Implementación del sistema de archivos 12

Sistemas operativos y distribuidos

Gustavo Distel
gd@cs.uns.edu.ar

DCIC - UNS

Implementación del sistema de archivos

Contenido

- Métodos de asignación.
- Gestión del espacio libre.

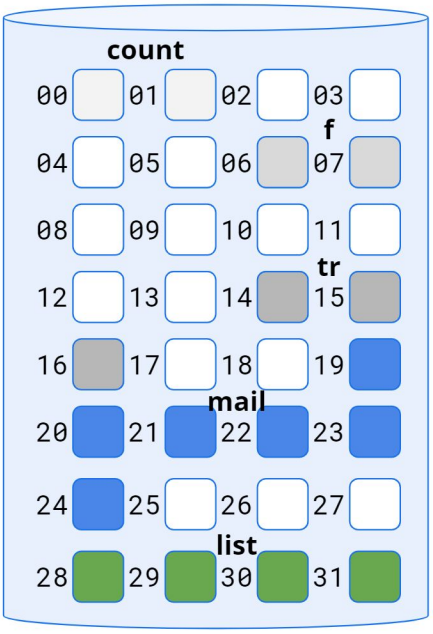
Métodos de asignación

Asignación contigua

- Cada archivo ocupa un conjunto de bloques contiguos en el dispositivo.
- El acceso al bloque **b + 1** después del bloque **b** requiere pocos movimientos en un disco *HDD*.
- Un archivo se define por la dirección del primer bloque y la longitud (en bloques) del archivo.
- Si el archivo tiene **n** bloques de largo y comienza en la ubicación **b**, entonces ocupa los bloques **b**, `b + 1, b + 2, ..., b + n - 1`.
- La entrada del directorio para cada archivo indica la dirección del bloque de inicio y la longitud asignada para el archivo.
- La asignación contigua es fácil de implementar pero tiene limitaciones y, por lo tanto, no se usa en los sistemas de archivos modernos.

Métodos de asignación

Asignación contigua



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Métodos de asignación

Asignación contigua

- **Acceso secuencial**: el sistema de archivos mantiene la dirección del último bloque al que se hace referencia y, cuando es necesario, lee el siguiente bloque.
- **Acceso directo**: para acceder al bloque `i` de un archivo que comienza en el bloque `b`, se accede inmediatamente al bloque `b + i`.
- **Desventajas**:
  - Encontrar espacio para un archivo nuevo. Idem a los problemas de asignación de memoria contigua, los cuales tienen fragmentación externa.
  - Determinar cuánto espacio se necesita para un archivo.
- Estas desventajas pueden ser minimizadas con un esquema de asignación contigua modificado, donde se utilizan *extents*: conjunto contiguo de bloques los cuales, a su vez, están enlazados.
- Si los *extents* son grandes habrá fragmentación interna.
- Habrá fragmentación externa a medida que se asignen y desasignen *extents* de diferentes tamaños.

Métodos de asignación

Asignación contigua: ejemplo

- Considerar un sistema que utiliza asignación contigua y asumiendo que:
  - El bloque de control de archivo se encuentra en memoria.
  - El bloque con información a agregar se encuentra en memoria.
  - En el cálculo de la cantidad operaciones de `E/S`, si la ubicación del primer bloque del archivo cambia, no incluir la operación necesaria para reescribir esa información en la entrada de directorio del disco.
  - Hay espacio para que el archivo crezca al final, pero no al principio.
  - El bloque del medio es el 25to bloque.
  - No incluir ninguna operación de `E/S` para agregar o remover un bloque de la lista de espacio libre.
- ¿Cuántas operaciones de lectura y escritura se requieren para agregar o remover un bloque de un archivo de `50` bloques para cada uno de los siguientes casos? Si el bloque:
  - `a)` Se agrega al inicio.
  - `b)` Se agrega después del bloque del medio.
  - `c)` Se agrega al final.
  - `d)` Se elimina del inicio.
  - `e)` Se elimina del medio.
  - `f)` Se elimina del final.

Métodos de asignación

Asignación contigua: ejemplo

- Respuestas:
  - `a)Se agrega al inicio`: `101`; cada bloque de todo el archivo debe desplazarse sobre el siguiente bloque (una lectura y una escritura por cada bloque) y luego se escribe el nuevo bloque al inicio.
  - `b)Se agrega después del bloque del medio`: `51`; la segunda mitad, que consta de `25` bloques, debe desplazarse sobre el siguiente bloque y luego se escribe el nuevo bloque.
  - `c)Se agrega al final`: `1;` solo se escribe el nuevo bloque al final.
  - `d)Se elimina del inicio`: `0` o `98`
    - `0`: simplemente se actualiza el puntero de entrada al directorio para que apunte al segundo bloque.
    - `98`: se mueven todos los bloques desde el segundo bloque hasta el final un bloque, lo que requiere `49` operaciones de lectura y `49` operaciones de escritura.
  - `e)Se elimina del medio`: `50;` para eliminar el bloque `25`, se deben mover `25` bloques de a un bloque, lo que requiere una operación de lectura y una de escritura por cada uno.
  - `f)Se elimina del final`: `0;` se actualiza la información de longitud en memoria.

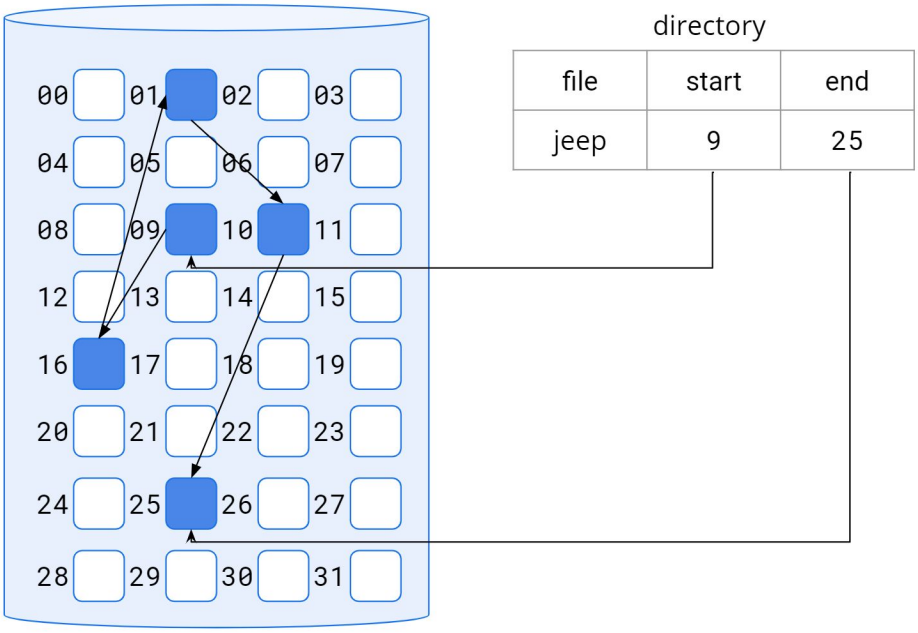
Métodos de asignación

Asignación enlazada

- La asignación enlazada resuelve los problemas de la asignación contigua.
- Cada archivo es una lista enlazada de bloques, que pueden estar dispersos en cualquier parte del dispositivo.
- El directorio contiene un puntero al primer y último bloque del archivo.
- Por ej., un archivo de cinco bloques puede comenzar en el bloque `9` y continuar en el bloque `16`, luego el bloque `1`, luego el bloque `10` y finalmente el bloque `25`.
- Cada bloque contiene un puntero al siguiente bloque, los cuales no están disponibles para el usuario. Por lo tanto, si cada bloque tiene un tamaño de `512` ***`bytes`*** y una dirección de bloque (el puntero) requiere `4` ***`bytes`***, entonces el usuario ve bloques de `508` ***`bytes`***.

Métodos de asignación

Asignación enlazada



Métodos de asignación

Asignación enlazada

- Para crear un archivo se agrega una nueva entrada en el directorio; cada entrada de directorio tiene un puntero al primer bloque, el cual se inicializa en *null* indicando un archivo vacío.
- No hay fragmentación externa y no es necesario declarar el tamaño de un archivo al crearlo.
- Desventajas:
  - Solo se puede usar de manera efectiva para archivos de acceso secuencial. Para encontrar el bloque `i-ésimo` de un archivo, debemos comenzar al principio de ese archivo y seguir los punteros hasta llegar al bloque `i-ésimo`.
    - Cada acceso a un puntero requiere una lectura al dispositivo de almacenamiento. En consecuencia, es ineficiente para el acceso directo.
  - El espacio de los punteros: si un puntero requiere `4 bytes` de un bloque de `512 bytes`, entonces el `0.78%` del disco se está utilizando para punteros, en lugar de información.
  - Confiabilidad (*reliability*).

Métodos de asignación

Asignación enlazada

- La solución habitual a este problema es reunir múltiples bloques, llamados *clusters*, y asignar *clusters* en lugar de bloques.
- Los punteros usan un porcentaje menor del espacio del archivo.
- Este método mejora el rendimiento del *HDD* (se requieren menos búsquedas de la cabeza del disco) y disminuye el espacio y gestión de la lista de espacio libre.
- El costo de este enfoque es que aumenta la fragmentación interna.
- Los *clusters* se pueden usar para mejorar el tiempo de acceso a disco para muchos otros algoritmos, por lo que se usan en la mayoría de los sistemas de archivos.
- Otro problema es la confiabilidad (*reliability*).
  - Un *bug* en *SW* del SO o una falla de *HW* puede dar lugar a un puntero equivocado. Este error a su vez podría dar lugar a un enlace a la lista de espacio libre o a otro archivo.
  - Una solución parcial es usar listas doblemente enlazada, y otra es almacenar el nombre del archivo y el número de bloque relativo en cada bloque. Sin embargo, estos esquemas requieren aún más sobrecarga para cada archivo.

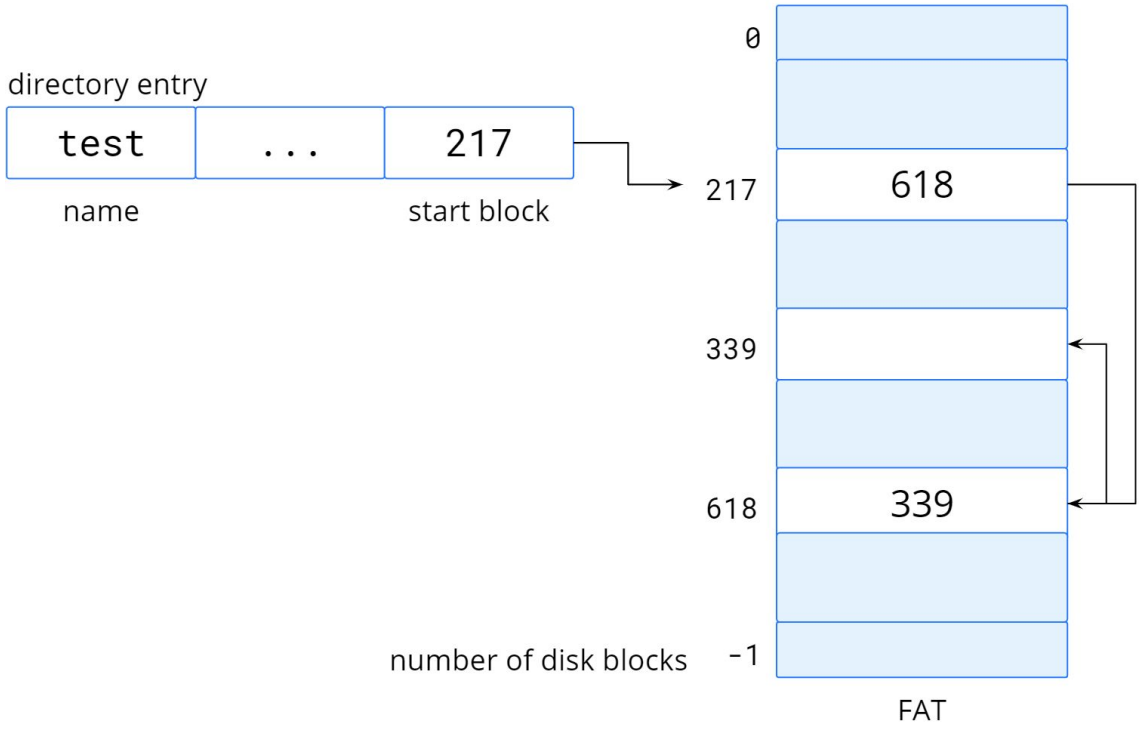
Métodos de asignación

Asignación enlazada

- Una variación es el uso de una tabla de asignación de archivos (*file allocation table, FAT*).
- Al comienzo de cada volumen se reserva espacio para la tabla, la cual tiene una entrada para cada bloque y está indexada por número de bloque.
- La entrada de directorio contiene el número de bloque del primer bloque del archivo.
  - La entrada de la tabla indexada por ese número de bloque contiene el número de bloque del siguiente bloque en el archivo.
  - Esta cadena continúa hasta llegar al último bloque, que tiene un valor especial de final de archivo como entrada de la tabla.
- Un bloque no utilizado se indica mediante un valor de tabla de `0`.
  - Asignar un nuevo bloque a un archivo consta en encontrar la primer entrada de la tabla con valor `0` y reemplazar el valor de fin de archivo anterior con la dirección del nuevo bloque.
- Un ejemplo ilustrativo de la estructura *FAT* se muestra a continuación para un archivo con bloques de disco `217`, `618` y `339`.

Métodos de asignación

Asignación enlazada



Métodos de asignación

Asignación enlazada: ejemplo

- Respuestas:
  - `a)Se agrega al inicio`: `1`; se escribe el nuevo bloque, haciendo que apunte al siguiente bloque y luego se actualiza el primer puntero del bloque en memoria.
  - `b)Se agrega después del bloque del medio`: `27`; para posicionarse en el medio se deben leer 25 bloques. Posteriormente, se necesita una escritura del nuevo nodo apuntando al bloque 26 anterior. Se necesita otra escritura para escribir el bloque 25 y que apunte al nuevo bloque.
  - `c)Se agrega al final`: `3`; primero se debe leer el último bloque, dado por puntero al último bloque y, entonces, se escribe el nuevo bloque. A continuación, el último bloque leído anteriormente se vuelve a escribir con el puntero del siguiente bloque modificado para apuntar al nuevo bloque. Al final, el puntero al último bloque se actualiza en memoria.
  - `d)Se elimina del inicio`: `1`; se lee el primer bloque para llegar al puntero del segundo bloque y luego el puntero de la entrada del directorio se actualiza para apuntar al segundo bloque.
  - `e)Se elimina del medio`: `26`; para encontrar el puntero al bloque 26, son necesarias 25 lecturas. Luego, el siguiente puntero del bloque 24 se actualiza con este valor.
  - `f)Se elimina del final`: `50`; el puntero del bloque de cola debe actualizarse para que apunte al anterior al último bloque y la única forma de obtener el penúltimo bloque es leer los 49 bloques anteriores. Luego, el siguiente puntero del bloque 49 debe actualizarse a un puntero nulo. Se actualiza el puntero a la cola en memoria.

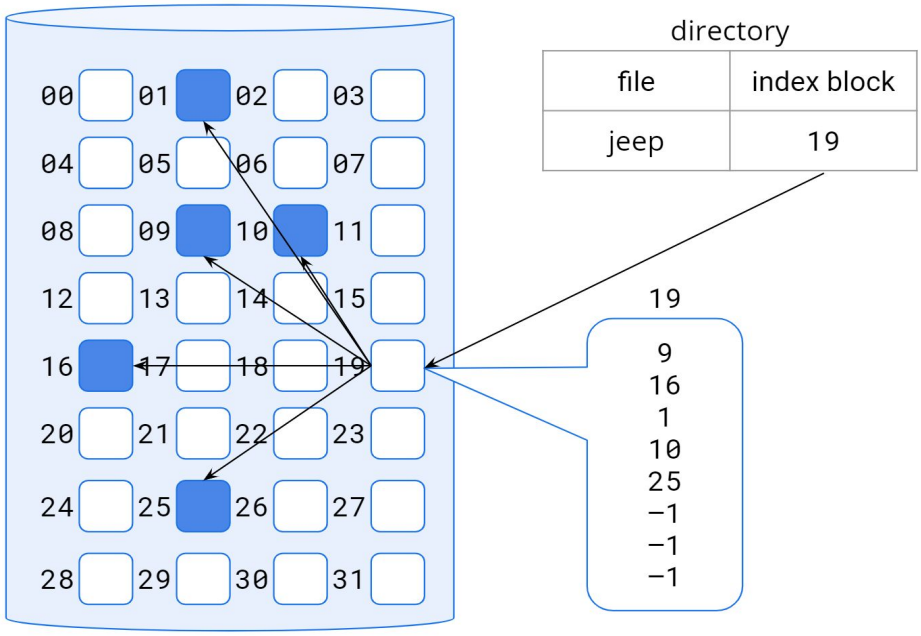
Métodos de asignación

Asignación indexada

- La asignación enlazada resuelve los problemas de fragmentación externa y la declaración de tamaño de la asignación contigua. Pero el acceso directo no es eficiente, ya que los punteros a los bloques están dispersos con bloques en todo el disco y deben recuperarse en orden.
- La asignación indexada resuelve este problema al reunir todos los punteros en una ubicación: el bloque índice.
- Cada archivo tiene su propio bloque índice, que es una matriz de direcciones de bloques.
- La **`i-ésima`** entrada en el bloque índice apunta al **`i-ésimo`** bloque del archivo.
- El directorio contiene la dirección del bloque índice.
- Para encontrar y leer el bloque **`i-ésimo`**, usamos el puntero en la entrada **`i-ésima`** del bloque índice.

Métodos de asignación

Asignación indexada



Métodos de asignación

Asignación indexada

- Cuando se crea el archivo, todos los punteros en el bloque índice se setean a *null*.
- Cuando el bloque **`i-ésimo`** se escribe por primera vez, se obtiene un bloque del espacio libre y su dirección se coloca en la entrada **`i-ésima`** del bloque índice.
- Admite acceso directo, sin fragmentación externa, porque cualquier bloque libre en el dispositivo de almacenamiento puede satisfacer la solicitud.
- Sin embargo, se desperdicia espacio.
  - La sobrecarga de punteros en el bloque índice es generalmente mayor que la sobrecarga de punteros en la asignación enlazada.
  - Considere el caso que se tiene un archivo de uno o dos bloques.
    - Con la asignación enlazada, perdemos el espacio de solo un puntero por bloque. Con la asignación indexada, se debe asignar un bloque índice completo, incluso si solo uno o dos punteros son distintos de *null*.

Métodos de asignación

Asignación indexada

- Cada archivo debe tener un bloque índice, por lo que se presente que el bloque índice sea lo más pequeño posible, aunque si es demasiado pequeño no podrá contener suficientes punteros para un archivo grande.
- Para este propósito existen los siguientes mecanismos:
- **Esquema enlazado**: un bloque índice es del tamaño de un bloque. Para permitir archivos grandes, se pueden enlazar varios bloques índice.
  - Por ej., un bloque índice puede contener un encabezado pequeño que indique el nombre del archivo y un conjunto de las primeras 100 direcciones de bloque de disco. La siguiente dirección (la última palabra en el bloque índice) es *null* (para un archivo pequeño) o es un puntero a otro bloque índice (para un archivo grande).

Métodos de asignación

Asignación indexada

- **Índice multinivel**: un bloque índice de primer nivel apunta a un conjunto de bloques de índice de segundo nivel, que a su vez apuntan a los bloques de archivo.
  - Para acceder a un bloque, el SO usa el índice de primer nivel para encontrar un bloque índice de segundo nivel y luego usa ese bloque para encontrar el bloque de datos deseado.
  - Este enfoque podría continuar a un tercer o cuarto nivel, dependiendo del tamaño máximo de archivo deseado.
  - Con bloques de `4.096 bytes`, podríamos almacenar `1.024` punteros de `4 bytes` en un bloque índice. Dos niveles de índices permiten `1.048.576` bloques de datos y un tamaño de archivo de hasta `4 GB`.

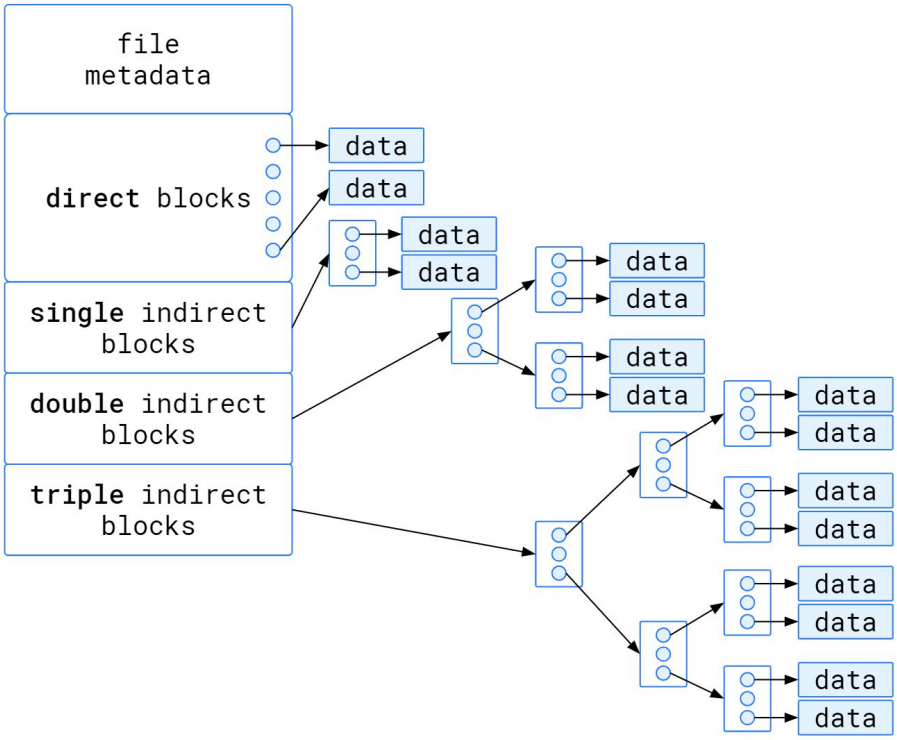
Métodos de asignación

Asignación indexada

- **Esquema combinado**: otra alternativa, utilizada en sistemas de archivos basados en `UNIX`, es mantener los primeros, por ej., `15` punteros del bloque índice en el `inodo` del archivo.
  - Los primeros `12` de estos punteros apuntan a `bloques directos`; es decir, contienen direcciones de bloques que contienen datos del archivo.
    - Archivos pequeños (no más de `12` bloques) no necesitan un bloque índice separado.
    - Si el tamaño del bloque es de `4 KB`, se accede directamente hasta `48 KB` de datos.
  - Los siguientes tres punteros apuntan a `bloques indirectos`.
  - El primero apunta a un `bloque indirecto`: bloque índice que no contiene datos, sino las direcciones de los bloques que sí contienen datos.
  - El segundo apunta a un `doble bloque indirecto`: contiene la dirección de un bloque que contiene las direcciones de los bloques que contienen punteros a bloques de datos.
  - El último puntero contiene la dirección de un `triple bloque indirecto`.

Métodos de asignación

Asignación indexada



Métodos de asignación

Asignación indexada

- Bajo este método, el número de bloques que se pueden asignar a un archivo excede la cantidad de espacio direccionable por los punteros de archivo de `4 bytes` utilizados por muchos SOs. Un puntero de archivo de `32 bits` alcanza solo 2^{32} `bytes`, o `4 GB`.
- Muchas implementaciones de `UNIX` y `Linux` ahora admiten punteros de archivos de `64 bits`, lo que permite que los archivos y sistemas de archivos tengan un tamaño de varios `exbibytes`.
- El sistema de archivos `ZFS` admite punteros de archivos de `128 bits`.
- Los esquemas de asignación indexada sufren algunos de los mismos problemas de rendimiento que la asignación enlazada.
- Los bloques de índice pueden almacenarse en la memoria caché, pero los bloques de datos pueden extenderse por todo un volumen.

Métodos de asignación

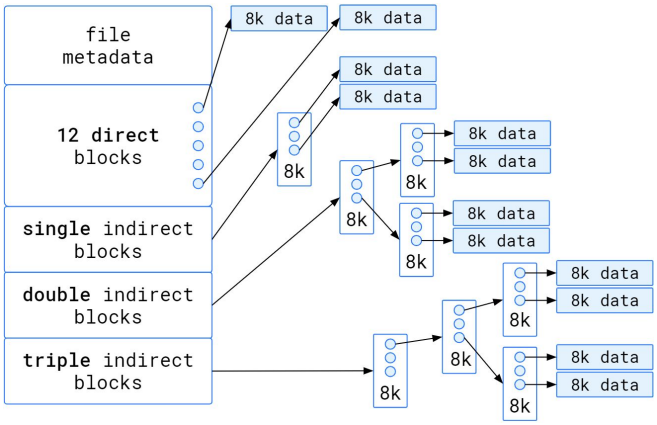
Asignación indexada: ejemplo

- Respuestas (asumiendo que todos los bloques índice están en memoria y no es necesario escribir éste en disco):
  - **a)Se agrega al inicio**: **1;** se escribe el nuevo bloque y se actualiza el bloque índice en memoria.
  - **b)Se agrega después del bloque del medio**: **1**; se escribe el nuevo bloque y se actualiza el bloque índice en memoria.
  - **c)Se agrega al final**: **1**; se escribe el nuevo bloque y se actualiza el índice en memoria. El nuevo bloque se escribe, haciendo que apunte al siguiente bloque. Actualizar el primer puntero de bloque en memoria.
  - **d)Se elimina del inicio**: **0**; el bloque se elimina del bloque índice en memoria.
  - **e)Se elimina del medio**: **0;** el bloque se elimina del bloque índice en memoria.
  - **f)Se elimina del final**: **0**; el bloque se elimina del bloque índice en memoria..

Métodos de asignación

Ejemplo cálculo de inodo

- Considere un **i-nodo** con **12** punteros a bloques directos, **1** puntero a un bloque indirecto simple, **1** puntero a un bloque indirecto doble y **1** puntero a un bloque indirecto triple. Si los bloques tienen un tamaño de **8-kilobyte** y los punteros son de **64 *bits***, ¿Cual es el tamaño máximo teórico de un archivo para cada uno de los siguientes casos?
- **a)** El archivo usa solo punteros directos.
- **b)** El archivo usa solo punteros directos y simples indirectos.
- **c)** El archivo usa solo punteros directos, simples y dobles indirectos.
- **d)** El archivo usa solo punteros directos, simples, dobles y triples indirectos.



Métodos de asignación

Ejemplo cálculo de inodo

- Los punteros directos apuntan a bloques de datos en disco y dado que cada bloque es de 8 *kilobytes*:
  - `Directos = 12 x 8 KB = 96 KB`
- Para el caso de los indirectos se debe saber cuantos punteros caben en un bloque de disco
  - `Cantidad de punteros = 8 KB ÷ 8 bytes = 1024 punteros a bloques.`
- Luego se puede calcular cada caso en particular:
  - `Simple indirecto = 1024 * 8KB = 8 MB = 8,388,608` *bytes*.
  - `Doble indirecto  = 1024 * 1024 * 8 KB = 8 GB = 8,589,934,592` *bytes*.
  - `Triple indirecto = 1024 * 1024 * 1024 * 8 KB = 8 TB = 8,796,093,022,208` *bytes*.

Métodos de asignación

Ejemplo cálculo de inodo

- Respuestas:
  - a) 96 KB = 98.304 *bytes*.
  - b) 8 MB + 96 KB = 8.486.912 *bytes*.
  - c) 8 GB + 8 MB + 96 KB = 8.598.421.504 *bytes*.
  - d) 8 TB + 8 GB + 8 MB + 96 KB = 8.804.691.443.712 bytes.

Gestión de espacio libre

Vector de *bits*

- Cada bloque está representado por `1 bit`. Si el bloque está libre, el `bit` es `1`; si el bloque está asignado, el `bit` es `0`.
- Por ejemplo, consideremos un disco donde los bloques `2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26` y `27` están libres y el resto de los bloques están asignados. El mapa de bits de espacio libre sería:
  - `00111100111111000110000001110000 0 …`
- La principal ventaja de este enfoque es su relativa simplicidad y su eficiencia para encontrar el primer bloque libre o `n` bloques libres consecutivos en el disco.
- Para encontrar el primer bloque libre se busca en cada palabra del mapa un valor distinto de `0`, ya que una palabra con `0` contiene solo `0 bits` y representa un conjunto de bloques asignados. La primera palabra que no es `0` se explora buscando el primer *bit* en `1`.
- El cálculo del número de bloque es:
  - `(number of bits per word) × (number of 0-value words) + offset of first 1 bit.`

Gestión de espacio libre

Lista enlazada

- Otro enfoque para la gestión del espacio libre es enlazar todos los bloques libres, manteniendo un puntero al primer bloque libre en una ubicación especial en el sistema de archivos y almacenándolo en memoria caché.
- Este primer bloque contiene un puntero al siguiente bloque libre, y así sucesivamente.
- Para el ej. anterior se mantiene un puntero al bloque `2` como el primer bloque libre. El bloque `2` contendría un puntero al bloque `3`, que apuntaría al bloque `4`, que apuntaría al bloque `5`, que apuntaría al bloque `8`, y así sucesivamente.
- Este esquema no es eficiente:
  - Para recorrer la lista, debemos leer cada bloque, lo que requiere un tiempo considerable de E/S en discos duros. Sin embargo, recorrer la lista libre no es una acción frecuente.
  - Por lo general, el SO simplemente necesita un bloque libre para poder asignarlo a un archivo, por lo que se utiliza el primer bloque de la lista libre.

Gestión de espacio libre

Lista enlazada