Tema 4: Gestión de Memoria

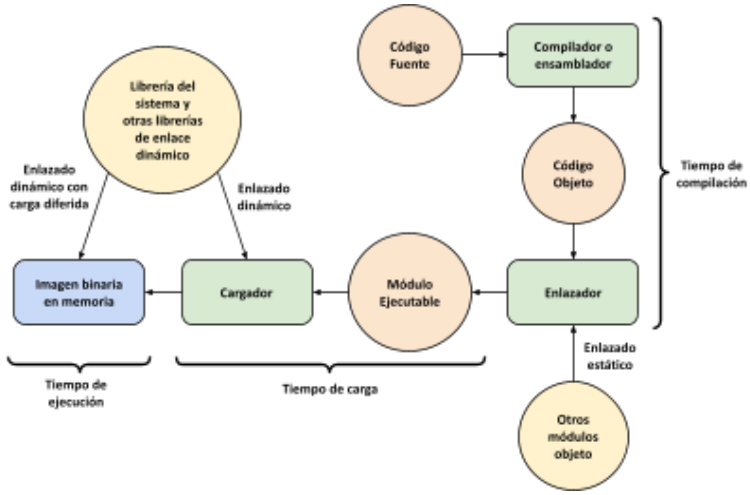
1. Memoria Principal

Es el único medio de almacenamiento al que la CPU puede acceder directamente, para ejecutar un programa, debe ser cargado desde disco a la memoria y convertido en un proceso. En los sistemas multiprogramados, la cola de entrada es aquella formada por el conjunto de procesos en disco que esperan ser cargados en la memoria para su ejecución. El procedimiento normal de ejecución es:

1. **Seleccionar un proceso de la cola y cargarlo en memoria.**
2. **Mientras se ejecuta, éste accede a instrucciones y datos de memoria.**
3. **Termina y su espacio en memoria se marca como disponible.**

En los sistemas de tiempo compartido no existe cola de entrada, por lo que los programas se cargan inmediatamente en memoria cuando se solicita su ejecución por los usuarios.

2. Reasignación de Direcciones



La mayor parte de los sistemas permiten que un proceso de usuario resida en cualquier parte de la memoria física. Un programa pasa por diferentes etapas y en cada una de ellas las direcciones pueden representarse de formas distintas. En cada paso es necesario reasignar las direcciones de una etapa en las direcciones de la siguiente.

Para que el programa se pueda ejecutar se necesita que a datos e instrucciones se les asignen direcciones absolutas de la memoria. Etapas:

1. **Tiempo de Compilación** → Se conoce el lugar de memoria donde va a ser ejecutado el proceso (Código con direcciones absolutas, o código absoluto).
2. **Tiempo de Carga** → No se conoce durante la compilación el lugar donde va a residir un programa cuando sea ejecutado (Código reubicable estático).
3. **Tiempo de ejecución** → Si un proceso puede ser movido durante su ejecución de un lugar de la memoria a otro (Código reubicable dinámico).

En los sistemas operativos modernos, los procesos no tienen acceso libre a la memoria física. El sistema operativo (usando la MMU **[Memory Management Unit]**) proporciona a cada proceso un espacio de direcciones virtuales (da una ilusión privada de la memoria). Durante los accesos a la memoria principal en tiempo de ejecución, estas direcciones virtuales son convertidas en direcciones físicas para acceder a memoria. En la asignación de direcciones:

- **Los programas pueden ser cargados en cualquier zona libre** de la memoria y movidos de una región a otra durante la ejecución de procesos (reasignación en tiempo de ejecución).
- **La reubicación de las direcciones virtuales** puede hacerse en tiempo de compilación. Lo común es que los programas se ubiquen en la parte baja del espacio de direcciones virtuales.
- **Se puede reducir el consumo de memoria principal** compartiendo las regiones de memoria física asignadas al código y los datos de solo lectura de los procesos de un mismo programa.

3. Enlazado dinámico

- **Enlazado Estático** → Las librerías del sistema y otros módulos son combinados por el enlazador para formar la imagen binaria del programa que es almacenada en disco.
- **Enlazado Dinámico** → Se pospone hasta la carga o ejecución.

Cuando el enlazado dinámico ocurre durante la carga del programa:

- Durante la carga del módulo ejecutable **se comprueban las dependencias del mismo.** Estas se almacenan en el mismo archivo en disco que dicho módulo.
- **Las librerías a enlazar se cargan y ubican en el espacio de direcciones virtuales** creado para el nuevo proceso.
- Las referencias del programa a las rutinas de cada una de las librerías cargadas se actualizan con la dirección en memoria de las mismas.

Cuando el enlazado ocurre en tiempo de ejecución (Enlazado dinámico con carga diferida):

- Durante el enlazado estático del módulo ejecutable se pone un stub a cada referencia a alguna rutina de la librería que va a ser enlazada dinámicamente.
- Si durante la ejecución alguna de dichas rutas es invocada, se ejecuta el stub.
- El stub se sustituye a sí mismo con la dirección de la rutina y la invoca. Esto permite que la siguiente ejecución de la rutina no incurra en ningún coste adicional.

Durante la compilación de una librería no se conoce la región que va a ocupar dentro de los espacios de direcciones virtuales de los distintos procesos que la van a utilizar, por tanto, el compilador debe generar código PIC (independiente de la posición). Este tipo de código se puede ejecutar independientemente del lugar de la memoria donde esté ubicado. En los sistemas donde no se usa código PIC, el compilador debe generar código reubicable para que la reubicación de las direcciones virtuales de librerías dinámicas se haga en tiempo de carga.

4. Librerías Compartidas

Las librerías incluyen información acerca de la versión que puede ser utilizada para evitar que los programas se ejecuten con versiones incompatibles de las mismas, o para permitir que haya más de una versión de cada librería en memoria. Los programas antiguos funcionan con versiones anteriores o con versiones actualizadas pero compatibles y programas nuevos pueden ejecutarse con versiones más recientes pero son incompatibles con los antiguos.

5. Asignación Contigua de Memoria

A cada proceso se le asigna una única sección de memoria contigua, mediante particionado fijo o dinámico. En el particionado fijo la memoria se divide en varias secciones de tamaño fijo en las que cada una contiene un proceso, mientras que en el dinámico:

- **El sistema operativo mantiene una tabla** que indica las partes libres y ocupadas.
- Cuando un proceso llega a memoria **se le asigna un hueco lo suficientemente grande para alojarlo**, solo se asigna el espacio necesario, que se marca como ocupado y el sobrante sigue estando libre.
- Si un proceso termina y se crean **dos huecos adyacentes, se fusionan**.

Para encontrar huecos libres hay varias soluciones:

- **Primer Ajuste:** Se escoge el primer hueco lo suficientemente grande como para satisfacer la petición. Se busca desde el principio de la lista o desde donde ha terminado la búsqueda anterior. **(MEJOR TIEMPO, MEJOR APROVECHAMIENTO)**
- **Mejor Ajuste:** Se escoge el hueco más pequeño que sea lo suficientemente grande para satisfacer la petición. Hay que recorrer la lista de huecos completa o tenerla ordenada por tamaño. **(BUEN TIEMPO, MEJOR APROVECHAMIENTO)**
- **Peor Ajuste:** Se escoge el hueco más grande. Igual que el mejor ajuste pero el orden de tamaños es inverso. **(PEOR TIEMPO, PEOR APROVECHAMIENTO)**

6. Fragmentación Externa

Ocurre cuando hay suficiente espacio libre para satisfacer una petición, pero el espacio no es contiguo (está fraccionado en muchos huecos pequeños). Soluciones:

- **Utilizar técnicas de compactación** (mover procesos para que quede un gran hueco). Es caro en términos de tiempo y solo se puede cuando la asignación de direcciones absolutas se realiza en tiempo de ejecución.
- **Permitir que el espacio físico de direcciones de un proceso no sea contiguo**. Existen dos técnicas: La paginación y la segmentación.

7. Fragmentación Interna

Se origina por la diferencia entre el espacio solicitado y el espacio finalmente asignado. La solución común es dividir la memoria física en unidades de tamaño fijo y asignarla en múltiplos de tamaño de dichos bloques, esto hace que en general se asigne más memoria de la que realmente se ha solicitado y de la que se va a usar. A esto se denomina fragmentación interna.

8. Intercambio

Un proceso debe estar en memoria para ser ejecutado, pero en algunos sistemas un proceso puede ser sacado de la memoria y copiado a un almacenamiento de respaldo de forma temporal para eventualmente volver a ser traído a la memoria y continuar su ejecución.

- La **cola de preparados** contiene los procesos que esperan ser ejecutados por la CPU.
- Cuando el planificador de la CPU decide ejecutar un proceso, **llama al asignador**.
- El **asignador comprueba** si el siguiente proceso está en memoria. Si no lo está y no hay memoria libre, el asignador hace que el **gestor de memoria intercambie el proceso con alguno** de los que sí lo está.
- El asignador ejecuta el resto del cambio de contexto para entregar la CPU al proceso.

Limitaciones:

- Si un sistema reubica las direcciones **en tiempo de compilación o carga, el proceso sólo puede ser intercambiado en la misma región de memoria**. Sin embargo, si se reubica **en tiempo de ejecución, el proceso puede ser intercambiado en cualquier región de la memoria**, ya que las direcciones físicas son calculadas en la ejecución.

- **El tiempo de cambio de contexto en un sistema con intercambio puede ser mucho mayor**, ya que incluye el tiempo que se tarda en hacer el intercambio.

También presenta dificultades cuando el proceso que va a ser sacado de memoria está esperando por una operación de E/S. Las soluciones podrían ser:

- **No intercambiar procesos con operaciones E/S pendientes.**
- **Utilizar búferes del sistema en las operaciones E/S.**

Debido a que el tiempo de intercambio es muy alto, no se utiliza el intercambio estándar en los sistemas actuales, pero se pueden encontrar versiones modificadas de este mecanismo.

9. Paginación

La memoria física se divide en bloques de tamaño fijo denominados marcos, mientras que el espacio de direcciones virtual se divide en bloques del mismo tamaño que los marcos, denominados páginas. Cuando un proceso va a ser ejecutado sus páginas son cargadas desde disco en marcos libres de la memoria física. El tamaño de marcos viene definido por el hardware y normalmente es una potencia de 2 que varía entre 512 bytes y 16 MiB.

Cada dirección virtual generada por la CPU es dividida en dos partes: un **número de página** P y un **desplazamiento** D. El número de página es utilizado por la MMU para indexar la tabla de páginas, que contiene el número del marco ***f*** de cada página en la memoria física. El número de marco es combinado con el desplazamiento para generar la dirección física que será enviada por el bus de direcciones hacia la memoria.

Desde el punto de vista de los procesos:

- Cada página de un proceso requiere un marco.
- Si el proceso requiere **n** páginas, el sistema operativo debe escoger **n** marcos, estos marcos son tomados de la lista de marcos libres que mantiene el sistema. **El espacio de direcciones físico puede no ser contiguo** aunque los procesos vean un espacio de direcciones virtual contiguo.
- Los **marcos seleccionados son asignados al proceso y cada página del proceso es cargada en uno de dichos marcos**.
- La tabla de páginas es actualizada de manera en que **en la entrada de cada página se pone el número de marco correspondiente**.

Desde el punto de vista del sistema operativo:

- Se encarga de gestionar la memoria física, así que **éste debe mantenerse al tanto de las particularidades de su uso** (Marcos asignados, y a que página de qué proceso o procesos lo están; y marcos disponibles).
- Toda esta información se guarda en la **tabla de marcos**, que tiene una entrada por cada marco de la memoria física.

El sistema mantiene una copia de la TP para cada proceso en el PCB. Esta copia es utilizada:

- Por el asignador para **sustituir la tabla de páginas hardware cuando realiza un cambio de contexto** (paginación incrementa el tiempo de cambio de contexto).
- Para el **mapeo manual de direcciones virtuales a físicas**. Cuando un proceso realiza una llamada al sistema para una operación E/S y proporciona una dirección, esa dirección debe ser mapeada para producir la dirección física que será utilizada por el hardware para realizar la operación.

Tamaño de páginas:

- **Páginas pequeñas** → Menos fragmentación interna, más entradas en la TP, peor E/S.
- **Páginas grandes** → Más fragmentación interna, menos entradas en la TP, mejor E/S.

Soporte hardware de la TP: La implementación en hardware de la tabla de páginas puede realizarse de diversas maneras:

- **Como conjunto de registros dedicados de la CPU** → Más eficiente pero solo se puede para tablas de páginas pequeñas.
- **Almacenada en memoria** → Esto permite disponer tablas de páginas de gran tamaño aunque se necesitan dos accesos a memoria física (Solución: TLB).

Para que la MMU pueda conocer la ubicación de la tabla de páginas, la CPU debe disponer de un registro (PTBR) donde se guarda la dirección de la tabla de páginas actual.

TLB: Es una memoria asociativa de alta velocidad que contiene unas pocas entradas de la tabla de páginas. Cuando la CPU genera una dirección virtual, el número de página es entregado a la TLB, que utiliza estos números como clave, por lo que sí hay coincidencia el número de marco es extraído de la entrada devuelta por la TLB y es utilizado para generar la dirección física. En caso contrario, es necesario acceder a la tabla de páginas para obtener la entrada correspondiente directamente de ella y la entrada recuperada añadida a la TLB.

Cuando el sistema realiza un cambio de contexto es necesario que el asignador realice un borrado de la TLB, de lo contrario el nuevo proceso podría utilizar las entradas de la tabla de páginas del viejo proceso. El borrado se puede evitar si cada entrada de la TLB tiene un ASID (*Address-Space Identification*).

10. Tiempo de Acceso a Memoria

El tiempo de acceso efectivo a memoria $\mathbf{T_{EM}}$ intenta estimar el tiempo que se tarda en acceder a memoria, teniendo en cuenta mecanismos como la paginación o el uso de TLB. $\mathbf{T_{EM}=2T_M}$, donde $\mathbf{T_M}$ es el tiempo de acceso a la memoria física (normalmente en orden nano). En un sistema con TLB, si **P** es la probabilidad de que la entrada esté en la TLB, entonces, el tiempo de acceso efectivo se podría calcular así: $\mathbf{T_{EM} = (1 - P) \cdot (2 \cdot T_M + P \cdot T_M) = (2 - P) \cdot T_M}$**.** Donde (1 - P) es la probabilidad de que la entrada no esté en la TLB.

11. Protección

La protección de las páginas se hace mediante unos bits de protección asociados a cada entrada de la tabla de páginas, que pueden indicar sólo **lectura, lectura - escritura y sólo ejecución**. La MMU comprueba que el tipo de acceso es válido, sino, se genera una excepción de violación de protección de memoria. Además se suele añadir a cada entrada un bit de validez:

- Cuando una **página es válida**, la página asociada está en el espacio de direcciones virtual del proceso **(Es legal)**.
- Cuando una **página no es válida**, la página no está asociada al espacio de direcciones virtual del proceso **(No es legal)**.

En general los procesos solo necesitan una porción muy pequeña de su espacio de direcciones virtual. Para no tener que almacenar una tabla de página completa con una entrada para cada página del espacio de direcciones existe el **PTLR (Page-Table Length Reg)** que se utiliza para indicar el tamaño actual de la tabla de página. Este valor es comprobado por la MMU de manera que las páginas con entradas que superan la última almacenada en la tabla son consideradas ilegales.

12. Páginas Compartidas

Una de las ventajas de la paginación es la posibilidad de **compartir páginas entre procesos**, para ello las **páginas compartidas de distintos procesos tienen asignadas un mismo marco**. Los procesos de un mismo programa pueden compartir páginas de código o datos de solo lectura **para ahorrar memoria**, además, **se pueden compartir páginas de código de una librería compartida** enlazada a diferentes procesos.

13. Paginación Jerárquica

Al método básico de paginación se le conoce como **tabla de páginas lineal**, la tabla de páginas de un proceso debe ser alojada en una **región contigua del espacio de direcciones físico**, por lo que podría darse el caso de que en algún momento no hubiera un hueco contiguo lo suficientemente grande. **La solución sería partir la tabla de páginas**, de manera que no sea necesario asignarle memoria de forma contigua.

La paginación jerárquica se basa en la idea de que un vector de gran tamaño puede ser mapeado en uno más pequeño, y así sucesivamente. Resolver una dirección virtual necesita tantos accesos a memoria como niveles hay en la jerarquía.

14. Memoria Virtual

Es una técnica que permite la **ejecución de procesos sin que estos tengan que ser cargados completamente** en la memoria. Un proceso cargado parcialmente proporciona beneficios:

- Un programa **no estará limitado por la cantidad de memoria** disponible.
- **Más programas se pueden ejecutar al mismo tiempo**, con su incremento en el uso de la CPU y sin efectos negativos en el tiempo de respuesta ni el de ejecución.

15. Paginación Bajo Demanda

Es la técnica con la que frecuentemente se implementa la memoria virtual en los sistemas de paginación, de esta forma se evita leer y colocar en la memoria páginas no usadas, reduciendo el tiempo de intercambio y la cantidad de memoria física requerida.

Las páginas individuales pueden ser sacadas de la memoria de manera temporal y copiadas a un disco de respaldo y serán traídas a la memoria cuando son necesitadas por su proceso. Esto se denomina intercambio o swapping y es llevado a cabo por el paginador.

Se incorpora un bit de válido a la entrada de cada página en la tabla:

- **Cuando el bit está a 1** → La página es legal y está en memoria.
- **Cuando el bit está a 0** → Pueden ocurrir varias cosas:
    - La página es legal pero está almacenada en disco y no en memoria.
    - La página no es legal y no existe en el espacio de direcciones virtual.

Si un proceso accede a una página residente en memoria (marcada como válida) no ocurre nada y la instrucción se ejecuta con normalidad. Mientras que si se accede a una página marcada como inválida:

1. Al intentar acceder a la página la **MMU comprueba el bit de válido** y genera una **excepción de fallo de página** al estar marcada como inválida (capturada por el SO).
2. El sistema **comprueba en una tabla interna si la página es legal o no**. Esta tabla suele almacenarse en el PCB del proceso.
3. Si la página es **ilegal, el proceso ha cometido un error** y debe terminar.
4. Si la página es **legal, debe ser cargada desde disco**.

Secuencia de carga desde disco:

1. El núcleo debe **buscar un marco de memoria libre** (Lista de marcos libres).
2. Se solicita una **operación de disco para leer la página** deseada en el marco asignado (poner al proceso en estado de espera para ser más eficiente).
3. Cuando la lectura del disco haya terminado se debe **modificar la tabla interna y la tabla de páginas** para indicar que la página está en memoria.
4. **Reiniciar la instrucción que fue interrumpida** por la excepción (colocando el proceso en la cola de preparados y dejando que el asignador lo reinicie cuando sea escogido).

Un caso extremo es la paginación bajo demanda pura, donde la ejecución de un proceso se inicia sin cargar ninguna página, sus principales ventajas son:

- **Nunca se traerá del disco una página que no sea necesaria**.
- El **inicio de la ejecución es mucho más rápido** que si se cargara todo el proceso desde el principio.

Requerimientos de la paginación bajo demanda: Los requerimientos hardware para que un sistema operativo pueda soportar la paginación bajo demanda son:

- **Tabla de páginas** → Con habilidad para marcar entradas inválidas, ya sea utilizando un bit específico o con valores especiales en los bits de protección.
- **Disponibilidad del disco** → Aquí se guardan las páginas que no están presentes en la memoria. Normalmente se trata de un disco conocido como dispositivo de intercambio, mientras que la sección de disco utilizada concretamente para dicho propósito se le conoce como espacio de intercambio o swap.
- **Posibilidad de reiniciar cualquier instrucción después de un fallo de página**.

Tiempo de acceso efectivo: El rendimiento de estos sistemas se ve afectado por el número de fallos de páginas. El tiempo de acceso efectivo intenta estimar el tiempo que realmente se tarda en acceder a memoria teniendo en cuenta mecanismos como la paginación bajo demanda: $\mathbf{T_{EM} = (1 - P) \cdot T_M + P \cdot T_{FP}}$. El tiempo de fallo de página $\mathbf{T_{FP}}$ se compone de:

- **La excepción de fallo de página** → Capturar la interrupción, salvar los registros y el estado de proceso, determinar que la interrupción es debida a una excepción de fallo de página, comprobar si la página es legal y determinar su localización en disco.
- **Leer la página en un marco libre.**
- **Reiniciar el proceso.**

Si estamos en un sistema que utiliza paginación y que puede tener una TLB para mejorar su rendimiento podemos considerar $\mathbf{T_M}$ como el tiempo de acceso: $\mathbf{T_{EM} \approx (2 - PTLB) \cdot T_M + T_{FP}}$. Donde **PTLB** es la probabilidad de que una entrada esté en la TLB y $\mathbf{T_M}$ es el tiempo de acceso a la memoria física.

16. Espacio de Intercambio

Cuando un proceso genera un fallo de página, el sistema debe recuperar la página de donde esté almacenada. Si ocurre al principio de la ejecución, ese lugar seguramente será el archivo que contiene la imagen binaria del programa, pues es donde se encuentran las páginas en su estado inicial. El acceso al espacio de intercambio es mucho más eficiente que el acceso a un sistema de archivos, ya que los datos se organizan en bloques contiguos de gran tamaño, que evitan búsquedas de archivos y métodos indirectos de asignación de espacio.

Se puede mejorar el rendimiento copiando en el espacio de intercambio la imagen completa de los programas al inicio del proceso. Se cargan las páginas desde el archivo que contiene la imagen cuando son usadas por primera vez pero siendo escritas en el espacio de intercambio cuando dichas páginas tienen que ser reemplazadas. Esto garantiza que solo las páginas necesarias sean leídas del sistema de archivos reduciendo el uso de espacio de intercambio.

Si se supone que el código de los procesos no puede cambiar, se puede utilizar el archivo de la imagen binaria para recargas las páginas de código, lo que también evita escribirlas cuando son sustituidas, este método parece conseguir un buen compromiso entre el tamaño del espacio de intercambio y el rendimiento. Se utiliza en gran parte de los sistemas modernos.

17. Copy-On-Write

Permite la creación rápida de nuevos procesos, minimizando la cantidad de páginas asignadas a estos, además, permite ahorrar memoria y tiempo en la creación, puesto que solo se copian las páginas que son modificadas por estos y minimiza el número de marcos que deben ser asignadas al nuevo proceso **(fork() → el hijo comparte páginas con el padre)**.

Las páginas compartidas se marcan como copy-on-write, para ello se pueden marcar todas las páginas como de sólo lectura en la tabla de páginas de ambos procesos y utilizar una tabla interna en el PCB para indicar cuales son realmente solo lectura o copy-on-write. Si algún proceso intenta escribir en una página copy-on-write, la MMU genera una excepción para notificar el suceso al sistema operativo.

Si el sistema detecta una escritura a una página del copy-on-write solo tiene que copiarla en un marco libre y mapear en el espacio de direcciones del proceso, la página original marcada como copy-on-write puede ser marcada como de escritura en vez de copy-on-write, pero solo si ya no va a seguir siendo compartida. El sistema operativo puede reiniciar el proceso, y a partir de ahora éste puede escribir en la página sin afectar al resto de los procesos, sin embargo puede seguir compartiendo otras páginas en copy-on-write.

18. Archivos Mapeados en Memoria

Permiten acceder a un archivo como parte del espacio de direcciones virtuales de un proceso. Características:

- Cuando una región del espacio de direcciones se marca para ser mapeada sobre una región de un archivo, **las páginas son cargadas desde dicho archivo y no desde el espacio de intercambio**, es decir, en un primer acceso a una página mapeada se produce un fallo de página que es resuelto por el sistema leyendo una porción del archivo en el marco asignado a la página.
- **La lectura y escritura del archivo se realiza a través de operaciones en memoria**, lo que simplifica el acceso y elimina el coste adicional de llamadas al sistema.
- **Las escrituras en disco se realizan de forma asíncrona**, para ello el sistema comprueba periódicamente las páginas modificadas y las escribe en disco.
- **Los marcos utilizados en el mapeo pueden ser compartidos**, lo que permite compartir datos de los archivos.

Algunos sistemas ofrecen servicios de mapeo en memoria a través de llamadas al sistema **read(), write()**, etc. Sin embargo, muchos sistemas modernos usan el mapeo en memoria independiente de si se piden o no. Estos sistemas tratan la E/S como mapeada en memoria.

En los sistemas POSIX, si un proceso utiliza llamada al sistema **mmap()** es porque pide que el archivo sea mapeado en memoria, el núcleo mapea el archivo en el espacio de direcciones del proceso. En sistemas modernos, cuando un archivo es abierto con llamadas estándar (open) el archivo es mapeado en el espacio de direcciones del núcleo y las llamadas read y write son traducidas en accesos a memoria de dicha región.

19. Reemplazo de Página

Si no hay marcos libres y la página es legal, será cargada siguiendo el procedimiento:

1. **Buscar la localización de la página en disco.**
2. **El núcleo debe buscar un marco de memoria libre.**
   Si hay uno → Usarlo || Si no hay → Usar un algoritmo de reemplazo de página. Escribir la víctima en disco y cambiar tablas de páginas y marcos libres.
3. **Se solicita operación de disco para leer la página deseada en el marco asignado.**
4. **Modificar tabla de páginas válidas y la TP indicando que está en memoria.**
5. **Reiniciar proceso interrumpido.**

En caso de reemplazo se necesitan realizar dos accesos a disco. Esto se puede evitar utilizando un bit de modificado asociado a cada página en la TP, este bit es puesto a 1 por hardware cuando la página es modificada. Se puede evitar escribir en disco aquellas páginas que tienen el bit a 0 cuando son seleccionadas para reemplazo (si no han sido modificadas). En general para implementar paginación bajo demanda necesitamos:

- **Un algoritmo de asignación de marcos** que se encargue de asignar los marcos.
- **Un algoritmo de reemplazo de página** para seleccionar qué página reemplazamos cuando no tenemos marcos suficientes.

20. Algoritmos de Reemplazo de Página

Algoritmo Óptimo: Consiste en seleccionar siempre la página que más se va a tardar en necesitar, pero este algoritmo es difícil de implementar, ya que no somos capaces de ver las páginas referenciadas en el futuro. Solo se puede utilizar en estudios comparativos para saber cuánto se aproxima un algoritmo concreto al óptimo.

FIFO: Reemplazar la página que hace más tiempo que fue cargada. No siempre tiene un buen rendimiento, ya que la más antigua no siempre es la que más se va a tardar en necesitar.

LFU: Reemplazar la página que ha sido usada con menor frecuencia utilizando contadores.

MFU: Reemplazar la página que ha sido usada con mayor frecuencia utilizando contadores.

Familia LRU: Reemplazar la página que hace más tiempo fue utilizada bajo la hipótesis de que si una página no ha sido usada en un gran periodo de tiempo puede que no se use en un futuro. Estos algoritmos requieren soporte por parte del hardware a través de un bit en la tabla de páginas (bit de referencia). El bit se pone a 1 cada vez que la instrucción ejecutada referencia una página. A los algoritmos que siguen esta aproximación se les denomina **NRU**. También están aquellos que son mejorados incluyendo el valor del bit de modificado en el criterio de elección de la página.

Algoritmos de Buffering de Páginas: Se puede mantener una lista de marcos libres. Cuando se produce un fallo de página, se escoge un marco de la lista y se carga la página, a la vez que se elige una página como víctima y se copia al disco. Esto permite que el proceso se reinicie lo antes posible, sin esperar a que la página reemplazada sea escrita en disco. Cuando la escritura finaliza, el marco es incluido en la lista de marcos libres.

Una mejora de lo anterior sería recordar qué página estuvo en cada marco antes de que éste pasara a la lista de marcos libres, para que las páginas puedan ser recuperadas desde la lista si fallara alguna antes de que su marco fuera utilizado por otra página. Se puede mantener una lista de páginas modificadas e ir escribiendolas cuando el dispositivo del espacio de intercambio no esté ocupado. Esto aumenta la probabilidad de que una página esté limpia cuando sea seleccionada, evitando escritura en el disco.

21. Reemplazo Local Frente a Reemplazo Global

Reemplazo Local: Solo se pueden escoger entre marcos asignados al proceso.

- **El número de marcos asignados no cambia porque ocurran fallos de página.**
- **Un proceso no puede hacer disponible a otros los marcos que menos usa.**

Reemplazo Global: Se pueden escoger entre todos los marcos del sistema.

- **El número de marcos asignados a un proceso puede aumentar.**
- **Un proceso no puede controlar su tasa de fallos de página**, ya que depende del comportamiento de otros procesos.
- **Mayor rendimiento**, por lo que es el método más utilizado.

Asignación de Marcos:

- **¿Cómo repartir los marcos entre procesos?**
  - Repartir la memoria por igual entre todos.
  - Repartir en proporción a la cantidad de memoria que utilizan.
- **El mínimo número de marcos viene establecido por diversos factores:**
  - Un proceso debe **disponer de suficientes marcos para guardar todas las páginas** a las que una instrucción pueda acceder, de lo contrario el proceso nunca podría ser reiniciado al fallar en alguno de los accesos a memoria.
  - **Todo proceso tiene una cantidad de páginas que utiliza frecuentemente**. Si no **dispone de suficientes marcos para alojar dichas páginas**, generará fallos de página con demasiada frecuencia.

22. Hiperpaginación

En los primeros sistemas multiprogramados el sistema operativo monitorizaba el uso de la CPU, si era bajo se cargaban nuevos procesos de la cola de entrada. Si un proceso necesitaba demasiada memoria, le podía quitar marcos a otro, ya que se utilizaba reemplazo global. Esto podía ocasionar que aumentara la tasa de fallos de página del proceso que perdía marcos. Al aumentar los fallos de página, el uso de la CPU decrecía, por lo que el sistema cargaba más procesos para aumentar el uso de la CPU, esto reducía la memoria disponible, lo que aumentaba la tasa de fallos de página, y así iterativamente reducía el rendimiento.

Se dice que un proceso sufre de **hiperpaginación** cuando gasta más tiempo paginando que ejecutándose. El uso de la CPU aumenta hasta alcanzar un máximo, si el grado de multiprogramación supera dicho punto, el sistema está hiperpaginado. Por lo que es necesario reducir el grado de multiprogramación para liberar memoria.

En los sistemas modernos ocurre algo parecido, aunque sin el efecto en cadena ocasionado por el planificador a largo plazo, ya que se carece de dicho planificador. Los procesos hiperpaginarán si no se les asigna un número suficiente de marcos.

Soluciones a la Hiperpaginación:

- **Utilizar un algoritmo de reemplazo local** pues de esta manera un proceso que hiperpagina no afecta a otro, pero el uso intensivo del dispositivo de intercambio puede afectar el rendimiento y aumentar el tiempo de acceso efectivo.

- **Proporcionar a un proceso los marcos que necesite.** A priori este número mínimo no es conocido, pero el modelo de conjunto del trabajo pretende estimar el número.

Modelo del Conjunto de Trabajo: Establece que una localidad es un conjunto de páginas juntas que se utilizan, y cuando un proceso se ejecuta se va moviendo entre localidades. Si proporcionamos a un proceso suficientes marcos para alojar toda su localidad en un momento dado, el proceso generará fallos de página hasta que todas sus páginas estén cargadas, pero después de esto no volverá a fallar hasta cambiar de localidad.

Esta estrategia permite obtener una aproximación de la localidad del programa:

- Definir Δ como el tamaño de la ventana del conjunto de trabajo
- En un instante dado, el conjunto de páginas presente en las Δ últimas referencias a la memoria se consideran el conjunto de trabajo.
- El conjunto de trabajo es una aproximación de localidad del programa.

La precisión de la aproximación de la localidad del programa depende de Δ. Por ejemplo:

- **Si Δ es muy pequeña** $\rightarrow$ El conjunto de trabajo no cubre la localidad.
- **Si Δ es muy grande** $\rightarrow$ En el conjunto de trabajo se superponen varias localidades.

Uso del Conjunto de Trabajo:

- Se selecciona Δ.
- El sistema monitoriza el conjunto de trabajo de cada proceso y le asigna tantos marcos como páginas hallan el conjunto de trabajo.
- Si sobran suficientes marcos se puede iniciar otro proceso o se puede destinar la memoria libre a otros usos.
- Si el tamaño del conjunto **D** crece y excede el número de marcos disponibles, el sistema selecciona un proceso para suspender (podrá ser reiniciado más tarde).
- **D** es la suma del tamaño de conjuntos WSS_i para cada proceso i ($\mathbf{D = \Sigma WSS_i}$).
- **D** representa la demanda total de marcos.
- Si **D** es mayuor que el número de marcos disponibles, habrá hiperpaginación.

Prepaginado: Es una técnica que consiste en cargar múltiples páginas junto con la página demandada en cada fallo de página. Las páginas se escogen especulativamente bajo la hipótesis de que van a ser necesitadas por el proceso en poco tiempo, de manera que si es cierto, la tasa de fallos se reduce significativamente. Puede ser utilizada:

- **En paginación bajo demanda pura**, cuando se inicia un proceso siempre fallan las primeras páginas, por lo que son buenas candidatas para el prepaginado.
- **En el acceso secuencial a archivos mapeados**. Si el sistema determina que el acceso es secuencial, en cada fallo puede cargar páginas siguientes en previsión de su uso.

23. Aplicaciones en Modo RAW

Muchos sistemas modernos permiten que los programas puedan acceder a los discos en modo RAW, donde no hay sistema de archivos, ni paginación bajo demanda, ni bloqueo de archivos, ni prepaginación, ni muchos otros servicios. Por lo que dichas aplicaciones deben implementar sus propios algoritmos de almacenamiento y gestión de memoria.

Los gestores de bases de datos conocen sus necesidades de memoria y disco mejor que cualquier sistema operativo general, por lo que salen beneficiados de sus propios algoritmos de gestión de memoria y buffering de E/S.

24. Tamaño de las Páginas

Páginas Grandes → Menos fallos de página, TP más pequeñas y menor tiempo de acceso de E/S a los contenidos de cada página.
Páginas Pequeñas → Menos frag. interna y mejor resolución para asignar y transferir al disco solo la memoria necesaria, a la larga debería redundar en menos memoria asignada y menos operaciones E/S.

Muchos sistemas modernos soportan el uso simultáneo de múltiples tamaños de página. La mayor parte de programas utilizan el tamaño estándar, mientras los que hacen uso intensivo de memoria pueden utilizar páginas de mayor tamaño. En la actualidad el tamaño de página más común es de **4 KiB en sistemas de 32 bits y 8 KiB en sistemas de 64 bits**.

25. Estructura de los Programas

Los programas estructurados con una buena localidad de referencia pueden mejorar su rendimiento en los sistemas con paginación bajo demanda. La selección cuidadosa de las estructuras de datos y de programación pueden mejorar la localidad, reduciendo los fallos de página y el tamaño del conjunto de trabajo(El lenguaje de programación puede tener efecto).

El compilador y el cargador también pueden tener un efecto importante:

- Separando el código de los datos para permitir que las páginas de código puedan ser de solo lectura, por ende las **páginas modificadas no tienen que ser escritas antes de ser reemplazadas**.
- El compilador puede **colocar las rutinas que se llaman entre sí en la misma página**.
- El cargador puede **evitar situar rutinas de forma que crucen los bordes de páginas**.

26. Interbloqueo de E/S

Si una página es reemplazada después de que el proceso queda en espera pero antes de que una operación E/S sea realizada, la operación se podría realizar sobre una página que pertenece a un proceso diferente.

Soluciones:

- Se puede utilizar memoria del núcleo como buffer en las operaciones E/S.
- Cada página puede tener un bit de bloqueo para indicar qué páginas no pueden ser seleccionadas para el reemplazo.

Además los bits de bloqueo se pueden utilizar para:

- **Bloquear páginas del núcleo para evitar que sean reemplazadas**.
- **Bloquear páginas que acaban de ser cargadas**.
- En los **sistemas de tiempo real flexible** se permite que las tareas de tiempo real informen de **cuáles son las páginas más importantes para que sean bloqueadas**.

27. Interfaz de Gestión de Memoria

Gracias a la abstracción de las técnicas de memoria virtual en cualquier sistema moderno solo hace falta una llamada al sistema para gestionar su espacio de direcciones virtual. En los sistemas POSIX esta llamada es **mmap()** (su opuesta **munmap()**) y sirve para:

- Reservar una porción del espacio de direcciones virtual del proceso. Siendo el componente de paginación bajo demanda el responsable de asignar la memoria física que la respalda.
- Establece permisos, compartición entre procesos, bloqueo de páginas en memoria física, páginas de gran tamaño, etc. en la región de memoria virtual a reservar.
- Mapear archivos en regiones del espacio de direcciones virtual.

28. Uso del Espacio de Direcciones Virtual del Proceso

Los procesos pueden utilizar diversas ubicaciones dentro de su espacio de direcciones virtual para almacenar los datos que necesitan para su ejecución:

- **Variables y constantes globales se almacenan en la sección de datos**, que tiene tamaño fijo ya que las dimensiones de estas variables se conocen de antemano en tiempo de compilación.
- **Variables locales y argumentos de las rutinas se almacenan en la pila** junto con las direcciones de retorno de las mismas. Puesto que la pila es restablecida en el retorno, al estado que tenía antes de invocar la rutina, esto hace que las variables desaparezcan automáticamente.
- **Variables asignadas dinámicamente se almacenan en el montón**, que es una región continua de memoria inmediatamente después de la sección de datos del proceso.

29. Gestión de la Memoria del Montón

Cuando un proceso hace una petición de asignación de memoria dinámica espera que el espacio ofrecido sea continuo en el espacio de direcciones virtual, por lo que es necesario utilizar algún algoritmo de reserva de memoria continua. Como las peticiones de los procesos son de tamaño variable, la forma más eficiente de enfrentar este problema es utilizando un esquema de particionado dinámico:

- La librería mantiene una lista indicando que regiones del montón están libres o no.
- Cuando un proceso realiza una petición se busca un hueco suficientemente grande para atenderla. Si se encuentra, solo se le asigna el espacio necesario, marcándolo como ocupado en la lista.
- Si el proceso libera una porción y se crean dos huecos adyacentes, se fusionan.

En la asignación dinámica hay diversas soluciones (Primer ajuste, Mejor ajuste, Peor ajuste).

30. Fragmentación

La estrategia comentada sufre de fragmentación interna, porque por ejemplo, en las peticiones grandes, **mmap()** reserva en múltiplos de tamaño de página, por lo que siempre se puede perder cierta cantidad, aunque pequeña en comparación con el tamaño de la región reservada. También existe fragmentación externa, es decir, el espacio está fraccionado en un gran número de huecos pequeños, obligando a la librería a invocar la llamada **brk()** con el objetivo de incrementar el tamaño del montón. Este problema:

- Afecta al primer y el mejor ajuste.
- Con el primer ajuste, con algunas optimizaciones, con **n** bloques asignados se pierde 0.5**n** por fragmentación externa (la regla del 50%).

Este problema no tiene una solución sencilla, ya que aunque se podría intentar mover los bloques de memoria para dejar un único hueco, sería necesario modificar en tiempo de ejecución las direcciones virtuales utilizadas por el proceso.

Tema 5: Gestión de Almacenamiento

1. Dispositivos de Almacenamiento

Discos Magnéticos: Son el tipo principal de almacenamiento secundario, generalmente denominados discos duros. La información se graba magnéticamente sobre platos, para lo cual se utilizan cabezas de lectura por encima y por debajo de cada plato. La superficie de cada plato está dividida en pistas circulares, cada una se subdivide en sectores y el conjunto de pistas situadas en la misma posición en distintos platos se denomina cilindro.

Discos Ópticos (CD, DVD, BluRay): Consisten en un disco circular en el que la información se guarda haciendo uso de surcos microscópicos que se leen incidiendo un láser sobre una de las caras planas que lo componen. La información se almacena siguiendo el recorrido en espiral de la superficie del disco, desde el interior hasta el exterior. El acceso aleatorio a los datos es lento, dado que para ello el láser debe desplazarse sobre la espiral.

Memorias de Estado Sólido (USB, SSD): Dispositivo de almacenamiento que usa una memoria no volátil, como las memorias flash, para almacenar datos. Se almacena como en un vector lineal de bytes, aunque de cara al resto del sistema, muestra una interfaz similar a la de los discos magnéticos.

2. Interfaz del Sistema de Archivos

El sistema de archivos proporciona mecanismos para el almacenamiento de datos y programas tanto para el SO como para los usuarios. Están compuestos de:

- **Una colección de archivos** → Almacena una serie de datos relacionados.
- **Una estructura de directorios** → Organiza todos los archivos del sistema y proporciona información acerca de los mismos.

3. Volúmenes de Datos

En ocasiones es interesante dividir el dispositivo con el objetivo de utilizar múltiples sistemas de archivos. En algunos sistemas estas regiones o dispositivos de almacenamiento pueden combinarse para crear estructuras de mayor tamaño, denominadas volúmenes, que pueden albergar un sistema de archivos. En general es un espacio de almacenamiento que alberga un sistema de archivos, tanto si es una partición como si es una estructura de mayor tamaño.

RAID (Redundant Array of Inexpensive Disks): Permite combinar varios discos duros para mejorar las prestaciones a través del paralelismo en el acceso y/o para mejorar la fiabilidad a través del almacenamiento de información redundante. Existen diversos niveles de RAID:

- **RAID 0** → Los datos se distribuyen equitativamente en bloques de tamaño fijo entre dos o más discos, sin incluir información redundante. Permite leer y escribir más datos al mismo tiempo (Se pueden enviar peticiones en paralelo). La fiabilidad es inversamente proporcional al número de discos, ya que si uno falla el conjunto falla.

- **RAID 1** → Se crea una copia exacta de los datos en dos o más discos, por tanto, se incrementa exponencialmente la fiabilidad, ya que para que el conjunto falle, todos deben fallar. El rendimiento de las operaciones de lectura se incrementa linealmente con el número de copias, ya que los datos están disponibles en todos los discos y se balancea la lectura entre todos ellos.

- **RAID 5** → Se distribuyen los datos equitativamente en bloques de tamaño fijo entre dos o más discos y se utiliza uno adicional para almacenar información de paridad de los bloques de una misma división. El disco utilizado para almacenar el bloque de paridad cambia de forma escalonada de una división a la siguiente, por ello se dice que está distribuido. Aspectos a tener en cuenta:

    - **Cuando se escribe un bloque se debe actualizar el de paridad** (costoso)
    - **Bloques de paridad no se leen durante las lecturas de datos**, ya que esto reduciría el rendimiento. Solo se hace cuando la lectura de un sector falle.
    - **El fallo de dos discos provoca la pérdida completa de los datos.** Aunque se pueden añadir discos de manera ilimitada, eso no suele ocurrir puesto a que cuantos más discos, más probabilidad de que dos de ellos fallen.

- **RAID 6** → Se utiliza la misma estrategia que el RAID 5 pero con dos bloques de paridad, lo que permite el fallo de dos discos sin pérdida de datos.

En un conjunto con niveles anidados se combinan varios niveles RAID básicos como si fuesen capas superpuestas:

- **RAID 0+1** → Se hace un espejo de un conjunto RAID 0.
- **RAID 1+0** → Varios conjuntos en espejo se combinan aumentando la capacidad total.
- **RAID 5+0** → Varios conjuntos RAID 5 se combinan aumentando la capacidad total.

La implementación de RAID es otra de las áreas donde existen diversas variantes:

- **Implementarse en el hardware de la controladora del disco**, de forma que sólo los discos conectados a esta pueden formar parte de un conjunto RAID determinado. Es una solución eficiente, especialmente cuando se utilizan niveles que requieren cálculo de paridad, ya que se evita usar la CPU. Sin embargo, estas controladoras son notablemente más caras que las que carecen de soporte RAID.

- **Implementarse dentro del SO** en lo que se denomina software de gestión de volúmenes. En este caso las soluciones RAID con paridad son bastante lentas, por lo que normalmente sólo se soportan los niveles RAID 0, 1, 1+0 o 0+1.

Cada conjunto RAID se comporta como una unidad de almacenamiento independiente, por lo que se puede utilizar entero para albergar un único sistema de archivos. Sin embargo lo más común es dividirlo en regiones para utilizar múltiples sistemas de archivos o combinarlo en estructuras de mayor tamaño.

4. Particiones

Un dispositivo de almacenamiento se puede dividir en regiones para utilizar en cada una de ellas un sistema de archivos diferente. A estas regiones se les conoce como particiones, franjas o minidiscos. Según la plataforma, existen diversas maneras de implementar el soporte de particiones, como el **MBR** (Master Boot Record) y la **GPT** (GUID Partition Table). Se almacena, en los primeros sectores del dispositivo de almacenamiento, una tabla con una entrada por partición donde se guardan las direcciones del primer y último sector de cada una de ellas en el dispositivo, así como otra información.

5. Volúmenes Dinámicos

Según la tecnología que se utilice para particionar es posible encontrarse con una serie de restricciones comunes:

- **El limitado número de particiones diferentes que puede contener un dispositivo.**
- **Imposibilidad de redimensionar particiones.** Especialmente si el sistema operativo está en ejecución.
- **Imposibilidad de crear particiones que hagan uso de regiones libres en diferentes dispositivos de almacenamiento.**

Algunos sistemas operativos incluyen un software de gestión de volúmenes que hace uso de tecnología propia para superar estas limitaciones. Estas herramientas generalmente permiten agrupar dispositivos, conjuntos RAID, particiones, etc. y sobre ellos construir los volúmenes que sean necesarios.

Estos volúmenes pueden ser redimensionados y se pueden incluir dinámicamente nuevos dispositivos para incrementar el espacio disponible. El software de gestión de volúmenes puede incluir alguna funcionalidad propia de conjuntos RAID con el objetivo de mejorar las prestaciones, a través del paralelismo en el acceso, y/o mejorar la fiabilidad a través del almacenamiento redundante.

6. Estructura de un Sistema de Archivos

1. **Control E/S (Nivel más bajo):** Está compuesto por los controladores de dispositivo encargados de transferir la información entre la memoria y el disco. Estos controladores, que generalmente son compartidos entre sistemas de archivos, transfieren los datos en unidades de bloques.

2. **Sistema Básico de Archivos:** Se encarga de enviar comandos al controlador de dispositivo apropiado con el fin de leer y escribir bloques físicos en el disco.

3. **Módulo de Organización de Archivos:** Tiene conocimiento de los archivos y se encarga de traducir las direcciones en direcciones físicas de bloque que serán enviadas al sistema básico de archivos para que realice las transferencias solicitadas.

4. **Sistema Lógico de Archivos:** Gestiona metadatos, que incluyen toda la estructura del sistema de archivos, excepto los propios datos de los archivos. Entre dichos metadatos está la estructura de directorios y los bloques de control de archivo.

   Un bloque de control de archivo o **FCB** (File Control Block) contiene información acerca del archivo, incluyendo su propietario, los permisos y la ubicación del

contenido del mismo. Además el sistema lógico también es responsable de las tareas de protección y seguridad.

Cada SO puede soportar uno o más sistemas de archivos para dispositivos de disco:

- **Unix** → Se utiliza el sistema de archivos **UFS** (Unix File System), basado en el sistema **FFS** (Fast File System) de Berkeley.
- **Linux** → Soporta más de 40 sistemas de archivo, entre los que podemos destacar el sistema de archivos extendido (ext2, ext3 y ext4), **XFS** y **BtrFS**.
- **Windows** → Soporta los sistemas de archivo **FAT**, **FAT32** y **NTFS** (NT File System).
- Además la mayoría de los SO modernos soportan otros sistemas de archivo, como los utilizados en soportes removibles, por ejemplo el **ISO-9660**, usado por la mayor parte de los CD-ROM, o el **UFS** (Universal File System), usado en los DVD-ROM.

7. Estructura de Metadatos

Para implementar un sistema de archivos se utilizan diversas estructuras de metadatos alojadas tanto en disco como en memoria. Estas estructuras varían dependiendo del SO y del sistema de archivos.

- **Un bloque de control de arranque (sector de arranque)** → que suele ocupar el primer bloque de cada volumen y que contiene la información necesaria para iniciar un SO a partir de dicho volumen.
- **Bloque de control de volumen** → contiene todos los detalles acerca del volumen. En los sistemas de archivos para UNIX y Linux, a esta estructura se la denomina superbloque. Mientras en NTFS esta información se almacena en la tabla maestra de archivos o **MFT** (Master File Table).
- **FCB por archivo** → Todos los FCB del sistema de archivos se almacenan en una tabla denominada directorio de dispositivo o tabla de contenidos del volumen.
    - Unix y Linux → Cada FCB se denomina **inodo**, se almacenan a continuación del superbloque.
    - NTFS → Esta información se guarda a la MFT, ya que cada entrada es un FCB.

Las estructuras existentes en memoria pueden ser:

- **Tabla de montaje** que contiene información sobre cada volumen montado.
- **Caché de la estructura de directorios** que almacena información relativa a los directorios a los que se han accedido recientemente. Los directorios que actúan como puntos de montaje pueden contener un puntero a la entrada, en la tabla de montaje, del volumen montado en el directorio.
- **Tabla global de archivos abiertos** que contiene una copia del **FCB** de cada archivo abierto en el sistema, además de otras informaciones.
- **Tabla de archivos abiertos de cada proceso** que contiene un puntero a la entrada de la tabla global, así como información adicional particular de cada proceso.

8. Montaje del Sistema de Archivos

Un sistema de archivos debe montarse para que sus archivos sean accesibles a los procesos del sistema. El proceso de montaje incluye los siguientes pasos:

1. **Al SO se le debe proporcionar el nombre del dispositivo y el punto de montaje** (Ubicación en la estructura a la que queremos conectar el sistema de archivos).
2. A continuación **el SO verifica que el dispositivo contiene un sistema de archivos válido** (Lee el directorio de dispositivo y comprueba la validez de su formato)
3. Finalmente, **el SO registra en la estructura de directorios que hay un sistema de archivo montado** en el punto de montaje especificado, esto permite que pueda ser recorrida la estructura de directorios, pasando de un sistema de archivos a otro según sea necesario.

En muchos SO modernos el montaje se ejecuta automáticamente cuando los dispositivos son detectados en el arranque del sistema o cuando se conectan durante su funcionamiento.

9. Archivos

Cada sistema de archivos contiene una tabla donde cada entrada almacena un bloque de control de archivo o **FCB** (File Control Block) por archivo. Un archivo es una colección de información relacionada cuya estructura y el significado de los datos lo define su creador. Para los usuarios un archivo es la unidad más pequeña de almacenamiento. No se pueden escribir datos en el almacenamiento secundario a menos que éstos se encuentren dentro de un archivo. Atributos:

- **Nombre.**
- **Identificador.**
- **Tipo.**
- **Ubicación.**
- **Tamaño.**
- **Protección.**
- **Fecha, hora.**
- **Identificación del usuario.**

El sistema operativo proporciona llamadas al sistema para:

- **Crear.**
- **Escribir.**
- **Leer.**
- **Reposicionar.**
- **Borrar.**
- **Truncar.**

Además, en muchos sistemas se suelen incluir llamadas para otras operaciones comunes, como añadir datos al final de un archivo y el renombrado de un archivo existente, y consultar y modificar diversos atributos de un archivo, como su longitud y el propietario del mismo.

Operaciones de Archivos

La mayor parte de estas implican realizar una búsqueda en el directorio para encontrar la entrada asociada con el archivo indicado. Para evitarlo, muchos sistemas requieren una llamada al sistema **open()** antes de realizar cualquiera de estas operaciones por primera vez sobre un archivo, ésto crea una entrada del archivo en la tabla de archivos abiertos, donde se almacena información como su ubicación en disco, las fechas de acceso y su tamaño.

A partir de ese momento, cuando se desea solicitar una operación sobre un archivo, sólo es necesario proporcionar un puntero a la entrada del mismo en la tabla de archivos abiertos, evitando así que haga falta realizar una exploración del directorio. Cuando el archivo deja de ser usado activamente por el proceso, puede ser cerrado con la llamada **close()**. En los SO donde varios procesos pueden abrir un mismo archivo se suelen usar dos niveles de tablas de archivos abiertos:

- **Tabla para cada proceso** (En el PCB) donde se indican todos los archivos que éste ha abierto. Se almacena toda la información referente al uso de cada archivo por parte del proceso, por ejemplo, el puntero actual utilizado por las operaciones de lectura y escritura, así como los derechos de acceso.

- **Tabla global para el sistema** donde se almacena toda la información independiente de los procesos, como la ubicación del archivo en disco, las fechas de acceso y el tamaño del archivo. También se puede almacenar un contador de aperturas para cada archivo, con objetivo de indicar cuántos procesos mantiene abierto cada archivo.

Tipos de archivos

Si el SO reconoce el tipo de un archivo, puede operar con el mismo de formas razonables. Cuando se diseña un SO es necesario considerar si debe reconocer y soportar el concepto de tipo de archivo. En los SO más comunes las técnicas utilizadas para implementar los tipos de archivo son las siguientes:

- **MSDOS y Windows** → El tipo de archivo se incluye como parte del nombre del archivo. El nombre se divide en **nombre y extensión** separadas por punto.
- **Mac OS X** → Cada archivo tiene un **atributo que almacena el tipo** (Ej: TEXT, archivos de texto; APPL, aplicaciones) **y otro que contiene el nombre del programa que lo creó**. Cuando el usuario hace clic con el ratón sobre el icono de un archivo, el programa que lo creó se ejecuta automáticamente y el archivo se carga en memoria.
- **UNIX** → Se utiliza un **número mágico almacenado al principio** de algunos archivos para indicar el tipo del mismo, pero no todos tienen estos números, por lo que se permite hacer sugerencias en forma de extensiones del nombre. Sin embargo, estas extensiones no son obligatorias ni el sistema depende de ellas.

10. Estructuras de Directorios

Para organizar los archivos se utilizan los directorios. Un directorio puede considerarse una tabla de símbolos que traduce los nombres de los archivos en sus correspondientes entradas en el directorio de dispositivo.

Directorios de un nivel: Todos los archivos están en un único directorio y cuando el número de usuarios del sistema aumenta es más difícil escoger nombres diferentes para los archivos.

Directorios de dos niveles: Cada usuario tiene su propio directorio de archivos de usuario o **UFD** que cuelga del directorio maestro o **MFD**. Cuando un usuario se conecta al sistema o inicia un trabajo se explora el MFD, que está indexado por el nombre de los usuarios y donde cada una de sus entradas apunta al UFD de dicho usuario. La estructura descrita aísla a los usuarios, lo cual puede ser un problema en la cooperación, la solución pasa por utilizar nombres de ruta para designar a un archivo de forma inequívoca.

Directorios con estructura de árbol: La estructura anterior puede generalizarse en una estructura en árbol de altura arbitraria. Esto permite que los usuarios puedan crear sus propios subdirectorios. Cada árbol tiene un directorio raíz que puede contener tanto archivos como otros directorios, y a su vez **cada directorio puede contener un conjunto de archivos y subdirectorios**. Normalmente cada entrada de directorio incluye un **bit donde se indica si dicha entrada apunta a un archivo (0) o a un subdirectorio (1)**. Tipos de nombres de ruta:

- **Nombre de ruta absoluto** → Comienza en la raíz y va indicando los directorios que componen la ruta hasta llegar al archivo especificado.
- **Nombre de ruta relativo** → Define una ruta a partir del directorio actual.

Con una estructura de árbol se puede permitir que unos usuarios accedan a los archivos de otros usuarios, solo es necesario que se utilicen nombres de ruta para designar los archivos o que se cambie el directorio actual.

Directorios en grafo acíclico: Es una generalización de la estructura en árbol, que permite que archivos y subdirectorios existan simultáneamente en distintos lugares de la estructura. Permite a los usuarios compartir archivos de forma que se pueda acceder a los mismos directamente desde el directorio propiedad de cada usuario (Existen diversas rutas). Se pueden implementar de diversas formas:

- **Crear una entrada de directorio (enlace)** → Es un archivo que contiene la ruta relativa o absoluta de otro archivo o subdirectorio (UNIX → enlaces simbólicos).
- **Duplicar información de los archivos compartidos en cada directorio** → En los sistemas UNIX las entradas duplicadas se las conoce como enlaces duros.

Inconvenientes:

- Cada archivo puede tener múltiples nombres de ruta absoluta, por lo que si se intenta recorrer el sistema completo se debe evitar acceder más de una vez a los archivos y subdirectorios compartidos.
- Para liberar espacio asignado a un archivo compartido se suele:

    - **Dejar los enlaces** hasta que se intenten utilizar, en tal caso se determina que el archivo fue borrado y es un acceso ilegal (no existe).
    - **Almacenar en la entrada del archivo original un contador con el número de referencias al archivo**, cuando el contador sea 0, sabremos que fue borrado y se debe liberar el espacio asignado. En UNIX se usa para los enlaces duros.

Directorios en forma de grafo general: Uno de los principales problemas del anterior es garantizar que no exista ningún ciclo. Para evitar que en un grafo aparezca un ciclo al añadir un nuevo enlace, se pueden utilizar diversos algoritmos, sin embargo, por lo costoso que es, lo más común es ignorar los enlaces durante el recorrido de directorios. En caso de duplicación de entradas de directorio lo más sencillo es evitar que puedan haber múltiples referencias al mismo directorio.

11. Compartición de Archivos (Múltiples Usuarios y Protección)

Una técnica para resolver el problema de la protección consiste en asociar una contraseña a cada archivo o directorio. El sistema debe mantener más atributos por archivo y directorio que en un sistema monousuario, para una compartición y mecanismos de protección más eficientes. El propietario de un archivo es el usuario que puede cambiar atributos y conceder el acceso, se trata del usuario que dispone del mayor grado de control sobre el archivo.

El grupo es un conjunto de usuarios que pueden compartir acceso al archivo, el propietario es quien define qué operaciones pueden ser ejecutadas por el grupo. Los identificadores de propietario y grupo de un archivo son almacenados en los atributos del archivo. Se compara el identificador de usuario con el atributo del propietario para determinar su identidad, se determinan qué permisos son aplicables, el sistema aplica dichos permisos a la operación solicitada y la autorizará o denegará. Esquema para determinar permisos aplicables:

- Asociar a cada archivo o directorio una **ACL** (Access-Control List) que especifique los nombres de usuario o grupos y tipos de acceso para cada uno.
- Cuando se solicita acceder a un archivo, el SO comprueba la ACL asociada a dicho archivo, si el usuario o alguno de sus grupos está en la lista para el tipo de acceso solicitado, se permite el acceso.
- Permite la implementación de complejas metodologías de acceso.
- Construir la lista puede ser una tarea tediosa.

La entrada de directorio ahora tendrá tamaño variable para almacenar la ACL, lo que requiere mecanismos más complejos de gestión de espacio. Para solucionar algunos problemas de las ACL, muchos sistemas utilizan listas de control de acceso condensadas, que para ser condensadas, muchos sistemas clasifican a los usuarios en tres grupos: **propietario, grupo y todos**. Sólo es necesario un campo para cada clase de usuario, siendo cada campo una colección de bits, donde cada uno permite o deniega el tipo de acceso asociado. Las ACL condensadas son más sencillas de construir y son mucho más simples de gestionar al tener una longitud fija.

La técnica más común en los SO modernos es combinar ambos tipos de listas. Uno de los problemas es que los usuarios deben poder determinar cuando están activados los permisos ACL más generales, en Linux se usa el símbolo "+" detrás de los permisos de la ACL condensada para indicar dicha circunstancia, estos permisos pueden ser gestionados utilizando los comandos **setfacl** y **getfacl**.

Otra dificultad es la relativa a la asignación de procedencias cuando ambas ACL entran en conflicto, en general, se suele asignar a la ACL más prioridad que la ACL condensada, pues la primera tiene una granularidad más fina y no se asigna de forma predeterminada. Windows usa las ACL generales, mientras que en SSOO como Linux y Solaris se usan ambos tipos.

12. Semántica de Coherencia

Específica cuando las modificaciones que un usuario realice en los archivos serán observables por los otros usuarios, está directamente relacionada con los algoritmos de sincronización de procesos, sin embargo, los algoritmos no se implementan normalmente en el caso de la E/S de archivo, debido a la alta latencia y las bajas velocidades de transferencia de discos y redes.

Semántica UNIX: Las escrituras en un archivo abierto por parte de un proceso son visibles inmediatamente para otros usuarios que hayan abierto ese mismo archivo. Existe un modo de compartición que permite a los procesos compartir el puntero de ubicación actual dentro del archivo, y el aumento del puntero afecta a todos los procesos que comparten el archivo. En la semántica de UNIX, cada archivo está asociado con una única imagen física a la que se accede en forma de recursos exclusivo, esto puede provocar retardos en los procesos.

Semántica de Sesión: Suponiendo que una sesión de archivo es el conjunto de operaciones entre las llamadas **open()** y **close()**, el sistema de archivos (AFS) utiliza la siguiente semántica:

- Las escrituras de un archivo abierto por parte de un proceso no son visibles inmediatamente para todos los usuarios que también lo hayan abierto.
- Si se cierra el archivo, los cambios realizados son visibles únicamente en las sesiones que comiencen posteriormente.
- Un archivo puede permanecer temporalmente asociado a varias imágenes al mismo tiempo. Así se permite que múltiples usuarios realicen accesos concurrentes de lectura y escritura en sus propias imágenes de archivo, evitando retardos.

Semántica de archivos compartidos inmutables: Cuando un archivo es declarado como compartido por su creador ya no puede ser modificado, su nombre no puede reutilizarse y su contenido no puede ser modificado. La implementación de esta semántica en un sistema distribuido es muy simple.

13. Bloqueos de Archivo

Algunos SO proporcionan funciones para bloquear un archivo (o porciones de él) abierto. Esto permite que un proceso bloquee un archivo, lo que impide que otros procesos puedan acceder al archivo bloqueado. Esto resulta útil para aquellos archivos compartidos por varios procesos, como un archivo de registro del sistema que puede ser consultado y modificado por procesos distintos. Tipos de bloqueo:

- **Bloqueo Compartido** → Puede ser adquirido concurrentemente por varios procesos.
- **Bloqueo Exclusivo** → Sólo puede ser adquirido por un proceso a la vez.

Los SO pueden proporcionar mecanismos de bloqueo de archivos:

- **Obligatorios** → Después de que un proceso adquiera un bloqueo exclusivo, el SO impedirá a todos los demás procesos que accedan al archivo, esto ocurrirá incluso si los otros procesos no han sido programados para adquirir explícitamente el bloqueo, por tanto es el SO quien garantiza la integridad de los bloqueos. **(Windows)**
- **Sugeridos** → El SO sólo impedirá que accedan al archivo los procesos programados para adquirir el bloqueo, en caso contrario el SO no lo impedirá, por tanto son los desarrolladores de software quienes garantizan que se adquieran y liberen apropiadamente los distintos bloqueos. **(UNIX)**

14. Coherencia

Comprobación de Coherencia: El comprobador compara los datos de la estructura de directorios con los bloques de datos del disco y trata de corregir todas las incoherencias que detecte. Los algoritmos de asignación y gestión del espacio libre determinan los problemas y el éxito del comprobador. Si se utiliza un sistema de asignación enlazada y existe un enlace entre cada bloque y el siguiente, puede reconstruirse el archivo completo a partir de los bloques de datos, y así volver a crear la estructura de directorios.

La pérdida de una entrada de directorio en un sistema de asignación indexada es desastrosa ya que no hay forma de saber qué datos pertenecen al archivo. UNIX almacena en caché las entradas de directorio para las lecturas, pero todas las escrituras de datos que provoquen algún cambio en la asignación de espacio o en algún otro tipo de metadato se realizan síncronamente, antes de escribir los correspondientes bloques de datos.

Soft Updates: Para mejorar la eficiencia del sistema de archivos sin comprometer la coherencia en caso de fallo, los distintos sabores de los sistemas UNIX BSD utilizan una técnica denominada Soft Updates. Cuando se monta un volumen con la opción Soft Updates el sistema desactiva la escritura síncrona de los metadatos, permitiendo que estos sean escritos cuando los algoritmos de gestión de la caché lo consideren necesario, pero se impone cierto orden en el que dichas operaciones de escritura deben ser realizadas.

Por ejemplo, cuando se van a escribir en disco las modificaciones debidas a la creación de un nuevo archivo, el sistema se asegura de que primero se escribe el nuevo inodo para posteriormente escribir el directorio con la nueva entrada de archivo. Teniendo en cuenta que la entrada de directorio contiene un puntero al inodo, es sencillo darse cuenta de que haciéndolo en este orden nos estamos asegurando de que el sistema de archivos permanezca consistente aunque el sistema falle antes de actualizar la información en disco.

15. Sistemas de Archivos Basados en Registro

Técnicas de recuperación en registro para las actualizaciones de los metadatos del sistema de archivos. Un efecto colateral de la utilización de un registro es la mejora del rendimiento en el acceso al sistema de archivos.

- **Escrituras asíncronas de metadatos** en lugares aleatorios del volumen, **se transforman en escrituras síncronas secuenciales** (más eficientes) en el registro.
- **Las operaciones indicadas en el registro se aplican asíncronamente** mediante escrituras aleatorias en las estructuras apropiadas.

En los sistemas de archivos basados en registro todos los cambios en metadatos se escriben secuencialmente en un registro:

- Cada conjunto de operaciones necesario para una tarea es una transacción.
- Las operaciones necesarias para completar una transacción se escriben secuencialmente en un registro, cuando los cambios son escritos, se consideran confirmados y la llamada al sistema puede volver al proceso de usuario permitiendo continuar su ejecución.
- Mientras tanto, las operaciones indicadas en el registro son ejecutadas sobre las estructuras reales del sistema de archivos. A medida que se realizan los cambios, se actualiza el registro para indicar las operaciones completadas.
- Cuando todas las operaciones se han ejecutado con éxito, la transacción se considera completada y se elimina del registro.

Si el sistema falla:

- Durante el montaje del sistema de archivos **se comprueba el registro.**
- **Todas las transacciones que contenga el registro no habrán sido aplicadas**, por lo que **será necesario aplicarlas antes de terminar el montaje**.
- Puede que existan **transacciones no confirmadas** (no terminaron de ser escritas), **todos los cambios** correspondientes a estas transacciones que hubieran sido aplicados al sistema de archivos **deberán deshacerse para preservar la coherencia**.
- Esta técnica resulta común en muchos SSOO, es utilizada en sistemas tales como **ext3, NTFS, XFS, JFS, ReiserFS**, etc.

16. Sistemas de Archivos Basados en Copy-on-Write

Las técnicas anteriores son necesarias para preservar la coherencia porque la modificación de los metadatos se hace sobreescribiendo los datos existentes. **Los sistemas basados en copy-on-write evitan cambiar los metadatos sobreescribiendo en el sitio**. Buscan un hueco libre, hacen en él una copia del bloque completo con los cambios y después modifican los metadatos del sistema de archivos que sirven para localizar el bloque en su nueva ubicación. Estos cambios no se hacen sobreescribiendo, sino que se crean copias modificadas de los bloques afectados, lo que va seguido de cambios en los metadatos que ayudan a localizarlos.

El proceso se repite hasta que se alcanza el **bloque de control de volumen** y se cambia, momento en que toda la secuencia de cambios se consolida. Si el sistema falla antes de la modificación del **bloque de control de volumen**, durante el montaje del sistema de archivos no quedará ni rastro de ningún cambio porque dicho bloque aún hace referencia a la antigua raíz del árbol de FCB y, a partir de ellas, a todos los nodos, bloques y FCB originales. Los sistemas que implementan este tipo de sistemas de archivo usan la memoria principal como caché, con el objeto de combinar varias modificaciones en un mismo bloque antes de proceder a escritura en disco, evitando ocasionar múltiples veces los cambios posteriores.

17. Implementación de Directorios

Lista Lineal: El método más sencillo para implementar un directorio consiste en emplear una **lista lineal de nombres de archivos con punteros a los bloques de datos**. Una lista lineal de entradas del directorio requiere una búsqueda lineal para encontrar una entrada particular. Es sencillo de implementar pero costoso en tiempo. Para **crear** un archivo se requiere una búsqueda lineal (comprobar que el nombre no esté repetido) y se añade la entrada al final.

Eliminación:

- Búsqueda y liberación de la entrada.
- Reutilizar la entrada de directorio.
    - Marcar como desocupada (nombre especial o bit ocupado).
    - Añadir a lista de entradas libres de directorio.
    - Copiar la última entrada en la posición desocupada.

Desventajas:

- Búsqueda lineal para encontrar un archivo.
- Muy costoso en directorios con gran número de archivos, se puede usar una lista ordenada para reducir tiempo de búsqueda pero se complica la creación y borrado.

Tabla de Dispersión: En los directorios se almacenan las entradas de directorio en una lista lineal, pero al mismo tiempo se utiliza una tabla de dispersión para reducir el tiempo de búsqueda. La tabla toma un valor calculado a partir del nombre del archivo y devuelve un puntero a la ubicación del archivo dentro de la lista lineal. Su ventaja es que el método es normalmente más rápido que la búsqueda lineal.

Desventaja:

- Aparición de colisiones → Situaciones en las que dos nombres de archivo en la tabla proporcionan la misma ubicación dentro de la lista. Se puede resolver utilizando una lista enlazada en cada entrada, pero las búsquedas serán un poco más lentas.

Árbol B: El sistema NTFS utiliza una estructura denominada árbol B+ para almacenar el índice de los nombres de archivos contenidos en un directorio. En la entrada en la **MFT** de cada directorio se almacena un atributo denominado raíz del índice, que contiene el nivel superior del árbol B+. Para un directorio pequeño, todas las entradas se almacenan en la raíz del índice, dentro de su entrada en la **MFT**. Para uno grande, la raíz del índice sólo puede almacenar unas pocas entradas del directorio.

Cada una de esas entradas incluye un puntero al bloque del disco que contiene el nodo del árbol con las entradas con nombres alfabéticamente anteriores. Si en dicho nodo no caben todas las entradas, sólo podrá contener algunas de ellas, por lo que cada una tendrá un puntero a un nuevo nodo del árbol y así sucesivamente.

Ventajas:

- Eliminan el coste de reordenar las entradas del directorio.
- La longitud de la raíz del árbol hasta un nodo hoja es la misma para todas las rutas.

18. Métodos de Asignación

Asignación Contigua: Cada fichero ocupa un conjunto de bloques contiguos en el disco (optimiza el movimiento de las cabezas del disco). Entrada de directorio para cada fichero:

- **Dirección del bloque inicial.**
- **Longitud del área asignada al archivo** (nº de bloques).

Esta asignación permite manejar acceso tanto secuencial como directo. Sus problemas son:

- **Encontrar espacio para la creación de un fichero**. Algoritmos más usados: Primer ajuste, Mejor ajuste y Peor ajuste.
- **Fragmentación Externa.**
- **Determinar cuánto espacio se necesita por fichero.**

Incluso cuando se conoce la cantidad total de espacio requerido, la preasignación puede ser ineficiente → Crecimiento lento (fragmentación interna). Para evitar estos problemas algunos SO utilizan un esquema de asignación contigua modificado:

- **Se asigna un trozo contiguo** de espacio.
- **Cuando se requiere más espacio se añade otro** espacio contiguo (extensiones).
- **Ubicación de los bloques de un archivo**: Bloque inicial, nº de bloques, enlace al primer bloque de la siguiente extensión.
- **Sigue existiendo fragmentación interna** (si las extensiones son grandes) **y externa** (consecuencia de la asignación y liberación de extensiones de distintos tamaños).

Asignación Enlazada: Cada fichero es una lista enlazada de bloques de disco, los bloques pueden estar dispersos en cualquier lado del disco. La entrada de directorio puede contener:

- **Puntero al primer bloque.**
- **Puntero al último bloque.**

Ventajas:

- **No hay fragmentación externa.**
- **No es necesario declarar previamente el tamaño del archivo.**

Desventajas:

- **Sólo eficiente para archivos de acceso secuencial.**
- **Requerirá más espacio del que requeriría normalmente**, debido al espacio necesario para los punteros de un archivo.
- **Si un puntero falla, se pierde el fichero.**

La solución para mejorar las desventajas es agrupar los bloques en clústers, para conseguir:

- **Correspondencia sencilla entre bloques lógicos y físicos.**
- **Mejorar rendimiento del disco** (menos búsquedas de disco).
- **Reducir el espacio necesario para la asignación de bloques y la administración de la lista de espacio libre.**
- **Aumento de fragmentación interna** → Se desperdicia más espacio cuando un clúster está parcialmente lleno que cuando un bloque lo está.

Una variación del método es la **FAT** (File-Allocation Table):

- Se reserva una sección del disco al principio de cada partición para guardar una tabla de asignación de archivos.
- La tabla tiene una entrada por bloque de disco y se indexa por número de bloque.
- La FAT se emplea de forma similar a una lista enlazada:
  - La entrada contiene el número del primer bloque del archivo.
  - La entrada en la tabla indexada por dicho número de bloque contiene el número del siguiente bloque en el archivo.
  - La cadena continúa hasta el último bloque, que tiene un fin de archivo.
- Los bloques no usados se indican mediante el valor 0 en la tabla.
- La asignación de un nuevo bloque a un archivo implica encontrar una entrada en la tabla con valor 0 (normalmente la primera).
- El esquema FAT provoca un número considerable de búsquedas en la cabeza del disco a menos que la FAT se coloque en caché.

Asignación Indexada: La enlazada resuelve los problemas de la asignación contigua, pero si no se usa FAT no se puede implementar un acceso eficiente. La asignación indexada resuelve este problema reuniendo todos los punteros en un mismo lugar que llamamos **bloque índice**. La i-ésima entrada del bloque índice apunta al i-ésimo bloque del archivo. Cada archivo debe tener un bloque índice y conviene que este sea **lo más pequeño posible**, pero si es demasiado pequeño no podrá tener suficientes punteros para un archivo grande. Soluciones:

- **Esquema Enlazado** → Un bloque índice ocupa un bloque de disco y para archivos grandes se pueden enlazar varios bloques índice.
- **Índice Multinivel** → El bloque índice del primer nivel apunta a bloques índice de segundo nivel que estos a su vez apuntan a bloques de archivo.
- **Esquema Combinado (BSD UNIX)** → Se mantienen 15 punteros en el bloque índice, los 12 primeros apuntan a bloques directos (direcciones de bloques que contienen datos del archivo) y los 3 siguientes hacen referencia a bloques indirectos:
    - **El primero es un puntero a la dirección de un bloque indirecto simple** (bloque que contiene direcciones de bloques contienen datos).
    - **El segundo es un puntero de bloque indirecto doble** (La dirección de un bloque con las direcciones de bloques con punteros a bloques de datos).
    - **El tercero es un puntero con la dirección de un bloque indirecto triple**.

19. Gestión del Espacio Libre

Vector de Bits: La lista de espacio libre se implementa como mapa de bits o vector de bits. Cada bloque se representa con 1 bit **(Bloque libre → 1, Bloque ocupado → 0)**. **Ventajas:**

- Resulta fácil encontrar el primer bloque libre o n bloques libres.
- Muchos ordenadores cuentan con instrucciones de manipulación de bits.

El problema es que son ineficientes si el vector entero no se mantiene en memoria.

Lista Enlazada: Consiste en **enlazar juntos todos los bloques libres del disco** manteniendo un puntero al primer bloque libre en una localidad especial del disco y colocándolo en caché de memoria. El primer bloque contiene un puntero al siguiente y así sucesivamente. Su ventaja es que **aprovecha mejor el espacio** (no requiere de estructuras adicionales), pero **para recorrerla se tiene que leer cada bloque**, aunque el recorrido de la lista de bloques libres no es frecuente (normalmente el SO solo necesita un bloque libre para asignarlo a un archivo).

Agrupamiento: **Consiste en almacenar las direcciones de n bloques libres en el primer bloque libre**. Los primeros n - 1 bloques de estos bloques estarían realmente libres, pero el último apuntaría a otros n bloques libres. Se pueden localizar rápidamente las direcciones de un gran número de bloques libres, lo que mejoraría la eficiencia respecto a la lista enlazada.

Recuento: Normalmente los bloques son asignados o liberados en bloques contiguos, sobre todo si el espacio es asignado mediante asignación contigua o si se agrupan en clústers, esto se puede aprovechar para **mantener una lista donde cada entrada almacena la dirección del primer bloque de un conjunto de bloques libres contiguo**, así como el número de bloques.

20. Sistemas de Archivos Virtuales

Un SO moderno debe ser capaz de soportar de manera eficiente **distintos tipos de sistemas de archivos**, ocultando sus diferencias de cara al usuario. Un método para implementar múltiples tipos consiste en **escribir diferentes rutinas de acceso, manipulación y gestión, para cada uno de los tipos de sistema de archivos existentes**. Sin embargo, en lugar de hacer esto, **la mayoría de SSOO utilizan técnicas de programación orientada a objetos** para implementar diferentes tipos de sistemas de archivos detrás de una misma interfaz de programación. Se utilizan estructuras de datos y procedimientos comunes para separar las llamadas al sistema de los detalles de su implementación real, para cada sistema de archivos.

La implementación de un sistema de archivos está compuesta por tres niveles esenciales:

- **Interfaz del Sistema de Archivos** → Es a la que acceden los desarrolladores a través de las llamadas al sistema (open, read, write, close, etc.). Esta interfaz es la misma sea cual sea el sistema de archivos al que se intente acceder.

- **Sistema de Archivos Virtual o VFS (Virtual File System)** → Es utilizado por el anterior para atender las peticiones realizadas, describe operaciones genéricas sobre cualquier sistema de archivos y estructuras genéricas. Por ejemplo, el FCB virtual, que identifica de forma inequívoca a cada archivo o directorio en uso en todo el sistema, dando acceso a sus metadatos.

- **Implementación Real del Sistema de Archivos** → Se implementa cada tipo de sistema de archivos o los distintos protocolos de los servidores de archivos en red. La interfaz VFS recurre a la implementación correspondiente para cada tipo de sistema de archivos para satisfacer las solicitudes de los niveles superiores.

21. Planificación de Disco

Rendimiento de Acceso a Disco: (En un disco duro magnético)

- **Tiempo de acceso** → $\mathbf{T_D = T_B + T_R}$.
- **Tiempo de Búsqueda (T_B)** → Es el tiempo que se tarda en mover el brazo del disco hasta el cilindro deseado.
- **Latencia Rotacional (T_R)** → Es el tiempo de espera para que el disco gire hasta que la cabeza de lectura llegue al sector deseado del cilindro.
- Por tanto, el tiempo de acceso al disco es menor cuando se realizan accesos consecutivos a sectores físicamente próximos que cuando están dispersos.

Ancho de Banda o Tasa de Transferencia: Es el total de bytes transferidos en una petición dividido por el tiempo total que transcurre desde la primera solicitud hasta el final de la última transferencia, con la que se termina de atender la petición.

En los dispositivos basados en memorias de estado sólido, el tiempo de acceso viene determinado por las características de la memoria, las diferencias entre accesos secuenciales y accesos aleatorios son menos significativas.

Cola de E/S al disco: Cuando se solicita E/S sobre el disco, si la controladora y la unidad de disco están desocupadas, el SO puede atender la petición sobre la marcha, en caso contrario, el SO almacena la solicitud en una cola de peticiones. Cuando se resuelve una solicitud, el SO escoge otra de la cola y se comunica con el hardware para programar la siguiente petición. Existen distintas opciones de planificación:

- **FCFS** → Las solicitudes se atienden en orden de llegada, es el más simple y es equitativo ya que atiende a todos los procesos por igual, pero no es rápido en discos duros magnéticos, donde interesa mover el brazo del disco lo menos posible.

- **SSTF** → Se selecciona la solicitud con el menor tiempo de búsqueda desde la posición actual de la cabeza. La solución no es óptima, dado que puede provocar inanición de algunas solicitudes si van llegando constantemente solicitudes sobre regiones cercanas a la cabeza del disco.

- **SCAN y C-SCAN** → En SCAN, el brazo comienza en un extremo del disco y se mueve hacia el otro, atendiendo solicitudes a medida que pasa por cada cilindro hasta llegar al otro extremo del disco, donde la dirección de movimiento de la cabeza se invierte para recorrer el disco en sentido inverso, repitiendo el proceso. En el C-SCAN, cuando llega a un extremo vuelve al inicio para volver a barrer desde allí, sin atender a ninguna solicitud por el camino. Con esta variante el tiempo que tiene que esperar una solicitud es más uniforme que con el SCAN original.

- **LOOK y C-LOOK** → Variantes de SCAN y C-SCAN donde, cuando en el recorrido del brazo, tras atender una solicitud, se descubre que no hay más solicitudes en la misma dirección, el brazo invierte su dirección sin llegar al extremo del disco.

Estos últimos algoritmos sufren un problema denominado rigidez de brazo. Cuando hay un flujo continuo de solicitudes para el mismo cilindro, esto hace que el brazo no avance por los cilindros hasta llegar al otro extremo. En los siguientes, se separan las solicitudes en dos colas, haciendo que las nuevas esperen, por lo que el brazo siempre continúa su barrido.

- **N-Step-SCAN, N-Step-LOOK y F-SCAN** → Variantes de SCAN y LOOK donde se limita a *N* el número de solicitudes que se atenderán en cada barrido. Se utiliza una cola con espacio para *N* solicitudes, que se van atendiendo mientras el brazo barre el disco, mientras que, todas las nuevas solicitudes se incorporan a una cola diferente. Cuando el brazo termina el barrido y las *N* primeras solicitudes son atendidas, el planificador toma otras *N* solicitudes de la otra cola y las introduce a la primera para repetir el proceso. Si en lugar de copiar *N* peticiones se copian todas las solicitudes pendientes de la otra cola el algoritmo se denomina F-SCAN.

- **CFQ** → El planificador CFQ se diseñó para compartir de forma equitativa el ancho de banda entre todos los procesos que solicitan acceso al disco. Se mantiene una cola de solicitudes por proceso y en ella se insertan las solicitudes síncronas de E/S. Cada cola tiene una ventana de tiempo (cuanto) para acceder al disco, la longitud del cuanto y el tamaño de cada cola dependen de la prioridad de E/S del proceso.

  Se mantiene una cola de solicitudes por cada prioridad de E/S, donde se insertan las solicitudes asíncronas de todos los procesos, una solicitud asíncrona se inserta en una cola u otra según la prioridad del proceso que la generó. Usando Round Robin, el planificador CFQ recorre las colas y extrae de ellas las solicitudes durante el tiempo marcado por el cuanto de cada cola. Las solicitudes extraídas se insertan en la cola de E/S al disco, donde se ordenan para minimizar el tiempo de búsqueda, antes de ser enviadas al dispositivo.