Sistemas de archivos: Estructura en el dispositivo

Gunnar Wolf

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Sistemas de archivos

- Gestión del espacio de almacenamiento
- Probablemente el rol con más visibilidad de los que cubren los sistemas operativos
  - Comprendido casi universalmente por los usuarios

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Bajando de nivel

- Entraremos directamente en detalles respecto a la organización de la información en un *dispositivo persistente*
- Principales estructuras y mecanismos para *implementar* un sistema de archivos *en un medio físico*

Discos y otras hierbas

- Cuando hablamos de almacenamiento, nos resulta natural hablar de *discos*
  - Disco: Medio giratorio, de acceso aleatorio
  - Típicamente magnético u óptico
- No siempre es el caso
  - Primeros años del cómputo: Medios secuenciales (tarjetas o cintas, (de papel o magnéticas)
- Hoy en día, tendencia a ir adoptando medios de *estado sólido*
  - Dispositivos sin partes móviles
  - Guardan la información en un tipo de *memoria* que no requiere corriente constante para mantener la información

Mantengámosnos idealizados

- El hablar de almacenamiento en estado sólido supone algunas consideraciones adicionales que no podemos perder de vista
- Por ahora, sigamos pensando en almacenamiento en *discos*
- Pronto veremos algunas de estas características diferenciadoras
  - Y el impacto que pueden tener en los sistemas de archivos

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Cómo se ve un disco?

- *Por ahora*, mantengamos la visión de un *disco* como un arreglo muy grande
- De bloques, todos ellos de tamaño fijo (fijado por el *hardware*)
- Cada bloque, *directamente direccionable*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Términos que se nos presentan

Para trabajar a *este nivel* de la implementación, presentemos unos cuantos términos

- Partición
- Volumen
- Metadatos
- I-nodo
- Mapa de bits de espacio libre

Partición

- Subdivisión de un disco
- El administrador del sistema puede emplearlas para definir la forma en que se emplea el espacio disponible
- Un disco puede tener cero, una o varias particiones
- Cada partición puede contener un sistema de archivos independiente

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Volumen

- Colección de bloques en el disco
- *Inicializados* con un sistema de archivos
- Pueden presentarse al usuario como una *unidad*
- Típicamente coincide con una *partición*
  - Aunque veremos algunos casos en que no es así
- Se *describe* ante el sistema operativo en el *bloque de control de volumen*
  - O *superbloque* (Unix), *Tabla maestra de archivos* (*MFT*, Windows)

Directorio raiz

- Estructura base con la relación entre nombres de archivo y números de *i-nodo*
- Típicamente (por rendimiento) sólo almacena los archivos del *primer nivel jerárquico*
  - Directorios interiores, referenciados desde éste

Metadatos

- Toda la información que haya *acerca de* un archivo
  - Que *no sean* los datos pertenecientes al archivo mismo
- En un sistema de archivos estándar:
  - Nombre
  - Tipo
  - Tamaño
  - Fechas de creación, último acceso y modificación
  - Ubicación en disco
- Ojo: *¡No están en un sólo lugar!*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

I-nodo

- Viene de la expresión *nodo de información*
- En sistemas Windows, frecuentemente se les llama *Bloque de control del archivo* (FCB)
- La estructura en disco que guarda *la mayor parte* de los metadatos da cada archivo
- Proporciona un vínculo entre la *entrada en el directorio* y los contenidos del archivo
- Incluye permisos, propietario, tipo de archivo, fechas
- Incluye también la *relación de bloques* que ocupa el archivo en disco

Mapa de bits (*bitmap*) de espacio libre

- Auxiliar para gestionar el espacio *libre* del disco
- Es una de varias estrategias para presentar esta información (no la única)
- Representa el estado de un bloque entero por cada bit
  - Bastante compacto — con bloques de 4096 bytes, 100GB pueden representarse en 3MB ($\frac{100 \times 10^9}{4096 \times 8}$)
  - Razonable para mantener en memoria para un sistema de escritorio actual
  - Y ha ido creciendo a una razón aceptable desde los *viejos tiempos* (p. ej., 10KB para 40MB en sectores de 512 bytes, común hacia fines de los 1980s)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Ejemplificaremos con FAT

- Para esta sección, iremos ejemplificando los conceptos refiriéndonos a una familia de sistema de archivos sencillo y ampliamente utilizada
  - FAT — FAT12, FAT16, FAT32
- Se origina a fines de los 1970, fue adoptado por diversas arquitecturas de los 1980
  - Incluyendo, claro, IBM con MS-DOS
- Sigue siendo uno de los sistemas más empleados del mundo

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Base para emplear un sistema de archivos

- El volumen es descrito por el superbloque
  - *No contiene* a los archivos, ni al directorio, ni estructuras que apunten hacia ellos
  - Sólo información acerca del volumen
- ¿Qué *tipo* de sistema de archivos es?
- Descripción básica del sistema de archivos
  - Extensión del sistema
  - Tamaño de los *bloques lógicos*
  - Etiqueta que lo describa ante el usuario

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Bloques físicos y lógicos

- El tamaño de los *bloques físicos* es establecido por el hardware
  - Tamaño de las transferencias del controlador al disco
- Típicamente se *agrupan* en *bloques lógicos* (también llamados *grupos* o *clusters*)
  - Por razones de rendimiento
  - Para alcanzar a direccionar mayor espacio
  - Revisaremos este tema al hablar de el *directorio*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Superbloque: Estructura repetida

- El *superbloque* es fundamental para poder abrir el sistema de archivos
- Es tan importante (y tan poco frecuentemente modificado) que se debe proteger de toda posible corrupción
- Muchos sistemas de archivos mantienen *copias adicionales* del superbloque dispersas a lo largo del sistema de archivos

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

FAT: El superbloque

- No guarda *tanta* información en un sistema así de simple
- Indica principalmente la *generación* del sistema de archivos (FAT12, FAT16, FAT32)
  - 12, 16 o 32 bits para referenciar a cada uno de los bloques lógicos o *clusters*
- Indica el tamaño empleado en este volumen para los *clusters*
  - Desde 2 y hasta 32KB

Volúmenes crudos

- Una de los principales tareas de un sistema operativo es la organización del espacio de almacenamiento en sistemas de archivos
- Pero en algunos casos, puede tener sentido *no* aprovechar esta característica
  - Principalmente, por rendimiento
  - Emplear un *dispositivo orientado a bloques* sin emplear un sistema de archivos se denomina emplear *dispositivos crudos* o *dispositivos en crudo*
- ¿Cuándo se usa?

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Volúmenes crudos como *espacio de intercambio*

- La gestión de la *memoria virtual* puede beneficiarse de *no* cruzar por la abstracción del sistema operativo
- El espacio de intercambio (*swap*) es manejado directamente, no a través de sistemas de archivos
- ¿Pero no es el gestor de memoria virtual parte del sistema operativo?
  - El uso del dispositivo es en crudo, incluso si es desde espacio del sistema operativo
  - Además, en un sistema *microkernel* puede ejecutarse como proceso de usuario

Volúmenes crudos para bases de datos

- Varios gestores de bases de datos relacionales administran volúmenes muy grandes de datos
  - Datos estrictamente estructurados
- Algunos gestores pueden optimizar los accesos al disco evitando las capas de abstracción del sistema operativo
  - Recomendado por Oracle, MaxDB, DB2, ...
- Sin embargo, muchos gestores han abandonado esta modalidad
  - Mayor complejidad de administración
  - Mejoría de rendimiento muy limitada
- Afirmación polémica, se presta a discusión e investigación

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Rol del directorio

- Estructura que relaciona a los archivos como son presentados al usuario –identificados por una ruta y un nombre
  - Con las estructuras que los describen ante el sistema operativo — Los *i-nodos*.
- A lo largo de la historia de los sistemas de archivos, se han implementado muy distintos esquemas de organización

El directorio raiz

- Una vez que montamos un sistema de archivos, todas las referencias a archivos dentro de éste deben pasar a través del directorio raiz
  - Y probablemente directorios adicionales
- Está siempre en una *ubicación bien conocida* en el disco, típicamente al inicio
  - Por su frecuencia de uso, en los 1980, los diseñadores de AmigaOS decidieron ponerlo *al centro*
  - Sector 40 (de 80 que tenían los floppies)
  - ¿Resultado? Reducción a la mitad de la demora por movimiento de cabezas
- Sistemas Unix modernos: Esparcir los subdirectorios como mini-sistemas de archivos sobre todo el disco

Formato del nombre de los archivos

- El directorio determina formato y restricciones para los nombres de archivos y directorios
- En un sistema moderno, es común que un archivo pueda tener hasta 256 caracteres arbitrarios
  - Incluyendo mayúsculas, minúsculas, espacios, acentos, ...
- Hay sistemas de archivos *sensibles a mayúsculas*, como los derivados de Unix (`ejemplo.txt` $\neq$ `Ejemplo.TXT`) o *insensibles a mayúsculas*, como los derivados de MS-DOS (`ejemplo.txt` = `Ejemplo.TXT`)

FAT: Entrada de directorio

- Esquema claramente antiguo de nombres de archivos
  - 8 caracteres (+3 para la extensión)
  - Expresado en mayúsculas; muy pocos caracteres legales no alfanuméricos
- FAT *no separa* entre directorio e i-nodo, como hoy es la norma
  - Cada entrada en el directorio mide exactamente 32 bytes
- Como ha tenido tantas generaciones, varios de los campos que veremos a continuación tienen más de un significado

FAT: Entrada de directorio

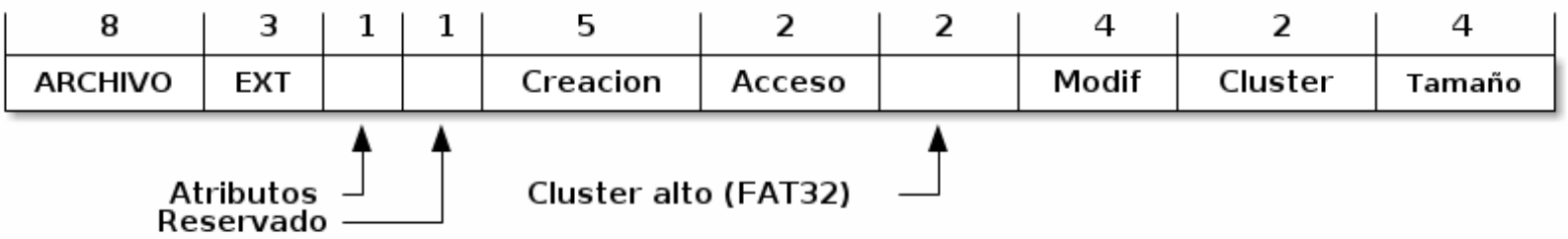

Figura: Formato de la entrada del directorio bajo FAT32 (longitudes en bytes) (Mohammed, 2007)

FAT: La extensión VFAT

- Introducida junto con Windows 95 para permitir nombres de archivo largos
  - Sin romper compatibilidad con la amplia base de sistemas ya existentes
- El nombre *real* de un archivo sigue siendo en formato 8.3
  - Se pueden agregar *entradas adicionales* al directorio, usando un atributo de *etiqueta de volumen*
  - Los sistemas MS-DOS ignoran dichas entradas

FAT: La extensión VFAT

Figura: Entradas de directorio representando archivos (con y sin nombres largos) bajo VFAT

FAT: Tamaño máximo del directorio

- Bajo FAT12 y FAT16, el *tamaño del directorio raiz* está limitado
- Está ubicado entre las FAT y el inicio de los datos
  - Esto es, en los 14 bloques desde el 20 hasta el 33
- En un floppy, le caben hasta $512 \times 14 = 7168$ bytes
  - A 32 bytes por entrada de FAT, $\frac{7168}{32} = 224$ entradas como máximo
- No está mal *para un floppy*...
  - ¿Y considerando VFAT? El espacio disponible se reduce fuertemente
- Por eso, FAT32 ya trata al directorio raiz como parte de la *región de datos*
  - Al igual que lo son todos los subdirectorios

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Directorios, ordenamiento y velocidad

- El directorio es una estructura *usada muy frecuentemente*
  - Hay que optimizarla a su principal tipo de uso
- Muy frecuentemente ocupa distintos sectores / clusters
  - Si guardáramos el directorio *ordenado*, cada modificación sería muy cara
    - Crear, eliminar, renombrar archivos
  - Obligaría a reescribir el directorio *entero*
- El directorio típicamente se graba *sin ordenar*, y en caso de requerirse presentar ordenado, se hace *en espacio de usuario*
  - Pero si el acceso típico es de lectura, puede tener sentido emplear un *árbol* que asegure ordenamiento
  - O un *hash*, que asegure encontrar rápido los datos de *un archivo* en particular

FAT: Eliminando archivos

- Cuando se crea un directorio, todas las entradas llevan en el campo de nombre caracteres `00` (`NUL`)
- Dado que el tamaño del directorio es limitado, tenemos que poder *reaprovechar* las entradas eliminadas del directorio
- Para hacer esto, podríamos *marcar como eliminada* a una de las entradas, y mantenerla disponible para guardar sobre ella *el siguiente archivo* que sea creado

FAT: Eliminando archivos

- En FAT, *eliminar* un archivo es muy barato
  - Pero *reaprovecharlo* es más caro
- Una entrada en el directorio se muestra como *eliminada* con sólo reemplazar el primer caracter de su nombre con `0xE5`
  - Esto permite *des-borrar* el archivo con sólo especificar su nombre de archivo
  - Si la entrada no ha sido re-ocupada
- Re-ocupar la entrada *requiere marcar el espacio del archivo como libre*
  - Lo explicaremos en un momento, una vez que veamos el funcionamiento de las *tablas de asignación de archivos*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Sabemos *qué hay*, pero... *¿Dónde está?*

- Como vimos al hablar de directorios, éstos sólo relacionan al *nombre de archivo* con su respectivo *i-nodo*
  - En el caso de FAT, incluyen *algo* de metadatos, y apuntan al *principio* de su sección de datos
- Pero, ¿qué mecanismos hay para *asignar, gestionar y manejar* el espacio que ocupa cada uno de los archivos?

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Asignación contigua

- Esquema empleado en los sistemas de archivos *muy* antiguos
  - Y en los diseñados para no modificación — p.ej. ISO9660
- Bastaría con la información que tiene el directorio FAT
  - *Punto* de inicio del archivo
  - Longitud total
- Ventajas:
  - Lo más simple de implementar
  - La mejor velocidad (minimiza movimientos de cabezas)
- Desventajas:
  - Muy sensible a fragmentación externa
  - Un archivo no puede *crecer*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Asignación contigua de archivos

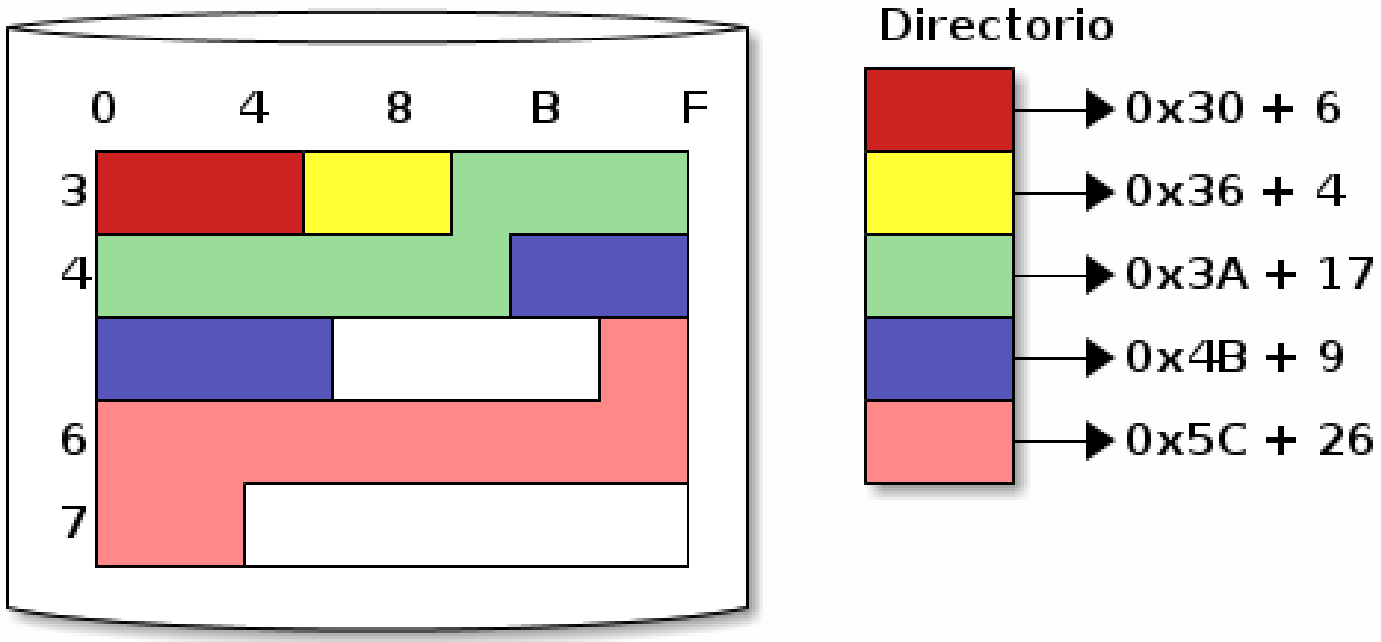

Figura: Asignción contigua de archivos: Directorio con inicio y longitud

Asignación ligada

- Puede proveerse mucho más flexibilidad al usuario si se permite la fragmentación
- Cada *entrada en el directorio* apunta a un *grupo* (*cluster*)
  - Cada cluster apunta al siguiente
  - El último cluster indica que el archivo terminó
- ¿Dónde se apunta al siguiente cluster?
  - Puede reservarse un espacio al final de cada cluster (p.ej. clusters de 2044 bytes, reservando 4 bytes para el apuntador al siguiente cluster)
  - Puede crearse una *tabla maestra* con la asignación de clusters *por bitmap*
- Más *sobrecarga administrativa* que la asignación contigua
  - Se pierde más espacio *apuntando al siguiente bloque*
- FAT es un ejemplo de asignación ligada

Asignación ligada

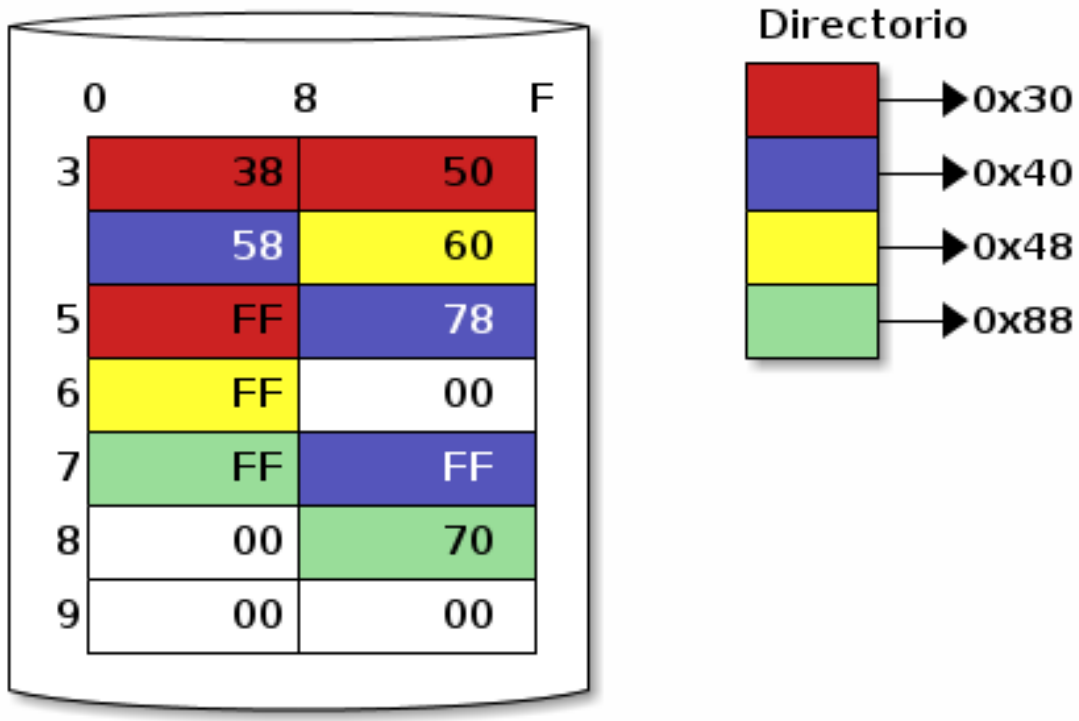

Figura: Asignción ligada de archivos: Directorio con apuntador sólo al primer cluster

Revisando la asignación ligada

- Ventajas:
  - Desaparece la fragmentación externa
  - *¡Ojo!* Eso no significa que no haya lo que *los usuarios conocen* por fragmentación — Que es *muy nocivo* para el rendimiento
- Desventajas
  - Mucho movimiento de cabezas → Rendimiento penalizado
  - Si el apuntador está al final de los datos, *imposibilidad de trabajar en acceso aleatorio*
  - Fragilidad de metadatos: Si se pierde/corrompe un apuntador, los datos de dos archivos podrían resultar afectados
    - El archivo dueño de ese bloque y el archivo al que ahora apuntaría por error
    - Podría prevenirse/reducirse: Lista *doblemente ligada*, incluyendo *apuntador al final*

Asignación indexada

- Crea una estructura intermedia entre el directorio y la asignación de bloques, exclusiva por archivo: El *nodo de información* (*i-nodo*)
- Cada i-nodo guarda la lista de bloques del archivo
  - Permite acceso aleatorio eficiente a todo el archivo
  - Reduce la probabilidad de *corrupción de apuntadores*
- Potencialmente, mayor sobrecarga administrativa
  - Al crear un archivo, se crea un i-nodo completo
  - Si el archivo es pequeño, puede que el i-nodo sólo apunte a unos pocos clusters
  - Pero el i-nodo mismo ocupa un cluster completo

Eficiencia de caché por tipo de asignación

- Mejoría en la eficiencia del caché
  - Con la *asignación contigua*, basta tener en caché el directorio para conocer todas las ubicaciones
  - Con *asignación ligada*, incluso con una tabla maestra, hay que hacer caché de *toda la tabla*
  - Con asignación indexada, basta hacer caché de *lo que nos importa* (los archivos actualmente abiertos)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Asignación indexada

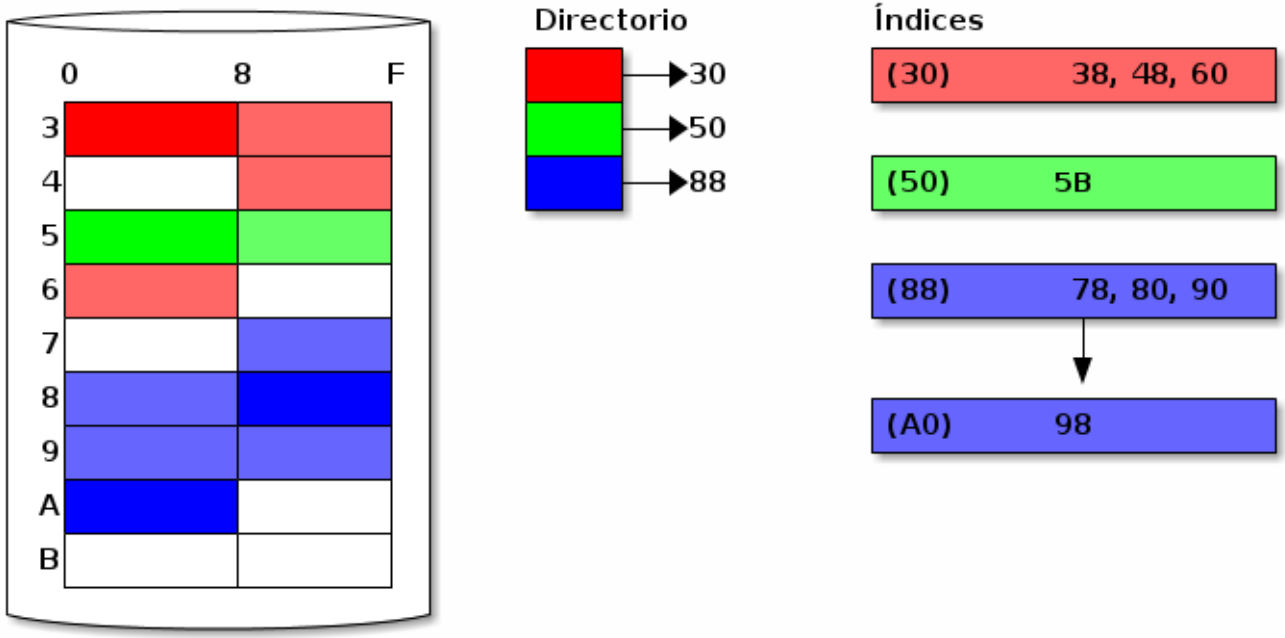

Figura: Asignción indexada de archivos: Directorio con apuntador al i-nodo (llevado a un i-nodo de tamaño extremadamente ineficiente)

Apuntadores directos e indirectos

- Al emplear asignación indexada, crece el espacio disponible en el directorio
  - Prácticamente todos los metadatos se van al i-nodo
  - Pero el espacio del i-nodo es también finito
- Las direcciones en disco para un archivo *razonablemente chico* pueden caber completamente en un sólo i-nodo → *Apuntadores directos*
  - P.ej. archivos *rojo* y *verde* del ejemplo anterior
- Cuando un archivo tiene más bloques que los que caben en un i-nodo, éste asigna clusters adicionales con los *niveles de indirección* requeridos
  - P.ej. archivo *azul* del ejemplo anterior
- Pero los niveles de indirección pueden *anidarse*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Apuntadores de indirección a varios niveles

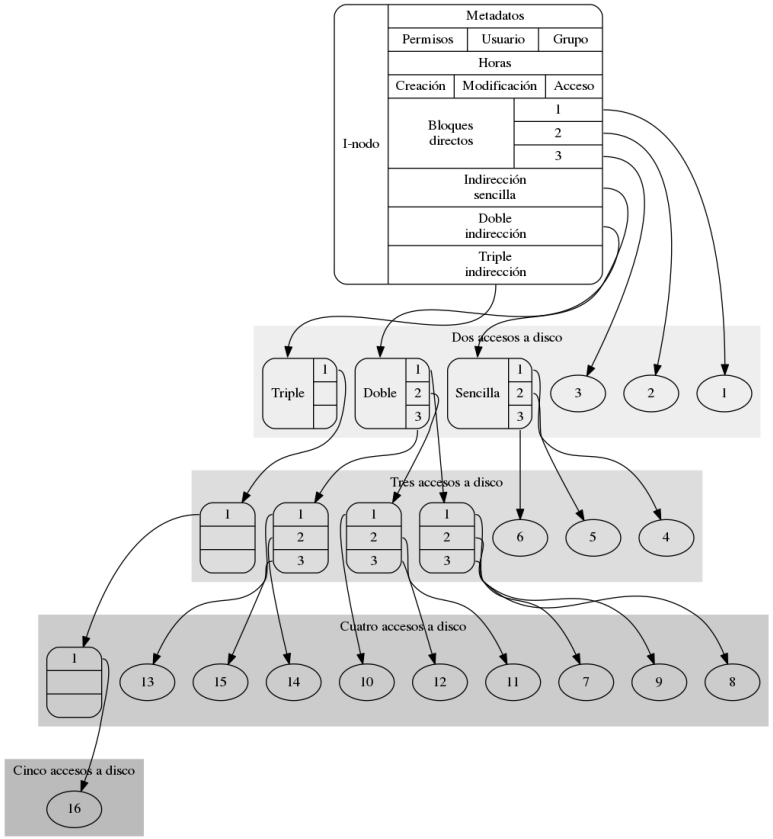

Figura: Estructura típica de un i-nodo en Unix, mostrando además el número de accesos a disco necesario para cada cluster

La lógica de la indirección multinivel

- Varios sistemas de archivos (la idea apareció originalmente en UFS, Unix de los 1980s) buscan dar un rendimiento *acorde al tamaño del archivo*
  - Aumentando los niveles de indirección según crece el archivo
- Por ejemplo:
  - Si en el bloque del i-nodo (que contiene también los metadatos) caben 100 apuntadores
  - Y el tamaño de cluster fuera de 4K
  - En un cluster vacío caben 128 apuntadores ($\frac{4096}{32}$)
- Reservamos los *últimos 3 apuntadores* para los *bloques indirectos*

Ejemplo de indirección multinivel (1)

- Un archivo de hasta $(100 - 3) \times 4KB = 388KB$ puede ser representado directamente en el i-nodo
  - Un sólo acceso a disco para obtener los clusters
- Un archivo de hasta $(100 - 3 + 128) \times 4KB = 900KB$ puede representarse con el bloque de indirección sencilla
  - Dos accesos a disco

Ejemplo de indirección multinivel (2)

- Con el bloque de doble indirección, llegamos a $(100 - 3 + 128 + (128 \times 128)) \times 4KB = 66436KB$
  - Hasta 131 accesos a disco
  - ¡Va valiendo la pena que los múltiples niveles sean adyacentes!
- Empleando triple indirección, hasta $(100 - 3 + 128 + (128 \times 128) + (128 \times 128 \times 128)) \times 2KB = 8455044 \approx 8GB$
  - Pero hasta 16516 accesos a disco

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

FAT: La tabla de asignación de archivos

- El directorio de FAT apunta al *primer* cluster que ocupa cada uno de los archivos → FAT maneja asignación ligada
- Tiene también un campo que indica la *longitud total* del archivo
- Sin embargo, no es tan simple como indicar inicio + desplazamiento
- La estructura fundamental de FAT (incluso *da su nombre* al sistema de archivos) es la *Tabla de Asignación de Archivos* → *File Allocation Table*
  - Tan importante es esta estructura que se mantiene *por duplicado* (triplicado en FAT12)
  - Ante daño físico — Los discos antes eran mucho menos confiables

FAT: Leyendo la tabla de asignación de archivos

- Cada entrada de la FAT mide lo que la longitud *correspondiente a su versión* (12, 16 o 32 bits), y puede tener uno de cuatro valores posibles:

  Libre `0x000`, `0x0000` o `0x00000000`
  - El *cluster* está libre
  - Puede ser empleado por el sistema

  Siguiente Valores hasta `0xFF6`, `0xFFF6` o `0xFFFFFFF6`
  - Ubicación del *siguiente cluster* del archivo

  Dañado `0xFF7`, `0xFFF7` o `0xFFFFFFF7`
  - El cluster es un espacio del disco dañado
  - No debe ser utilizado para almacenar datos

  Fin `0xFFF`, `0xFFFF` o `0xFFFFFFFF`
  - Último cluster de un archivo.

FAT: Fragmentación por diseño

- Por flexibilidad, FAT permite la *fragmentación de archivos*
  - Queda claro por el diseño del sistema: El descriptor en FAT de cada cluster *debe apuntar* al siguiente
  - Esto significa que muy frecuentemente los archivos *no estarán contiguos*

FAT: Vista de la tabla de asignación de archivos

00004000	f0 ff ff 0f ff ff ff 0f f8 ff ff 0f ff ff ff 0f	....
00004010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00004020	09 00 00 00 0a 00 00 00 0b 00 00 00 0c 00 00 00	....
00004030	0d 00 00 00 0e 00 00 00 0f 00 00 00 10 00 00 00	....
00004040	11 00 00 00 12 00 00 00 13 00 00 00 ff ff ff 0f	....
00004050	15 00 00 00 16 00 00 00 17 00 00 00 18 00 00 00	....
00004060	19 00 00 00 1a 00 00 00 1b 00 00 00 1c 00 00 00	....
00004070	1d 00 00 00 1e 00 00 00 1f 00 00 00 20 00 00 00	....
00004080	21 00 00 00 ff 00 ff ff 0f 00 00 00 00 00 00 00	!... ....
00004090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

FAT ID	Indicación de fin de cadena
Clusters vacíos	El directorio raiz cabe en un sólo cluster
Primer cadena (un solo cluster: 0003)	Segunda cadena (12 clusters: 0008 a 0013)
Tercera cadena (16 clusters: 0014 a 0021)	

Figura: Ejemplo de entradas en la tabla de asignación de archivos bajo FAT32

¿Cómo lidian con la fragmentación otros sistemas?

- La fragmentación es uno de los puntos más débiles de FAT (y los sistemas operativos de su época)
- Una estrategia seguida por varios sistemas tipo Unix es la de los *grupos de asignación*
- Los directorios (y los i-nodos dentro de éste) son ubicados *esparcidos por el disco*
  - Los archivos pertenecientes a un directorio son asignados *cerca* de éste
  - Esto garantiza menor desplazamiento de cabezas entre el directorio, el i-nodo y los datos
  - Si no hay espacio para hacer una asignación contigua, los datos *pueden* guardarse *lejos* de su directorio
    - Esto pasa implícitamente al usar *ligas duras*

Evitando la fragmentación por grupos

- La fragmentación *se produce*
  - Pero es mucho menos nociva
  - Obtenemos sus ventajas con *un mínimo* de sus desventajas
- Importancia del espacio vacío
  - Conforme se va llenando el disco, es más difícil encontrar espacio para seguir esta estrategia
- Reserva de espacio
  - Unix típicamente *reserva* $\approx 5\,\%$ *para uso del administrador*
  - Permite recuperar de situaciones críticas
  - Busca también no llegar a los umbrales de saturación descritos

FAT: ¿Y los subdirectorios?

- Vimos cómo está estructurado el *directorio raiz*
- Vimos también que siempre está en un *lugar bien conocido* en el disco
- ¿Qué hay con los subdirectorios?

FAT: ¿Y los subdirectorios?

- Vimos cómo está estructurado el *directorio raiz*
- Vimos también que siempre está en un *lugar bien conocido* en el disco
- ¿Qué hay con los subdirectorios?

- Un subdirectorio es *sencillamente un archivo*
  - De *tipo especial*: El byte de *atributos* (`0x0B`) vale 16
  - Es almacenado en disco *exactamente* como un archivo

FAT: Los directorios y la fragmentación (1)

- Si un subdirectorio es un archivo especial, está sujeto a la fragmentación
- Cuando se asigna espacio para un subdirectorio, se asigna un sólo cluster
  - 2048 bytes hasta 32768 bytes (64 a 1024 entradas)
  - La ubicación del archivo apunta a donde éste está alojado; la FAT guarda `0xFFFF`

FAT: Los directorios y la fragmentación (2)

- Cuando se llena este cluster, se agrega otro al final del directorio
  - En la entrada en la FAT se apunta al nuevo sector; la del nuevo sector guarda `0xFFFF`
- La lectura de un directorio con muchas entradas puede requerir muchos movimientos de cabeza
  - Entre más grande el tamaño de cluster, más entradas por cluster $\rightarrow$ menos movimiento de cabezas

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

FAT: ¿Qué problemas podría haber?

- FAT es un sistema relativamente frágil
- Puede presentarse corrupción de metadatos
  - Particularmente corrupción de la estructura de las FAT
- ¿Qué hacen los programas `CHKDSK` o `SCANDISK`?
  - Verifican que ambas copias del FAT *concuerden*
  - *Desarrollan* cada una de las *cadenas* que describen un archivo, buscando inconsistencias
- Principales inconsistencias
  - Archivos cruzados (*Cross-linked file*)
  - Cadenas perdidas o *huérfanas*
- ¿Por qué ocurren estos problemas?

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

FAT: Principales inconsistencias

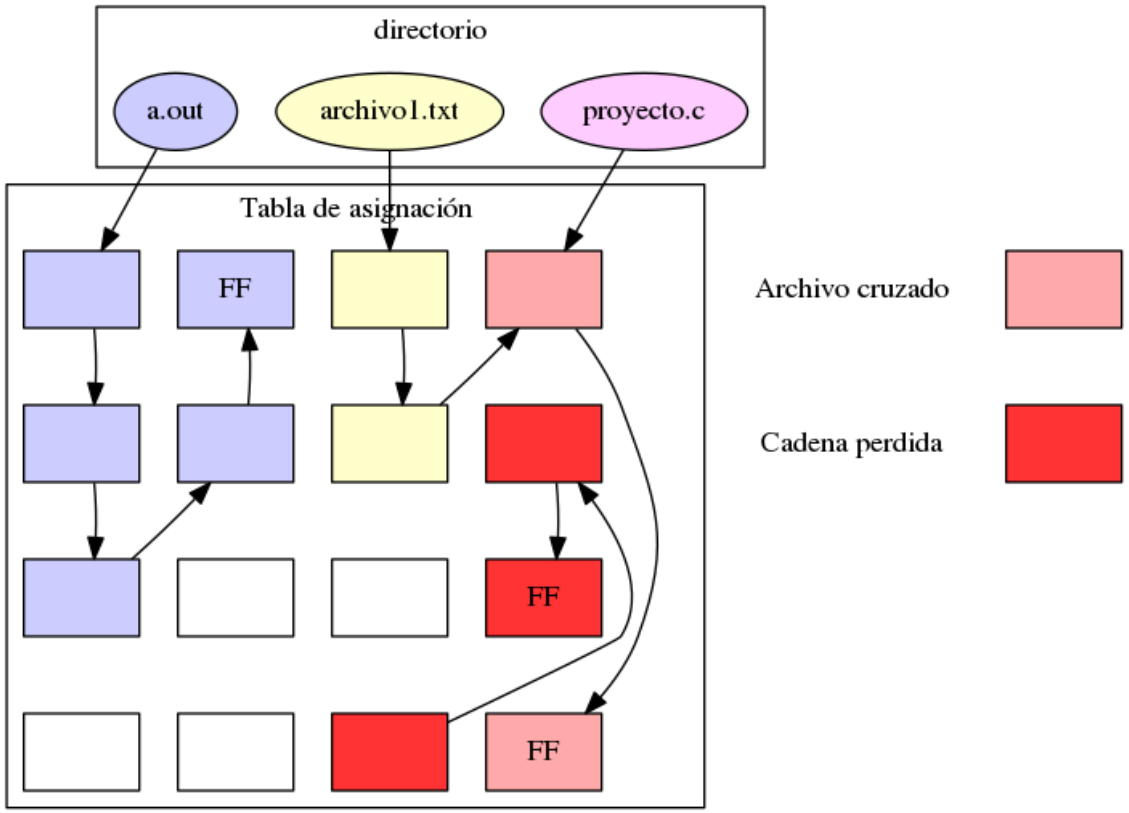

Figura: Principales inconsistencias que pueden presentarse en los sistemas de archivos tipo FAT

Problema generalizado a los distintos sistemas de archivos

- Prácticamente todos los sistemas de archivos tienen que cuidar estos aspectos
  - Distintos *síntomas* — A diferente organización de la información, diferentes probables fallos
- Los *controladores de disco inteligentes* agravan este problema
  - Caché incorporado
  - Notifican escrituras exitosas al sistema operativo antes de haberlas ejecutado

La coherencia del sistema de archivos

- Cada operación en el sistema de archivos requiere varias modificaciones al disco
- Por ejemplo, para crear un archivo:
  - Crear la entrada en directorio
  - Encontrar los clusters a emplear
  - Marcarlos en la tabla
  - Guardar los datos
- Si hay un fallo, corte de corriente, o el usuario retira el dispositivo antes de tiempo, puede que esta información se haya registrado sólo *parcialmente*
  - El sistema de archivos *presenta inconsistencias* o *está en un estado inconsistente*

Coherencia de datos *vs.* metadatos

- Hay que mantener en mente la separación entre *datos* y *metadatos*
- Si todos los cambios en las estructuras se realizaron, pero *los datos del usuario* no se registraron al disco, *no estamos* en un estado inconsistente
  - Sí, hubo pérdida de información
  - Pero *la estructura* del sistema de archivos no presenta ningún problema
  - No pone en riesgo ninguna *operación posterior*, ni implica a otros archivos

Verificación de la integridad

- Parte importante de los distintos sistemas de archivos son los *programas de verificación de integridad*
  - En Windows, `CHKDSK` y `SCANDISK`
  - En Unix, `fsck.vfat`, `fsck.ext2`, etc.
- Hacen un *barrido* del sistema de archivos, buscando evidencias de inconsistencia
  - Siguen todas las cadenas de clusters de archivos o tablas de i-nodos y verificando que no haya *archivos cruzados* (compartiendo erróneamente bloques)
  - Verifican que todos los directorios sean alcanzables
  - Recalculando espacio vacío, bitmap de libres, etc.
- Son siempre procesos *intensivos* y complejos
  - Y deben ejecutarse con el sistema de archivos *fuera de línea* (o en *sólo lectura*)

Evitando o recuperando de estados inconsistentes

Hay dos estrategias principales para enfrentarse a los estados inconsistentes:

- Actualizaciones suaves (*soft updates*)
- Sistemas de archivo con bitácora (*journaling file systems*)
- Sistemas de archivo estructurados en bitácora (*log-structured file systems*)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Actualizaciones suaves (*soft updates*)

- Organiza las escrituras a disco de modo que el estado resultante *no pueda* ser inconsistente
  - Permite inconsistencias *no destructivas*: Marcar como asignado espacio libre
- Verifica *dependencias* antes de escribir
  - Por ejemplo: No libera el espacio de un archivo antes de haber marcado su entrada como eliminada del directorio
- El programa de verificación (`fsck`) se vuelve una tarea *ejecutable en el fondo*, principalmente actuando como *recolector de basura*
  - Busca espacio marcado como asignado, pero no referenciado

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Actualizaciones suaves y archivos temporales

- Puede ahorrarse por completo muchas escrituras a disco
  - La creación de un archivo temporal (creación/obtención de descriptor/remoción) no tiene siquiera que llegar al disco
  - El proceso puede escribir sus datos en *espacio libre no-referenciado*, la estructura del sistema de archivos no se altera

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Uso de actualizaciones suaves

- La idea fue presentada hacia 1999, e implementada en FreeBSD hacia 2002
- Está implementado en UFS, empleado por varios sistemas operativos de la familia *BSD
- Pero no ha sido adoptado en otras familias de sistemas operativos
  - NetBSD retiró el soporte en 2012 (v6.0), *prefiriendo* el empleo de bitácora

Sistemas de archivos con bitácora (*journaling file systems*)

- Antes de efectuar cualquier operación *de metadatos* (una *transacción*), ésta se graba a una bitácora
  - Una transacción puede comprender varias operaciones independientes
  - La bitácora es típicamente una *lista ligada circular*
  - En algunas implementaciones, también los *datos* mismos (aunque es poco común)
- Periódicamente, se avanza por la bitácora, grabando las estructuras a disco, y avanzando el apuntador

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Sistemas de archivos con bitácora (*journaling file systems*)

- En caso de fallo, el sistema operativo:
  - Lee dónde quedó el apuntador
  - *Avanza* las operaciones faltantes
  - *Converge* rápidamente a un sistema de archivos estable
- Todas las operaciones deben ser *idempotentes*
  - Su ejecución repetida no debe alterar el resultado

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Sistemas de archivos con bitácora (*journaling file systems*)

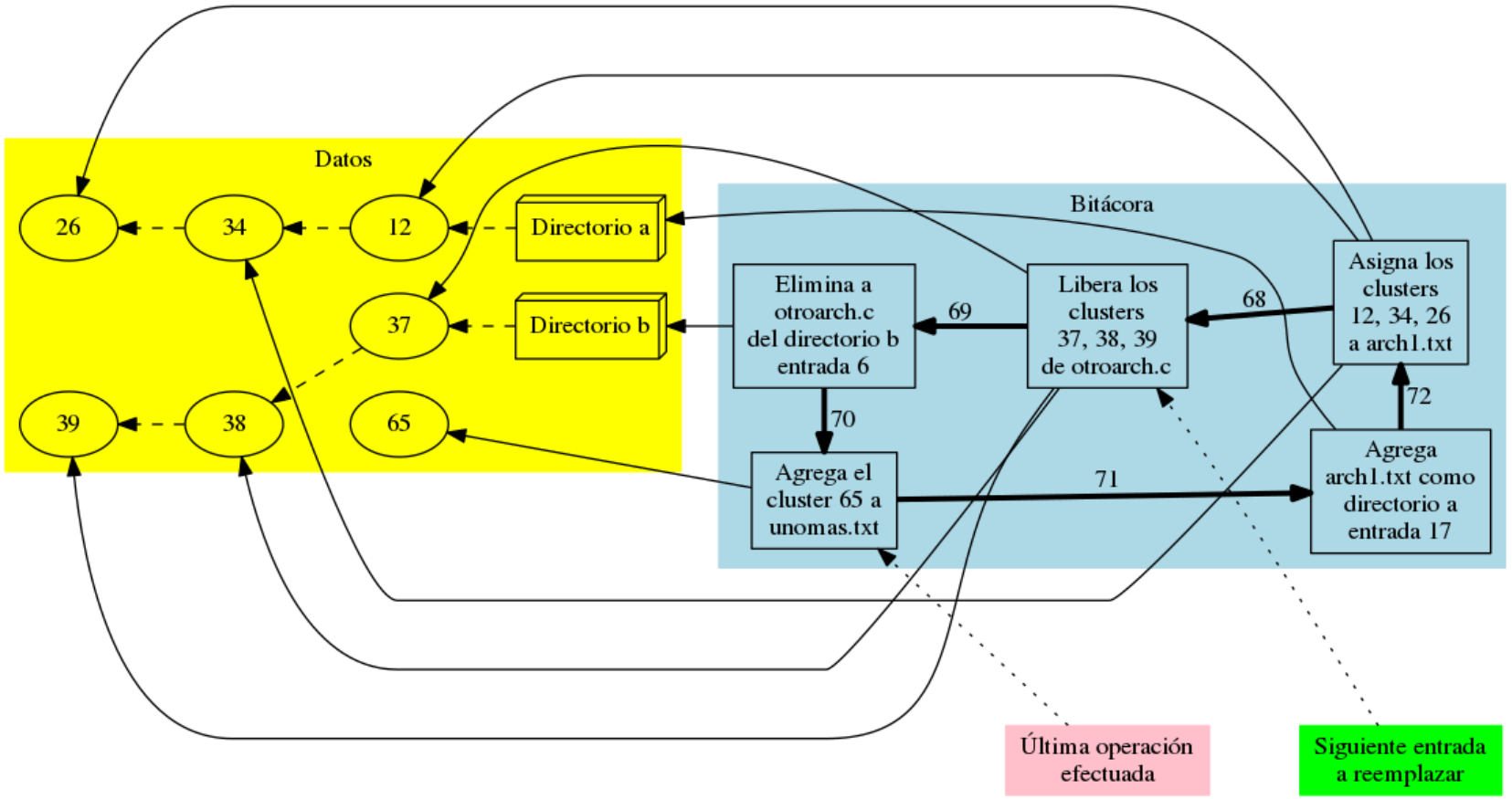

Figura: Sistema de archivos con bitácora

Sistemas de archivos con bitácora (*journaling file systems*)

- La bitácora *no incluye* (por lo general) datos, *sólo metadatos*
  - Por rendimiento
  - Porque la bitácora tendría que ser mucho más grande
- Es el esquema más empleado hoy en día
- Presente en casi todos los sistemas de archivos modernos
- Ojo: *No exime* de la verificación de sistema de archivos
  - Sigue siendo necesaria periódicamente (periodos largos)
  - Aunque no como procedimiento habitual tras una detención abrupta
  - Se recomienda principalmente para recuperar ante efectos de bugs en la implementación

Sistemas estructurados en bitácora (*log-structured file systems*)

- Llevan el concepto de *bitácora* al límite
- En vez de tener un área reservada para la bitácora, *el total* del sistema de archivos es una gran bitácora
- Siguen una organización radicalmente diferente del resto de los sistemas de archivo
  - Dependen de un *caché agresivo* para la lectura
  - Orientados a facilitar las escrituras, haciéndolas secuenciales
- Aptos sólo para ciertos tipos de carga
  - Tremendamente ineficientes para otros

Sistemas estructurados en bitácora (*log-structured file systems*)

- Siguen siendo en buena medida sujetos actualmente a investigación
- Hay varias implementaciones, pero casi todas se han detenido/abandonado
  - O reducido fuertemente su ritmo de desarrollo
- Pero han llevado al desarrollo de conceptos importantes que hoy se están aplicando en sistemas de archivo *más estándar*
  - Especialmente en el área de sistemas de archivos orientados a *dispositivos no-magnéticos o no-rotativos*