro.
Actualmente todos los sistemas operativos permiten cadenas de una a ocho letras como nombres de archivos, y muchos hasta 255 caracteres.

- Los sistemas UNIX diferencian letras mayúsculas de las minúsculas

- Los sistemas MS-DOS (FAT-16 y FAT-32) no lo diferencian

La parte que va después del punto se conoce como la **extensión del archivo** y por lo general indica algo acerca de su naturaleza.

Extensión	Significado
archivo.bak	Archivo de respaldo
archivo.c	Programa fuente en C
archivo.gif	Imagen en Formato de Intercambio de Gráficos de CompuServe
archivo.hlp	Archivo de ayuda
archivo.html	Documento en el Lenguaje de Marcación de Hipertexto de World Wide Web
archivo.jpg	Imagen fija codificada con el estándar JPEG
archivo.mp3	Música codificada en formato de audio MPEG capa 3
archivo.mpg	Película codificada con el estándar MPEG
archivo.o	Archivo objeto (producido por el compilador, no se ha enlazado todavía)
archivo.pdf	Archivo en Formato de Documento Portable
archivo.ps	Archivo de PostScript
archivo.tex	Entrada para el programa formateador TEX
archivo.txt	Archivo de texto general
archivo.zip	Archivo comprimido

Figura 4-1. Algunas extensiones de archivos comunes.

- En sistemas UNIX las extensiones de archivo son solo convenciones, ese nombre es más un recordatorio para el usuario que un medio para transportar información.
- Windows está consciente de estas extensiones y les asigna significado

Estructura de archivos

Los archivos se pueden estructurar de varias formas, entre ellas encontramos 3 principales:

- **Secuencia de bytes sin estructura**
  - El sistema operativo no sabe, ni le importa, qué hay en el archivo.
  - Todo lo que ve son bytes.
  - Cualquier significado debe ser impuesto por los programas a nivel usuario
  - Tanto UNIX como Windows utilizan esta metodología.
- **Secuencia de registros de longitud fija**
  - Cada uno tiene su propia estructura interna
  - El concepto central es la idea de que la operación de lectura devuelva un registro y la operación de escritura sobrescriba o agregue un registro
- **Árbol de registros**
  - Un archivo consiste de un árbol de registros, donde no todos son necesariamente de la misma longitud
  - Cada registro contiene un campo llave en una posición fija dentro de él mismo
  - El árbol se ordena con base en el campo llave para permitir una búsqueda rápida por una llave específica.
  - La operación básica aquí es obtener el registro con una llave específica

- Se pueden agregar nuevos registros al archivo con el sistema operativo decidiendo dónde colocarlo.

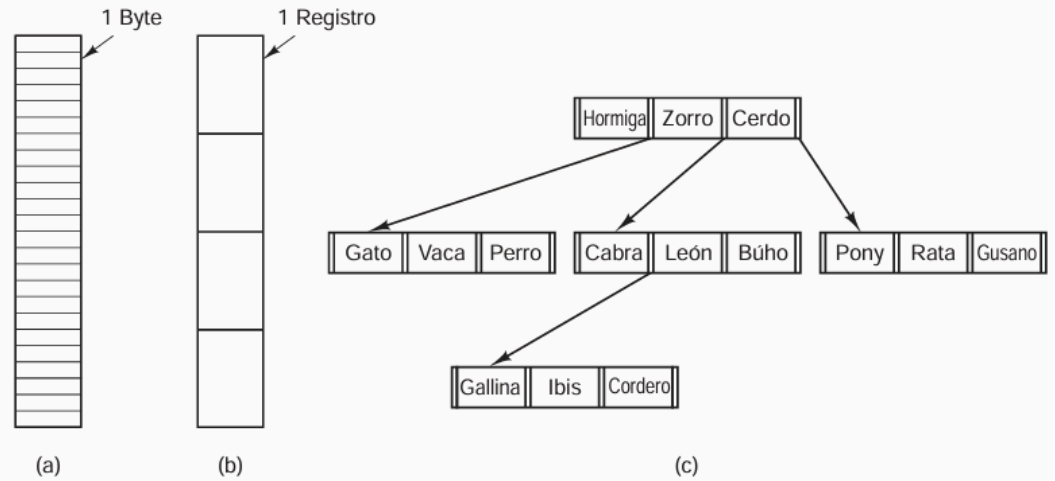

Figura 4-2. Tres tipos de archivos. (a) Secuencia de bytes. (b) Secuencia de registros. (c) Árbol.

Tipos de archivos

Muchos sistemas operativos soportan varios tipos de archivos.

- **Windows**:
  - Archivos y directorios regulares
- **UNIX**:
  - Archivos y directorios regulares
  - Archivos especiales de caracteres
  - Archivos especiales de bloque

Los **archivos regulares** son los que contienen información del usuario. Por lo general son archivos ASCII o binarios.
Los **archivos ASCII** consisten en líneas de texto, mientras que los **archivos binarios** son un listado de caracteres incomprensibles para el usuario, pero que tienen cierta estructura para los programas que lo utilizan. En este tipo el S.O solo ejecutará un archivo si tiene el formato apropiado. Este archivo tiene 5 secciones: encabezado, texto, datos, bits de reubicación y tabla de símbolos.
Los **directorios** son sistemas de archivos para mantener la estructura del sistema de archivos.
Los **archivos especiales de caracteres** se relacionan con la entrada/salida y se utilizan para modelar dispositivos de E/S en serie.
Los **archivos especiales de bloques** se utilizan para modelar discos.

Acceso a archivos

Los primeros sistemas operativos proporcionaban sólo un tipo de acceso:

- **Acceso secuencial**
  - Un proceso podía leer todos los bytes o registros en un archivo en orden, empezando desde el principio, pero no podía saltar algunos y leerlos fuera de orden.
  - Los archivos secuenciales eran convenientes cuando se utilizaba la cinta magnética

Cuando se empezó a usar discos para almacenar archivos, se hizo posible leer los bytes o registros de un archivo fuera de orden, accediendo a ellos mediante una llave en vez de posición.

- **Acceso aleatorio**
  - Permite leer en cualquier orden los bytes o registros de un archivo.
  - Es posible utilizar dos métodos para especificar dónde se debe empezar a leer: Read-Seek
  - Este método se utiliza en UNIX y Windows.

Atributos de archivos

Todo archivo tiene un nombre y sus datos. Además, todos los sistemas operativos asocian otra información con cada archivo.
A estos elementos de información adicionales se los conoce como **atributos** o **metadatos**. Varían en todos los sistemas operativos.

Atributo	Significado
Protección	Quién puede acceso al archivo y en qué forma
Contraseña	Contraseña necesaria para acceder al archivo
Creador	ID de la persona que creó el archivo
Propietario	El propietario actual
Bandera de sólo lectura	0 para lectura/escritura; 1 para sólo lectura
Bandera oculto	0 para normal; 1 para que no aparezca en los listados
Bandera del sistema	0 para archivos normales; 1 para archivo del sistema
Bandera de archivo	0 si ha sido respaldado; 1 si necesita respaldarse
Bandera ASCII/binario	0 para archivo ASCII; 1 para archivo binario
Bandera de acceso aleatorio	0 para sólo acceso secuencial; 1 para acceso aleatorio
Bandera temporal	0 para normal; 1 para eliminar archivo al salir del proceso
Banderas de bloqueo	0 para desbloqueado; distinto de cero para bloqueado
Longitud de registro	Número de bytes en un registro
Posición de la llave	Desplazamiento de la llave dentro de cada registro
Longitud de la llave	Número de bytes en el campo llave
Hora de creación	Fecha y hora en que se creó el archivo
Hora del último acceso	Fecha y hora en que se accedió al archivo por última vez
Hora de la última modificación	Fecha y hora en que se modificó por última vez el archivo
Tamaño actual	Número de bytes en el archivo
Tamaño máximo	Número de bytes hasta donde puede crecer el archivo

Relacionados con la protección del archivo e indican quién puede acceder a él y quien no.

Controlan/habilitan cierta propiedad específica.

Proporcionan información requerida para buscar las llaves.

Llevan la cuenta de cuando se creó el archivo, su acceso y modificaciones recientes.

Indican que tan grande es el archivo presente.

Figura 4-4. Algunos posibles atributos de archivos.

Operaciones de archivos

Distintos sistemas proveen diferentes operaciones para permitir el almacenamiento y la recuperación.
Por ejemplo:
Create - Delete - Open - Close - Read - Write- Append - Seek - Get attributes - Set attributes - Rename

DIRECTORIOS

Los directorios o carpetas son los que permiten llevar el registro de los archivos.

Sistemas de directorios de un solo nivel

La forma más simple de un sistema de directorios es tener un directorio que contenga todos los archivos, este directorio es llamado **directorio raíz**.
Las **ventajas** de este esquema son su simpleza y la habilidad de localizar archivos con rapidez

Sistemas de directorios jerárquicos

Son una forma de organizar y estructurar la información en sistemas de archivos. Consiste en una jerarquía (árbol) de directorios, donde el directorio raíz es el nivel superior del árbol y contiene todos los demás directorios.

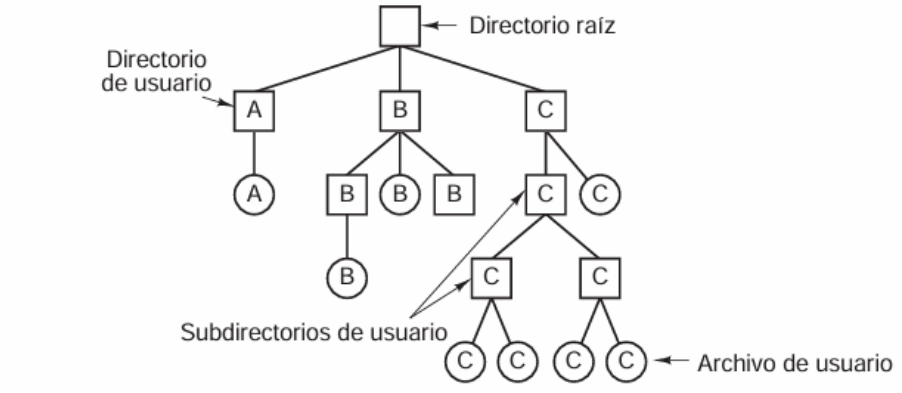

Figura 4-7. Un sistema de directorios jerárquico.

Nombres de rutas

Cuando el sistema de archivos está organizado como un árbol de directorios, se necesita cierta forma de especificar los nombres de los archivos.
Se utilizan 2 métodos distintos:

- **Nombre de ruta absoluto**:
  - Consiste en la ruta desde el directorio raíz al archivo.
  - Si el primer carácter del nombre de la ruta es el separador, entonces la ruta es absoluta.
  - En UNIX, los componentes de la ruta van separados por /
  - En Windows el separador es \
  - En MULTICS era >
- **Nombre de ruta relativa**:
  - Se utiliza en conjunto con el concepto del **directorio de trabajo** (también llamado directorio actual)
  - Un usuario puede designar un directorio como el directorio de trabajo actual, en cuyo caso todos los nombres de las rutas que no empiecen en el directorio raíz se toman en forma relativa al directorio de trabajo.
  - Dos entradas especiales en cada directorio
    - . (Punto): Se refiere al directorio actual
    - .. (PuntoPunto): Se refiere a su padre (En el raíz será él mismo)

Operaciones de directorios

Las llamadas al sistema permitidas para administrar directorios exhiben más variación de un sistema a otro que las llamadas al sistema para los archivos.

Ejemplos:
Create - Delete - Opendir - Closedir - Readdir - Rename - Link - Unlink - etc
Sobre la idea de vincular archivos, encontramos dos tipos de vínculos o links:

- **Hard link (enlace duro)**, son los que apuntan directamente al inodo del archivo en el sistema de archivos. Incrementan el contador en el nodo-i del archivo.
- **Symbolic/soft link (enlace simbólico)**, son los que llamamos como acceso directo, actúa como atajo o referencia a otro archivo o directorio.

IMPLEMENTACIÓN DE SISTEMAS DE ARCHIVOS

Los implementadores están interesados en la forma en que se almacenan los archivos y directorios, cómo se administra el espacio en el disco y cómo hacer que todo funcione con eficiencia y confiabilidad.

Distribución del sistema de archivos

Los sistemas de archivos se almacenan en discos. Estos discos se pueden dividir en particiones, **cada partición contiene un sistema de archivos independiente**.
El sector 0 del disco se conoce como el MBR y se utiliza para arrancar la computadora. El final del MBR contiene la tabla de particiones, la cual proporciona las direcciones de inicio y fin de cada partición.

Debemos aclarar que la distribución de una partición de disco varía mucho de un sistema de archivos a otro, pero a modo de ejemplo usaremos el siguiente:

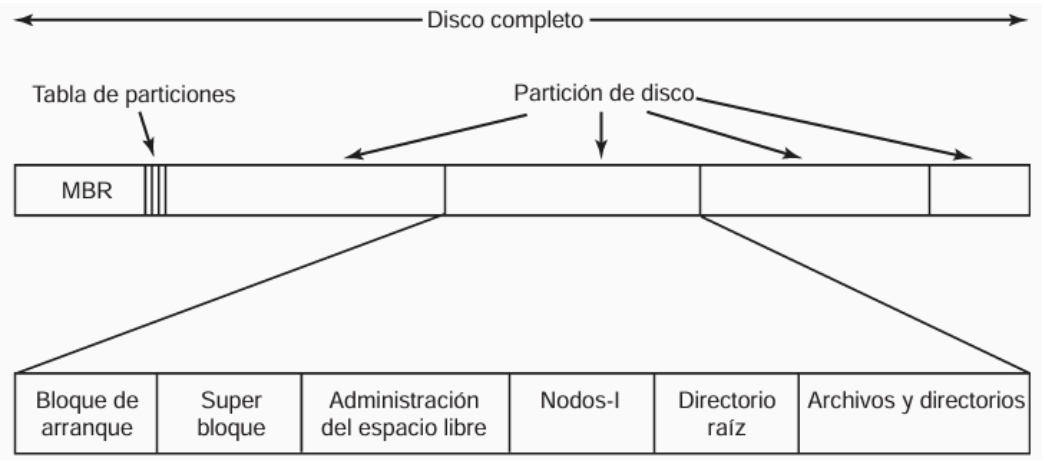

Figura 4-9. Una posible distribución del sistema de archivos.

Superbloque: Contiene todos los parámetros clave acerca del sistema de archivos y se lee en la memoria cuando se arranca la computadora o se entra en contacto con el sistema de archivos por primera vez. Incluye un número mágico para identificar el tipo del sistema de archivos, el número de bloques que contiene el sistema de archivos y otra información administrativa clave.
Administración del espacio libre: Información acerca de los bloques libres en el sistema de archivos.
Nodos-i: Un arreglo de estructuras de datos, uno por archivo, que indica todo acerca del archivo.
Directorio raíz: Contiene la parte superior del árbol del sistema de archivos.
Archivos y directorios: Contiene todos los otros directorios y archivos.

Implementación de archivos

La cuestión más importante al implementar el almacenamiento de archivos es mantener un registro acerca de qué bloques de disco van con cuál archivo.
Se utilizan varios métodos en distintos sistemas operativos:

❖ Asignación contigua

El esquema de asignación más simple es almacenar cada archivo como una serie contigua de bloques de disco.

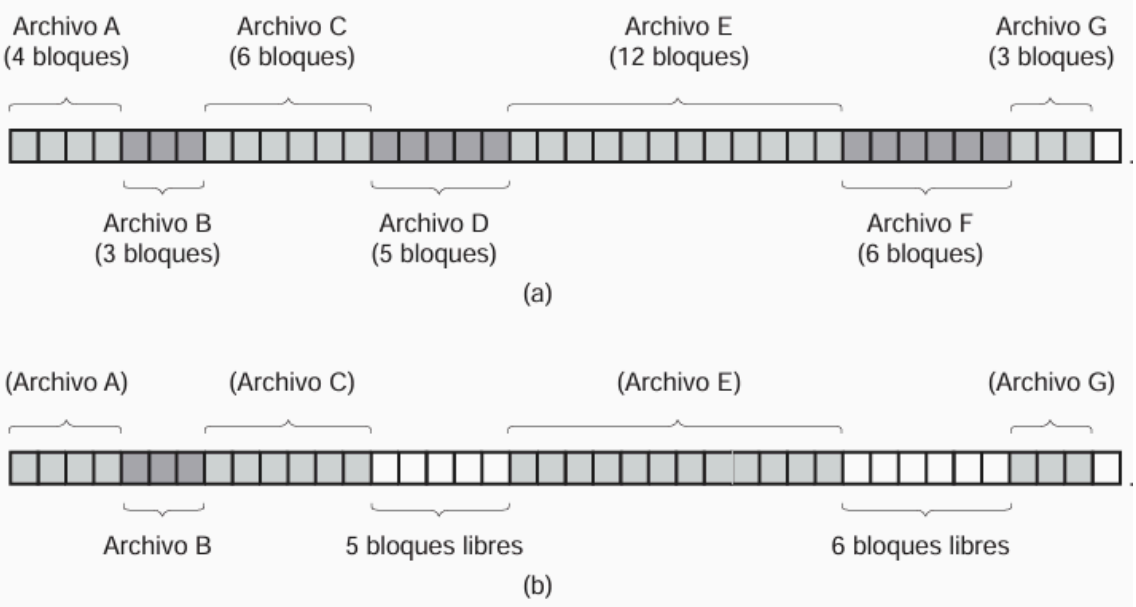

Figura 4-10. (a) Asignación contigua de espacio de disco para siete archivos. (b) El estado del disco después de haber removido los archivos *D* y *F*.

Tiene 2 **ventajas**:
Es **simple de implementar**, ya que la ubicación de los bloques de un archivo se reduce a recordar dos números: la dirección de disco del primer bloque y el número de bloques en el archivo.
El **rendimiento** de lectura es excelente debido a que el archivo completo se puede leer del disco en una sola operación.
Aunque, también tiene una gran **desventaja**: Con el transcurso del tiempo, los discos se **fragmentan**.

Sin embargo, hay una situación en la que es factible la asignación contigua y se utiliza ampliamente: en los CD-ROMs, ya que los tamaños de los archivos se conocen de antemano y nunca cambiarán durante el uso del sistema de archivos

❖ Asignación de lista enlazada (ligada)

Consiste en mantener cada archivo como una lista enlazada de bloques de disco.
La primera palabra de cada bloque se utiliza como apuntador al siguiente. El resto del bloque es para los datos.

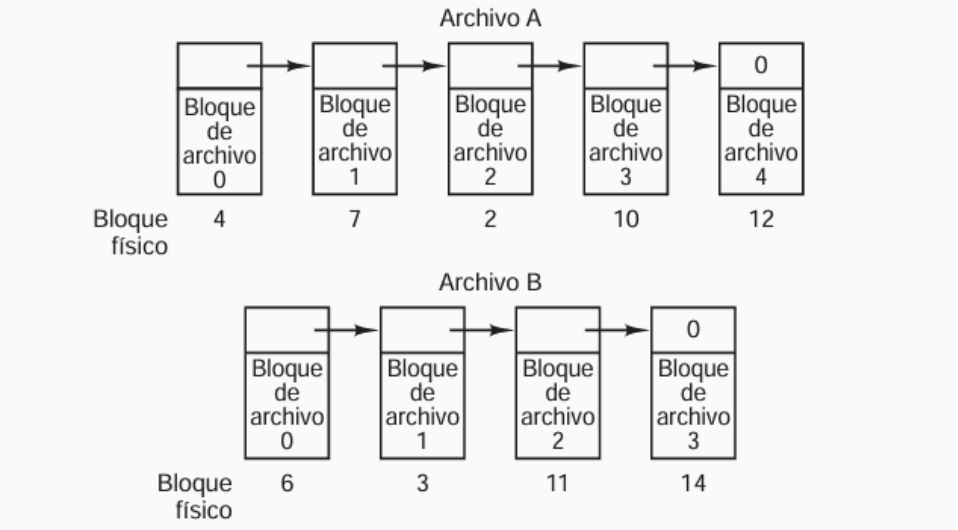

Figura 4-11. Almacenamiento de un archivo como una lista enlazada de bloques de disco.

A diferencia de la asignación contigua, en este método no se pierde espacio debido a la fragmentación del disco (Excepto el último bloque).
Sin embargo, tiene 2 **desventajas**:

- El acceso aleatorio es en extremo lento.
- La cantidad de almacenamiento de datos en un bloque ya no es una potencia de dos, debido a que el apuntador ocupa unos cuantos bytes

- **Asignación de lista enlazada utilizando una tabla en memoria**

Ambas desventajas de la asignación de lista enlazada se pueden eliminar si tomamos la palabra del apuntador de cada bloque de disco y la colocamos en una tabla en memoria principal. Dicha se conoce como FAT.

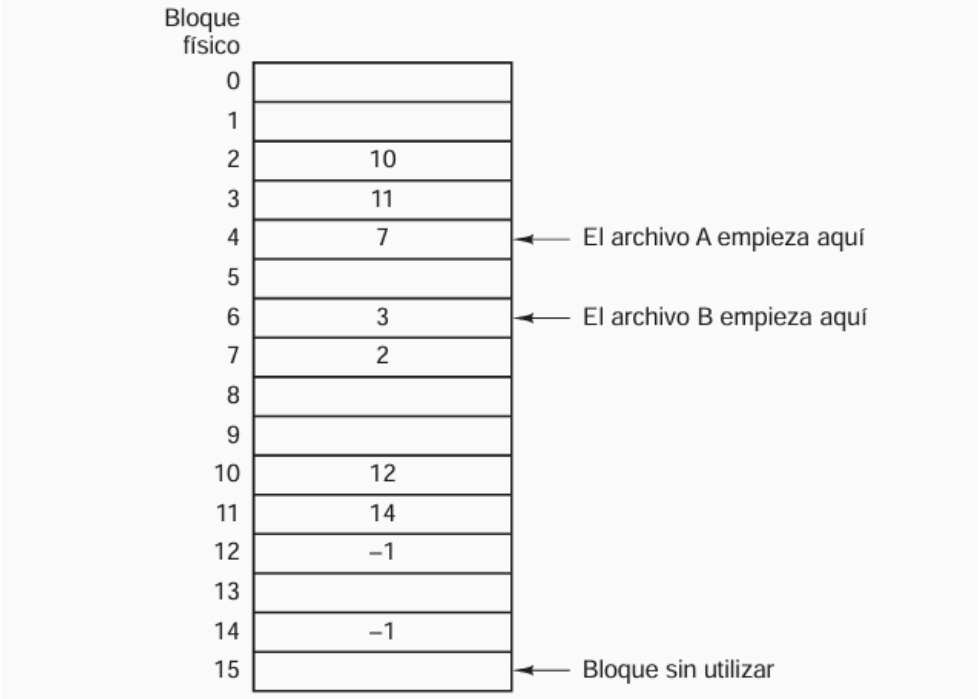

Figura 4-12. Asignación de lista enlazada que utiliza una tabla de asignación de archivos en la memoria principal.

Utilizando esta organización, el bloque completo está disponible para los datos.

La principal **desventaja** de este método es que toda la tabla debe estar en memoria todo el tiempo para que funcione. Además, no se escala muy bien en los discos grandes.

- ❖ **Nodos-i**

Consiste en asociar con cada archivo una **estructura de datos** conocida como **nodo-i** (nodo-índice), la cual **lista los atributos y las direcciones de disco de los bloques del archivo**.
La gran **ventaja** es que el nodo-i necesita estar en memoria sólo cuando está abierto el archivo correspondiente.
Un problema con los nodos-i es cuando un archivo crece más allá del límite de direcciones de disco del nodo.
Una solución es reservar la última dirección de disco no para un bloque de datos, sino para la dirección de un bloque que contenga más direcciones de bloques de disco.

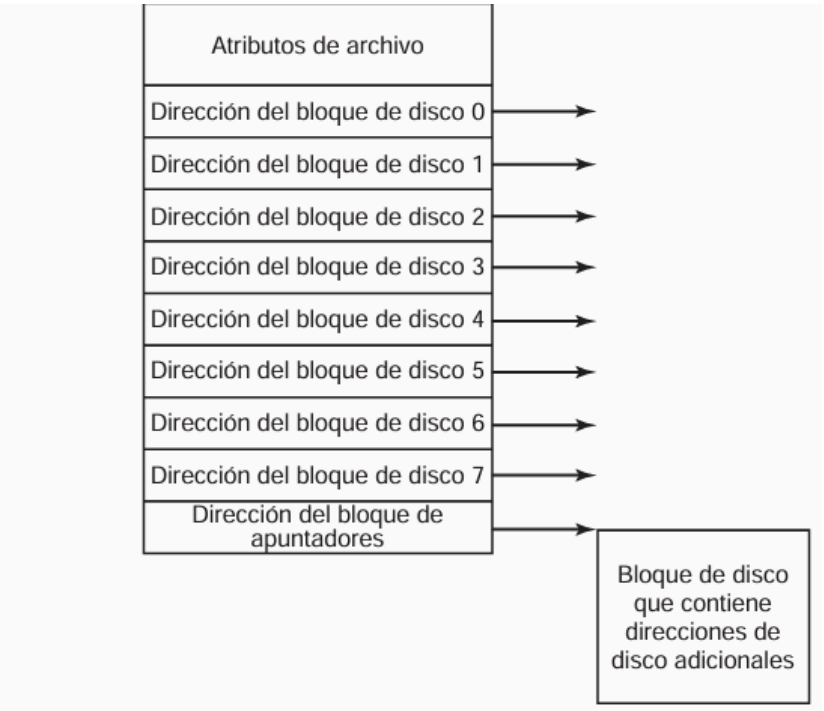

Figura 4-13. Un nodo-i de ejemplo.

Implementación de directorios

Cuando se abre un archivo, el sistema operativo utiliza el nombre de la ruta suministrado por el usuario para localizar la entrada de directorio. Esta **entrada** provee la **información** necesaria **para encontrar los bloques de disco**.
La **función principal** del sistema de directorios es asociar el nombre ASCII del archivo a la información necesaria para localizar los datos.

Una cuestión muy relacionada es dónde deben almacenarse los atributos. Encontramos 2 posibilidades principales:

- ❖ Una posibilidad es almacenarlos directamente en la entrada de directorio
- ❖ Para sistemas que utilizan nodos-i, podemos almacenar los atributos en los nodos-i. En ese caso, la entrada de directorio consiste de un nombre de archivo y un número de nodo-i.

Ambos esquemas corresponden a Windows y UNIX.

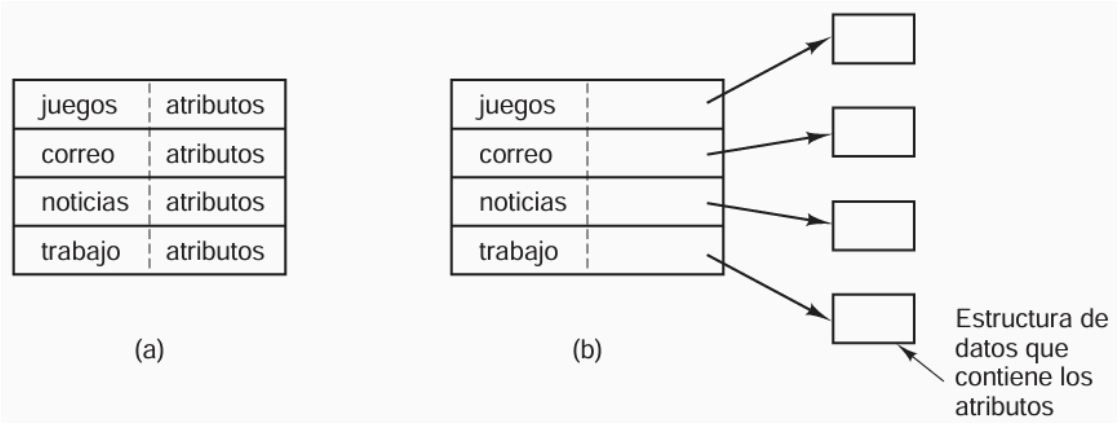

Figura 4-14. (a) Un directorio simple que contiene entradas de tamaño fijo, con las direcciones de disco y los atributos en la entrada de directorio. (b) Un directorio en el que cada entrada sólo hace referencia a un nodo-i.

Hasta ahora hemos hecho la suposición de que los archivos tienen nombres cortos con longitud fija. Pero casi todos los S.O modernos aceptan nombres de archivos más largos, con longitud variable.
Si queremos implementarlos podemos:

- Establecer un **límite en la longitud del nombre de archivo**, que por lo general es de 255 caracteres.
  Este esquema desperdicia mucho espacio de directorio, ya que pocos archivos tienen nombres tan largos.
- Una alternativa es renunciar a la idea de que todas las entradas de directorio sean del mismo tamaño.
  Cada entrada de directorio contiene una porción fija, que empieza con la longitud de la entrada y después va seguida de atributos.
  Una **desventaja** de este método es que cuando se elimina un archivo, en su lugar queda un hueco de tamaño variable dentro del directorio, dentro del cual el siguiente archivo a introducir puede que no quepa.
- Otra manera es hacer que las mismas entradas de directorio sean de longitud fija y mantener los nombres de los archivos juntos en un heap al final del directorio.
  Tiene la **ventaja** de que cuando se remueva una entrada, el siguiente archivo a introducir siempre cabrá ahí.

En todos los diseños mostrados hasta ahora se realizan búsquedas lineales en los directorios de principio a fin cuando hay que buscar el nombre de un archivo.
Una manera de **acelerar la búsqueda** es utilizar **una tabla de hash en cada directorio**.
El uso de una tabla de hash tiene la ventaja de que la búsqueda es mucho más rápida, pero la desventaja de una administración más compleja. Generalmente se usa para directorios que contienen cientos o miles de archivos.

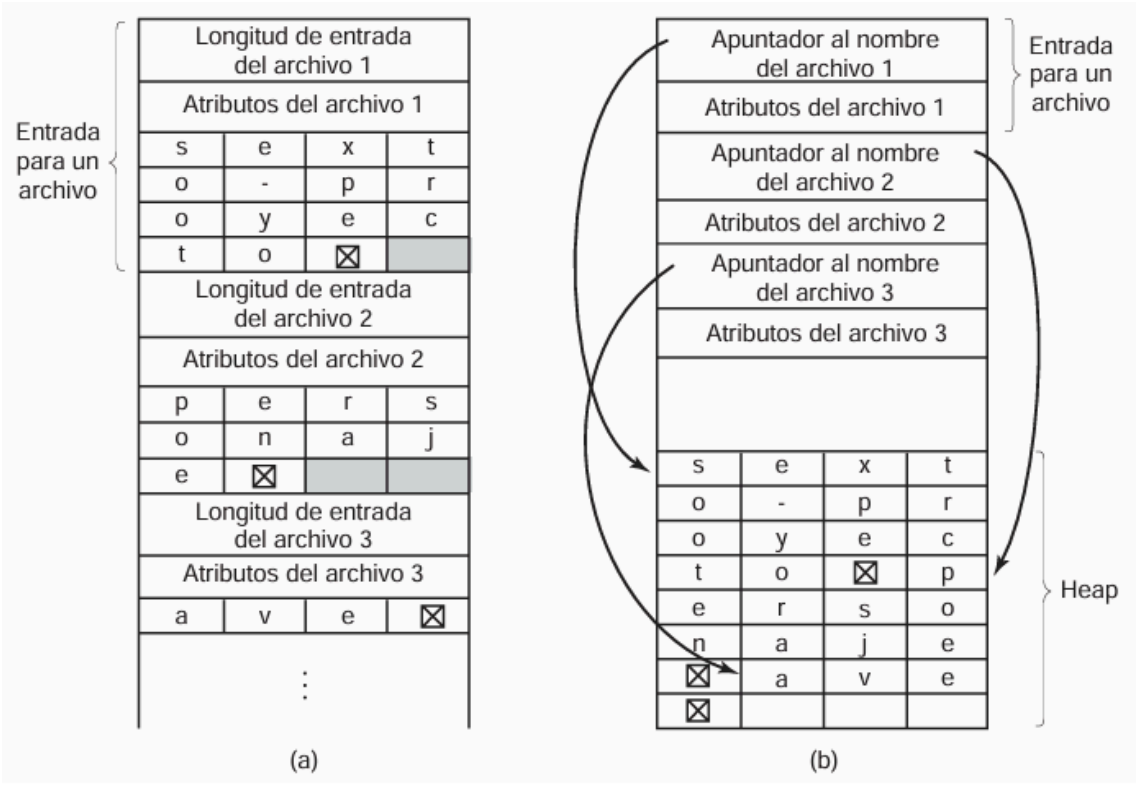

Figura 4-15. Dos maneras de manejar nombres de archivos largos en un directorio. (a) En línea. (b) En un heap.

Archivos compartidos

Cuando hay varios usuarios trabajando en conjunto en un proyecto, a menudo necesitan compartir archivos. Como resultado, con frecuencia es conveniente que aparezca un archivo compartido en forma simultánea en distintos directorios que pertenezcan a distintos usuarios.

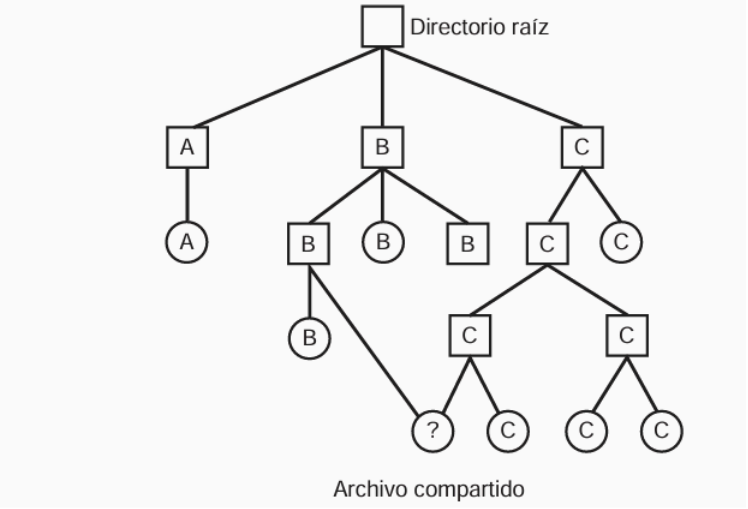

Figura 4-16. Sistema de archivos que contiene un archivo compartido.

El sistema de archivos en sí es ahora un **Gráfico acíclico dirigido (DAG)** en vez de un árbol.
La conexión entre el directorio de B y el archivo compartido se conoce como un **vínculo** (liga).

Compartir archivos es conveniente, pero también introduce ciertos problemas:

- Los directorios contienen direcciones de disco, entonces, habrá que realizar una copia de las direcciones de disco en el directorio de B cuando se ligue el archivo.

- Si B o C agregan al archivo, los nuevos bloques se listarán sólo en el directorio del usuario que agregó los datos, pero no serán visibles para el otro.

Compartir archivos entre usuarios puede ser conveniente, pero también presenta desafíos. Uno de los problemas es que si los directorios contienen referencias a bloques de disco, cuando se comparte un archivo, se deben copiar las referencias de disco al directorio del usuario receptor (por ejemplo, de B a C). Sin embargo, si B o C modifican el archivo posteriormente, los cambios no serán visibles para el otro usuario, lo que dificulta el propósito de la compartición.

Existen dos soluciones para abordar este problema:

- **Estructura de datos asociada al archivo:** En esta solución, los bloques de disco no se listan directamente en los directorios. En cambio, se utiliza una pequeña estructura de datos asociada con el archivo (como el nodo-i en UNIX). Los directorios apuntan solo a esta estructura de datos, lo que permite una mejor gestión de los cambios en el archivo.
- **Vínculo simbólico (liga simbólica):** Aquí, B se vincula a uno de los archivos de C mediante un archivo especial de tipo LINK. Este archivo de vínculo contiene solo la ruta del archivo al que está vinculado. Cuando B lee del archivo vinculado, el sistema operativo busca el nombre del archivo y accede al contenido. A diferencia del vínculo tradicional (duró), el vínculo simbólico permite una mayor flexibilidad y no requiere que los bloques de disco se copien directamente en el directorio de B.

Cada uno de estos métodos tiene sus **desventajas**:

- En el primer método, cuando comparten un mismo archivo, el sistema sabe quien es su propietario y cuantos entradas de directorio apuntan al archivo. El problema es cuando el directorio propietario elimina el archivo, ya que el otro usuario será el único con acceso al archivo, pero si el sistema realiza la contabilidad o tiene cuotas, seguirá cobrando al propietario por el archivo.
- Los vínculos simbólicos tienen un gasto adicional de procesamiento requerido. Además, se necesita un nodo-i adicional para cada vínculo simbólico, al igual que un bloque de disco adicional para almacenar la ruta.

Sistemas de archivos por bitácora

Sistemas de archivos por bitácora (JFS): Estos sistemas mantienen un registro de las operaciones que se realizarán antes de ejecutarlas. Si ocurre una falla antes de completar una operación, el sistema puede consultar el registro al reiniciar y finalizar las tareas pendientes.
Ej: NTFS de Microsoft, ext3 y ReiserFS de Linux.

Considere una operación simple que ocurre todo el tiempo: remover un archivo. Esta operación (en UNIX) requiere tres pasos:

1. Quitar el archivo de su directorio.
2. Liberar el nodo-i y pasarlo a la reserva de nodos-i libres.
3. Devolver todos los bloques de disco a la reserva de bloques de disco libres.

En la ausencia de fallas del sistema, el orden en el que se realizan estos pasos no importa; en la presencia de fallas, sí.

El sistema de archivos por bitácoras escribe una entrada de registro con las acciones pendientes antes de ejecutarlas. Solo después de escribir la entrada de registro en el disco, se realizan las operaciones. Una vez que las operaciones se completan con éxito, se borra la entrada de registro. Si ocurre una falla, el sistema verifica el registro y repite las operaciones pendientes.

Para que funcione el sistema por bitácora, las operaciones registradas deben ser **idempotentes**, lo cual significa que pueden repetirse todas las veces que sea necesario sin peligro.
Para una mayor confiabilidad, un sistema de archivos puede introducir el concepto de **transacción atómica**, es decir, el sistema de archivos sabe que debe completar todas las operaciones agrupadas o ninguna de ellas, pero ninguna otra combinación.

ADMINISTRACIÓN Y OPTIMIZACIÓN DE SISTEMAS DE ARCHIVOS

Hacer que el sistema de archivos funcione es una cosa; hacerlo que funcione de manera eficiente y robusta en la vida real es algo muy distinto.

Administración del espacio en disco

Hay **dos estrategias** generales posibles para almacenar un archivo de n bytes:

- Se asignan consecutivos de espacio en disco
- El archivo se divide en varios bloques (no necesariamente) contiguos.

La misma concesión está presente en los sistemas de administración de memoria, entre la segmentación pura y la paginación.

Tamaño de bloque

Una vez que se ha decidido almacenar archivos en bloques de tamaño fijo, surge la pregunta acerca de ¿qué tan grande debe ser el bloque?

Tener un tamaño de bloque **grande** significa que cada archivo ocupa un cilindro completo. También significa que los pequeños archivos desperdician una gran cantidad de espacio en disco.
Un tamaño de bloque **pequeño** significa que la mayoría de los archivos abarcarán varios bloques y por ende, necesitan varias búsquedas y retrasos rotacionales para leerlos, lo cual reduce el rendimiento.

En conclusión, si la unidad de asignación es demasiado grande, desperdiciamos espacio; si es demasiado pequeña, desperdiciamos tiempo.
Con un bloque de 4 KB el desperdicio de espacio al final de cada pequeño archivo no es muy importante.

Registro de bloques libres

La siguiente cuestión es cómo llevar registro de los bloques libres.
Hay dos métodos utilizados ampliamente:

- **Lista enlazada de bloques de disco**: Cada bloque contiene tantos números de bloques de disco libres como pueda.
- **Mapa de bits**. Un disco con n bloques requiere un mapa de bits con n bits.

El mapa de bits requiere menos espacio, ya que utiliza 1 bit por bloque, en comparación con 32 bits en el modelo de la lista enlazada. Los bloques libres se representan mediante 1s, y los bloques asignados mediante 0s (o viceversa).

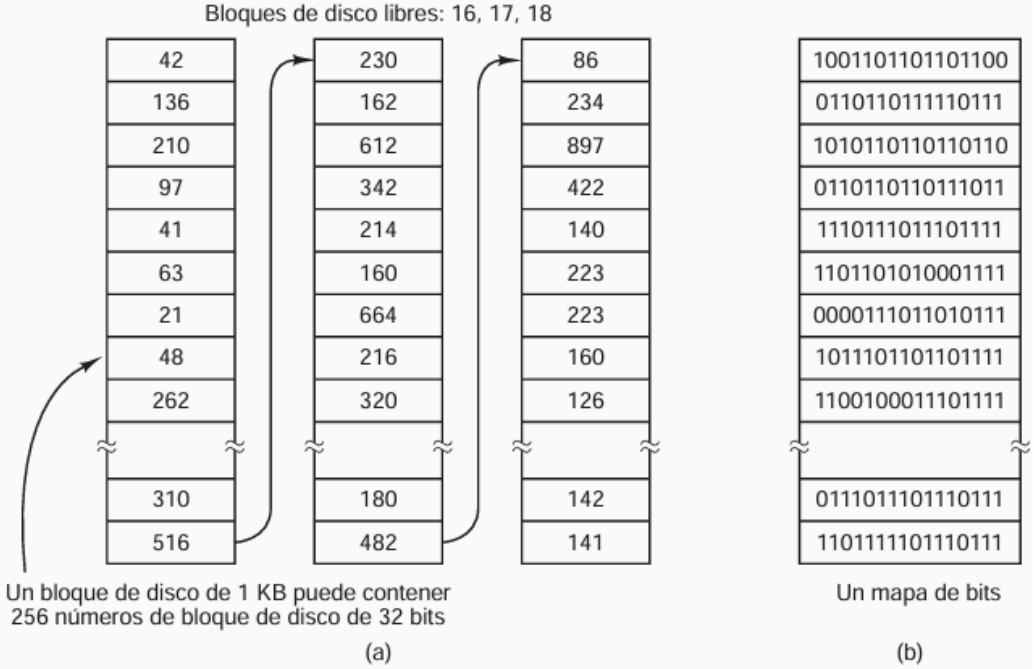

Figura 4-22. (a) Almacenamiento de la lista de bloques libres en una lista enlazada. (b) Un mapa de bits.

Cuotas de disco

Es un mecanismo que busca evitar que los usuarios ocupen demasiado espacio en disco. La idea es que el administrador del sistema asigne a cada usuario una cantidad máxima de archivos y bloques y que el sistema operativo se asegure de que los usuarios no excedan sus cuotas.

Respaldos del sistema de archivos

La destrucción de un sistema de archivos es a menudo un desastre aún mayor que la destrucción de una computadora. El sistema de archivos puede ayudar a proteger la información mediante la realización de respaldos.
Papelera de reciclaje: Directorio especial

Por lo general es conveniente respaldar sólo directorios específicos y todo lo que contengan, en vez de respaldar todo el sistema de archivos. Esto nos lleva a la idea de los vaciados incrementales:
Vaciado Incremental: La copia de seguridad incremental sólo respalda los archivos que han cambiado desde la última copia de seguridad incremental realizada.
Aunque en este esquema se minimiza el tiempo de vaciado, complica más la recuperación debido a que se tiene que restaurar el vaciado completo más reciente, seguido de todos los vaciados incrementales en orden inverso.

Se pueden usar dos estrategias para vaciar un disco en la cinta:
Vaciado Físico: Empieza en el bloque 0 del disco, escribe todos los bloques del disco en la cinta de salida en orden y se detiene cuando acaba de copiar el último. Este programa es tan simple que probablemente pueda hacerse 100% libre de errores.

Las principales **ventajas** del vaciado físico son: Simplicidad y gran velocidad
Las principales **desventajas** son: la incapacidad de omitir directorios seleccionados, incapacidad de realizar vaciados incrementales y restaurar archivos individuales a petición del usuario.
Vaciado Lógico: Empieza en uno o más directorios especificados y vacía en forma recursiva todos los archivos y directorios que se encuentran ahí que hayan sido modificados desde cierta fecha base dada. La mayoría de los sistemas UNIX utilizan este algoritmo y además, **facilita la restauración de un archivo o directorio específico a petición del usuario**.

Consistencia del sistema de archivos

Los sistemas de archivos leen bloques, los modifican y los escriben posteriormente. Si el sistema falla antes de escribir todos los bloques modificados, el sistema de archivos puede quedar en un estado **inconsistente**.
Para lidiar con el problema de los sistemas de archivos inconsistentes, la mayoría de las computadoras tienen un **programa utilitario que verifica la consistencia del sistema de archivos**. Ejemplos: **UNIX→fsck** y **Windows→scandisk**.

Se pueden realizar **dos tipos de verificaciones** de consistencia: archivos y bloques.

En la consistencia de bloques se crean dos tablas con un contador para cada bloque. La primera tabla registra cuántas veces está presente cada bloque en archivos. La segunda tabla registra cuántas veces está presente cada bloque en la lista de bloques libres.
Después el programa lee todos los nodos-i utilizando un dispositivo puro, el cual ignora la estructura de los archivos y sólo devuelve todos los bloques de disco que empiezan en 0.

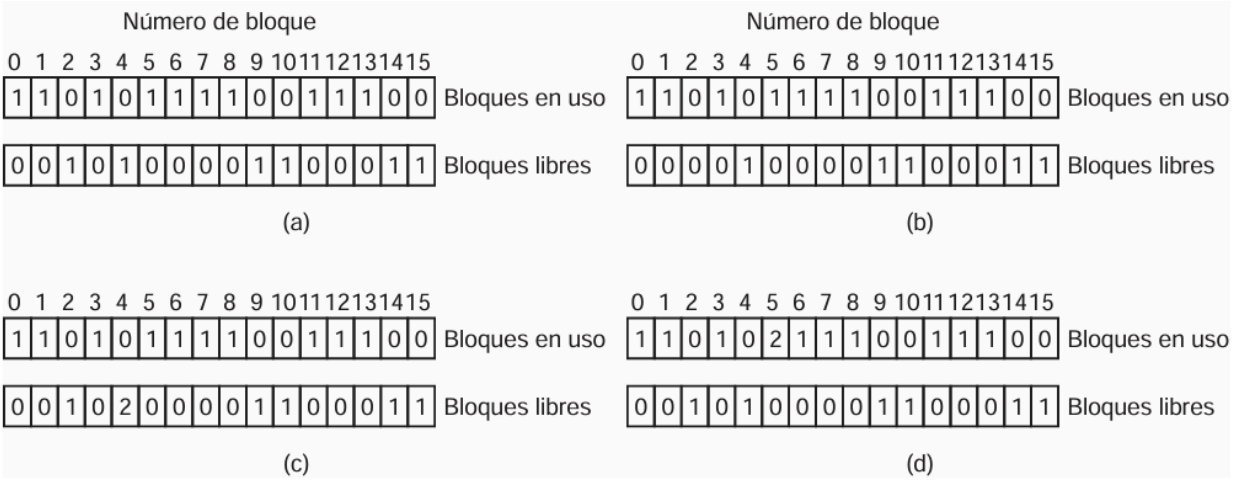

Figura 4-27. Estados del sistema de archivos. (a) Consistente. (b) Bloque faltante. (c) Bloque duplicado en la lista de bloques libres. (d) Bloque de datos duplicado.