udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA

Sistema de archivos: Partes internas

Adín Ramírez
adin.ramirez@mail.udp.cl

Sistemas Operativos (CIT2003-1)
1er Semestre 2015

Adaptado de Dave Eckhardt

Puntos a discutir

- Capas de código de sistema de archivos (abstracto)
- Discos y estructura de memoria
- Capa de indirección de Unix: “VFS”
- Directorios
- Estrategias de reserva de bloques y espacio libre
- Trucos de cache
- Recuperación y backups

Capas de sistema de archivos

- Drivers de dispositivos
  - Escribir y leer (discos, sector de arranque, conteo)
- Bloque I/O
  - Escribir y leer (partición, bloque) —almacenado en el cache
- Archivo I/O
  - Escribir y leer (archivo, bloque)
- Sistema archivos
  - Administrar directorios, espacio libre

Espacio de nombres de sistema de archivos múltiples

- Particionar, nombre para los dispositivos
- Mounting
- Unificar distintos tipos de sistemas de archivos
  - UFS
  - ext2fs
  - ext3fs
  - zfs
  - FAT
  - 9660

Particionar discos

- Dividir el disco en particiones, rodajas, mini-discos, ...
  - MBR (PC): 4 "particiones" —Windows, FreeBSD, Plan 9, ...
  - APM (Mac): "volumenes" —puede dividir: OS 9, OS X, archivos de usuarios, ...
  - GPT (nuevo, multi-plataforma) —muchas particiones, nombres largos
- Pegar discos en volumenes o discos lógicos
- Una partición (de un disco o un volumen) puede contener
  - Área de paginación
    - Indexar por estructuras en memoria
    - "basura aleatoria" cuando el SO se apaga
  - Sistema de archivos
    - Distribución de bloques: número de archivo $\Rightarrow$ lista de bloques
    - Directorio: nombre $\Rightarrow$ número de archivo

Discos en Unix

```
# fdisk -l

Disk /dev/sdb: 119,2 GiB, 128035676160 bytes, 250069680 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: dos
Disk identifier: 0xa5c0cf53

Device     Boot     Start       End   Sectors  Size Id Type
/dev/sdb1  *         2048 125034495 125032448 59,6G  7 HPFS/NTFS/exFAT
/dev/sdb2       125036542 250068991 125032450 59,6G  5 Extended
/dev/sdb5       233515008 250068991  16553984  7,9G 82 Linux swap / Solaris
/dev/sdb6  *    125036544 233515007 108478464 51,7G 83 Linux
```

Estructura de discos

- Área de arranque (primer bloque, pista, cilindro)
  - Interpretado por el arranque del hardware (BIOS)
  - Puede incluir una tabla de partición
- Bloque de control del sistema de archivos
  - Parámetros claves: número de bloque, capa de metadatos
  - Unix: “superbloques”
- “Bloque de control de archivos” (Unix: “inodos”)
  - Dueños, permisos
  - Localización de datos
- Posiblemente un mapa de espacio libre

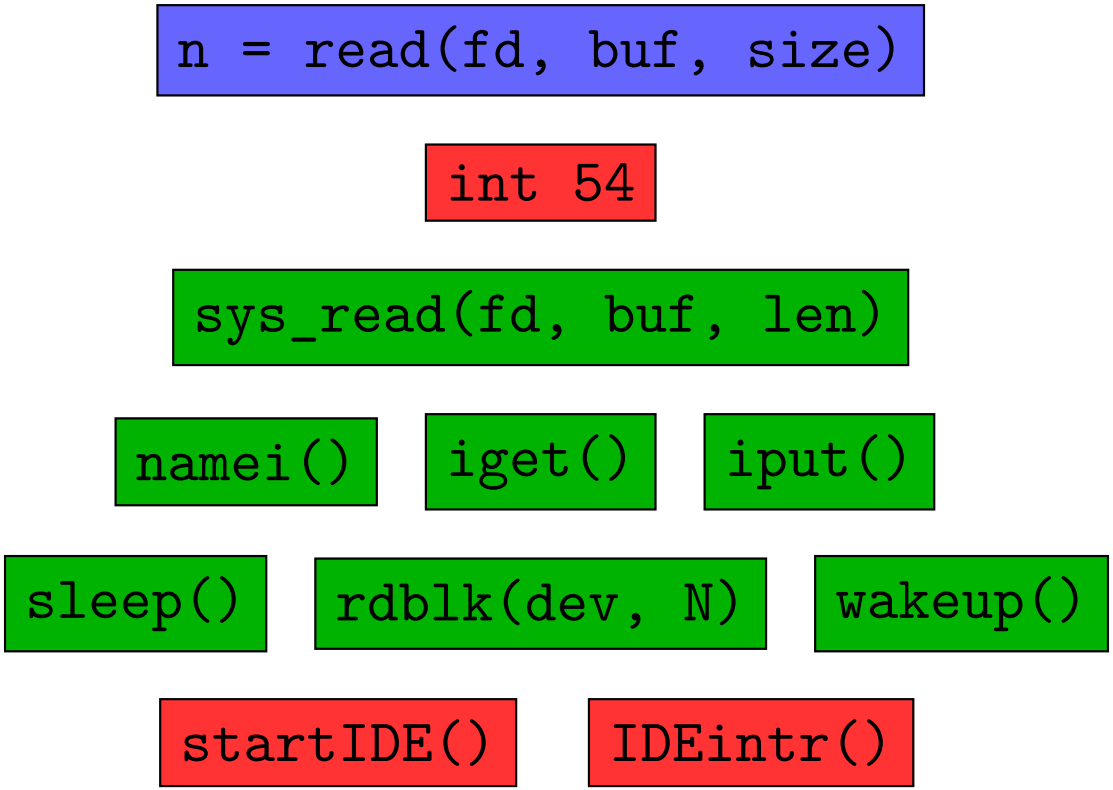
Estructuras en memoria

- Tablas de partición en memoria
  - Revisión de sanidad del sistema de archivos
  - I/O cabe en la partición correcta
- Información de directorios en cache
- Tabla de archivos abiertos del sistema
  - Bloques de control de archivos en memoria
- Tablas de archivos abiertos de procesos
  - Modo de apertura (escribir, leer, agregar, . . .)
  - Cursos (posición de leer y escribir)

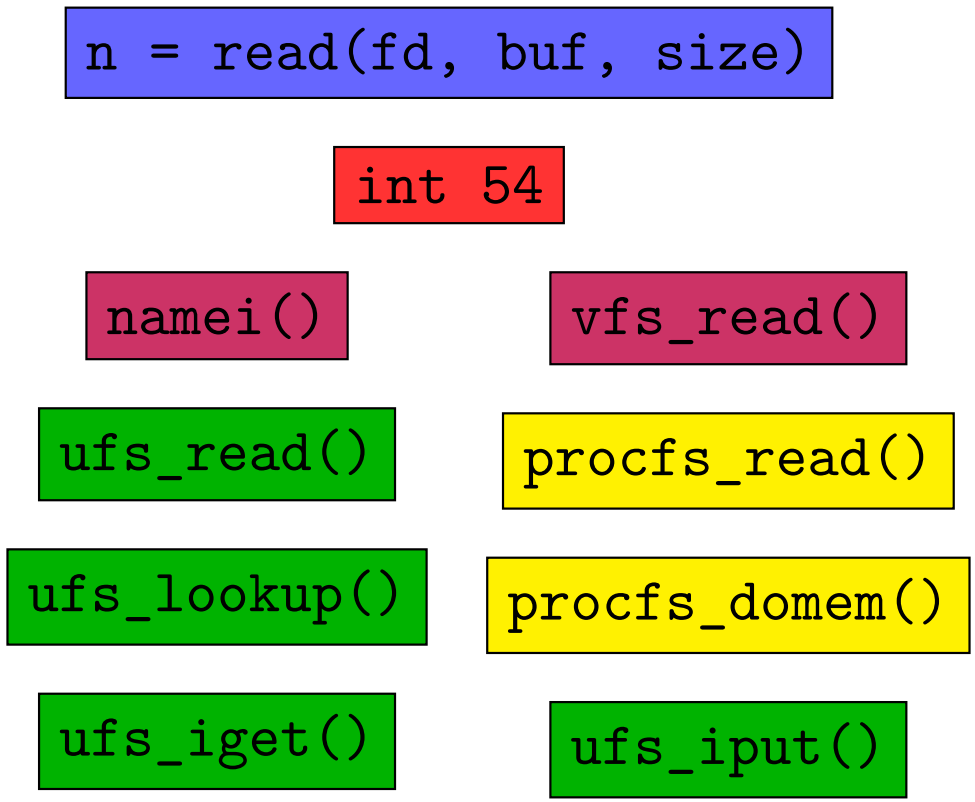
Capa VFS

- Objetivos
  - Permitir que una máquina pueda utilizar múltiples tipos de sistemas de archivos
    - Unix FFS
    - MS-DOS FAT
    - CD-ROM ISO 9660
    - Remotos o distribuidos: NFS/AFS
  - Llamadas estándar del sistema deben de funcionar de manera transparente
- Solución
  - Insertar un nivel de inderección!

Sistema de archivos simple



Virtualización VFS



Operaciones de archivos

- Son operaciones en los sistemas de archivos, no en los archivos individuales

```
struct vfsops{
  char *name;
  int (*vfs_mount)();
  int (*vfs_statfs)();
  int (*vfs_vget)();
  int (*vfs_unmount)();
  //...
}
```

Operaciones

- Cada VFS provee de un arreglo de métodos por archivo
  - `VOP_LOOKUP(vnode, new_vnode, name)`
  - `VOP_CREATE(vnode, new_vnode, name, attributes)`
  - `VOP_OPEN(vnode, mode, credentials, process)`
  - `VOP_READ(vnode, uio, readwrite, credentails)`
- Sistema operativo provee código independiente del sistema de archivos
  - Validar los parámetros de las llamadas de sistema
  - Mover datos de y hacia la memoria de usuario
  - Despertar y dormir hilos
  - Cache (bloques de datos, nombre $\Rightarrow$ mapeo a vnode)

Directorios

- Antes: un `namei()` $\Rightarrow$ VFS: el sistema de archivos provee un método de vnodes
  - `vnode2 = VOP_LOOKUP(vnode1, name)`
- Directorios FFS tradicionales de Unix
  - Lista de `(name, inode #)` —no ordenados
  - Nombres son variables en longitud
  - Búsqueda es lineal
    - ¿Cuánto tarda en borrar N archivos?
- Alternativa común: directorios de tablas de hash

Alojar y mapear

- Problema de alojamiento
  - ¿Donde colocamos el siguiente bloque de este archivo?
  - "Cerca del bloque anterior" no es una mala idea
  - Más allá de eso, se complica
- Problema de mapeo
  - ¿Donde está el bloque 42 de este archivo?
  - Similar a la memoria virtual
    - "Espacios de direcciones" grandes y múltiples específicos para cada archivo
    - Solo un "espacio de direcciones" de bloques
    - El espacio de direcciones fuente puede ser disperso (como la memoria virtual)

Tipos

- Contigua
- Enlazada
- FAT —File allocation table
- Indexado
  - Enlazado
  - Múlti nivel
  - Unix (árbol de índices)

Contiguo

- Enfoque
  - Posición de los archivos se define como `(inicio, longitud)`
- Motivación
  - Los accesos secuenciales al disco son baratos
  - Bookkeeping es sencillo
- Problemas
  - Asignación dinámica de almacenamiento (fragmentación, compactación)
  - Debe pre-declarar el tamaño del archivo al momento de crearlo
  - ¿Suena familiar? (revisar discusiones y notas de memoria virtual)

Enlazado

- Enfoque
  - Posición de los archivos se define como `(inicio)`
  - Cada bloque del disco contiene un puntero al siguiente bloque
- Motivación
  - Evita los problemas de fragmentación
  - Permite que los archivos crezcan
- ¿Problemas?

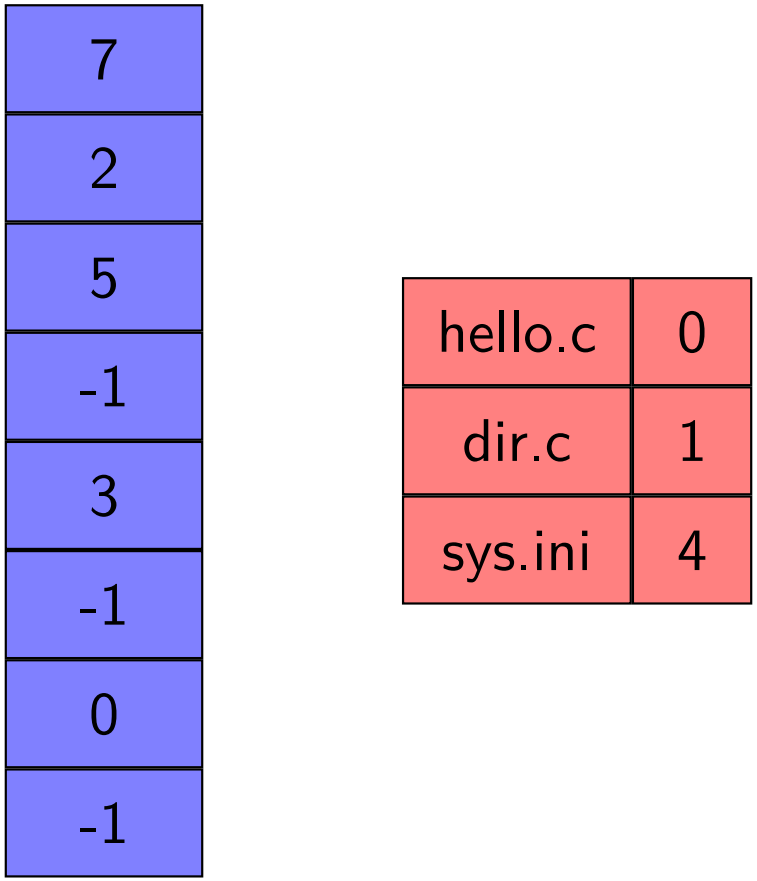
Problemas

- Problemas
  - Bloques de 508 bytes no se emparejan con las páginas de memoria
  - En general, uno busca bloques para leer y escribir —lento
  - Muy difícil de accesar bloques de archivos aleatoriamente `lseek(fd, 37*1024, SEEK_SET);`
- Beneficios
  - PUede recuperar archivos aunque los directorios han sido destruidos
- Modificación común
  - Enlazar clusters múlti-bloque, no bloques

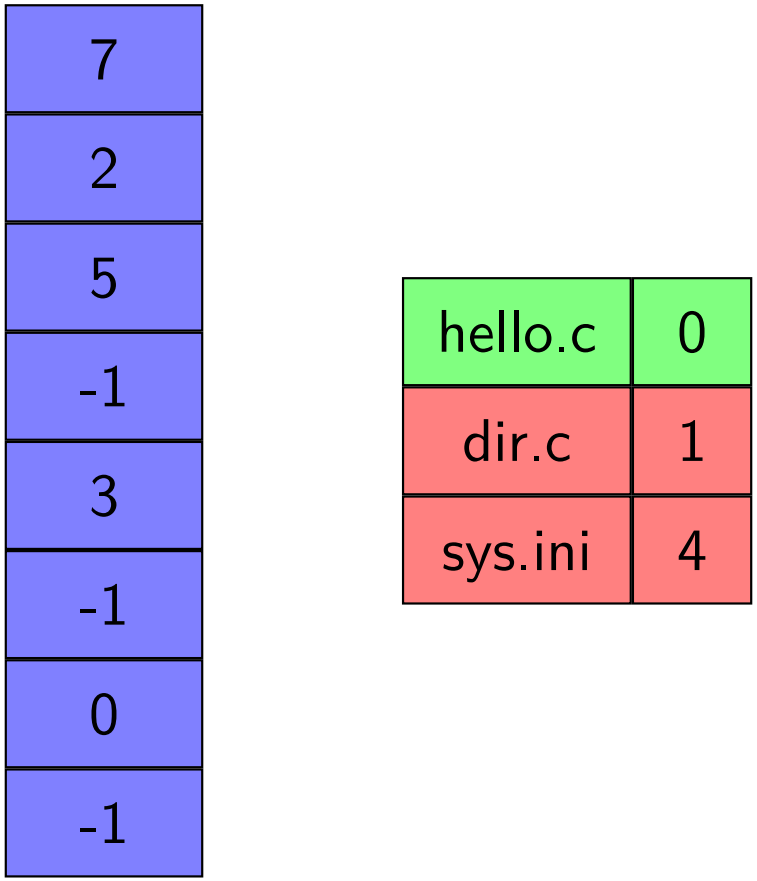
FAT

- Utilizada por MS-DOS, OS/2, Windows
  - Cámaras digitales, receptores de GPS, impresoras, PalmOS, etc.
- Semánticamente es lo mismo que la asignación enlazada
  - Pero, el enlace al "siguiente bloque" está almacenado en una tabla
  - Resultado: sectores de datos de 512 bytes
- Tabla al inicio del disco
  - Arreglo de punteros de siguiente-bloque
  - Indexado por número de bloque
  - `next = 0` es libre

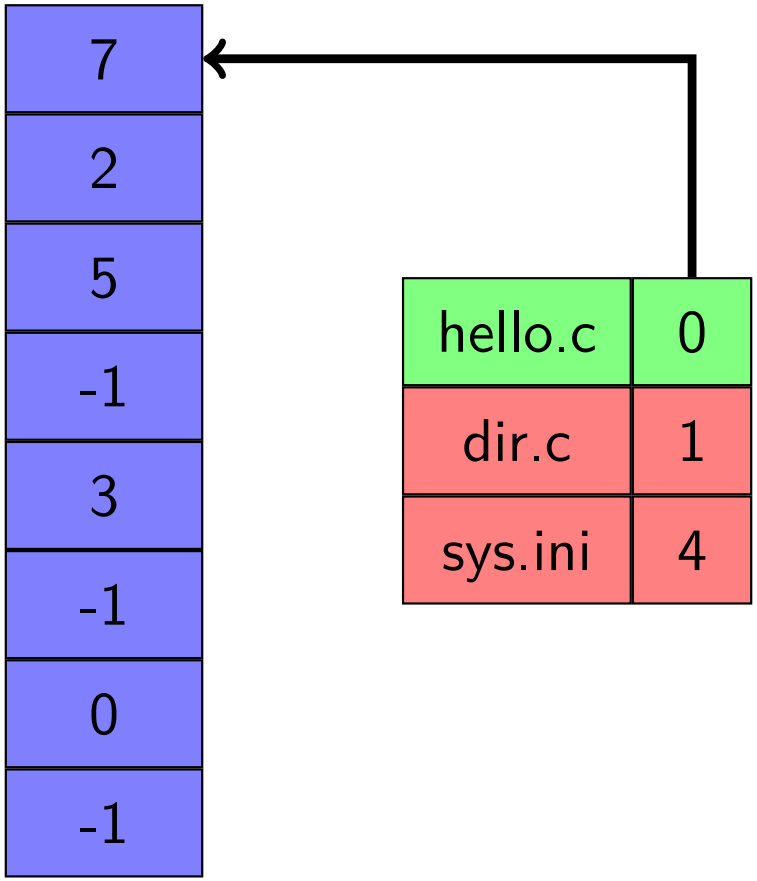
Ejemplo de FAT



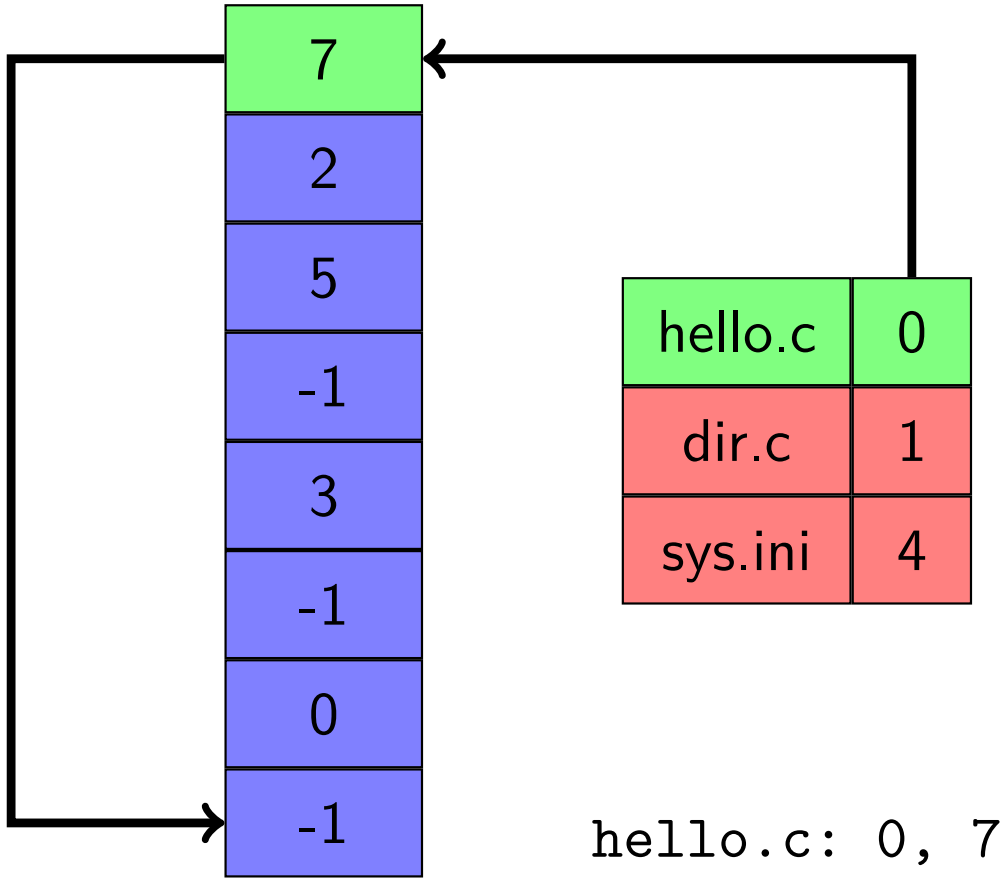
Ejemplo de FAT



Ejemplo de FAT



Ejemplo de FAT



Problemas

- Daños a la FAT revuelve todo el sistema de archivos
  - Solución: espejo de la FAT
- Generalmente dos búsquedas por lectura o escritura de bloques
  - Busca en la FAT, lee, busca el bloque actual (repetir)
  - A menos que se pueda mantener la FAT en RAM
- Aún difícil de acceder a los bloques de archivos de manera aleatoria
  - Tiempo lineal de recorrer la FAT
- FAT puede ser un "hot spot" (todos necesitan acceder a ella)
- Muchas actualizaciones a la FAT (cerca del inicio del disco)
  - A pesar de que los archivos modificados estén lejos

Indexado

- Motivación
  - Evitar problemas de fragmentación
  - Permitir crecimiento de archivos
  - Mejorar acceso aleatorio
- Enfoque
  - Arreglo de bloques por archivo
  - Número de bloque de archivo indexado en una tabla, entrega el número de bloque de disco

99	3004
100	-1
101	-1
3001	-1
3002	6002
-1	-1
-1	-1
-1	-1
-1	-1

Indexado

- Permite “hoyos”
  - `foo.c` es secuencial
  - `foo.db`, bloques 1 al 3 $\Rightarrow$ -1
  - Lógicamente blancos
- Asignación dispersa
  - a.k.a. “holes”
  - `read()` retorna nulo
  - `write()` requiere alojar
  - El tamaño del archivo no es el tamaño del archivo
    - `ls -l` índice del último byte
    - `ls -s` número de bloque

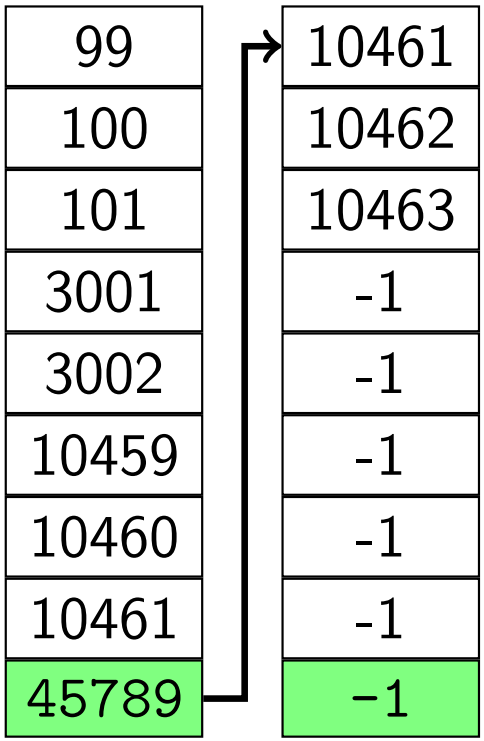
`foo.c`	`foo.db`
99	3004
100	-1
101	-1
3001	-1
3002	6002
-1	-1
-1	-1
-1	-1
-1	-1

Características

- ¿Qué tan grande debe de ser el bloque del índice?
  - Muy pequeño: limita el tamaño del archivo
  - Muy grande: desperdicia muchos punteros
- Combinando bloques de índices
  - Enlazados
  - Múlti-nivel
  - Lo que hace Unix

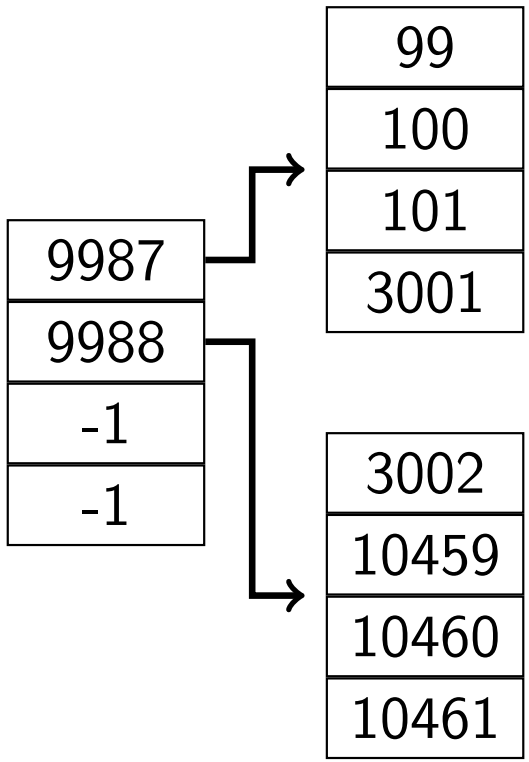
Bloques de índices enlazados

- El último puntero indica el siguiente índice de bloque
- Simple
- Acceso no es tan aleatorio
  - $O(n/c)$ es aún $O(n)$
  - $O(n)$ transferencias de disco



Bloques de índices múlti-nivel

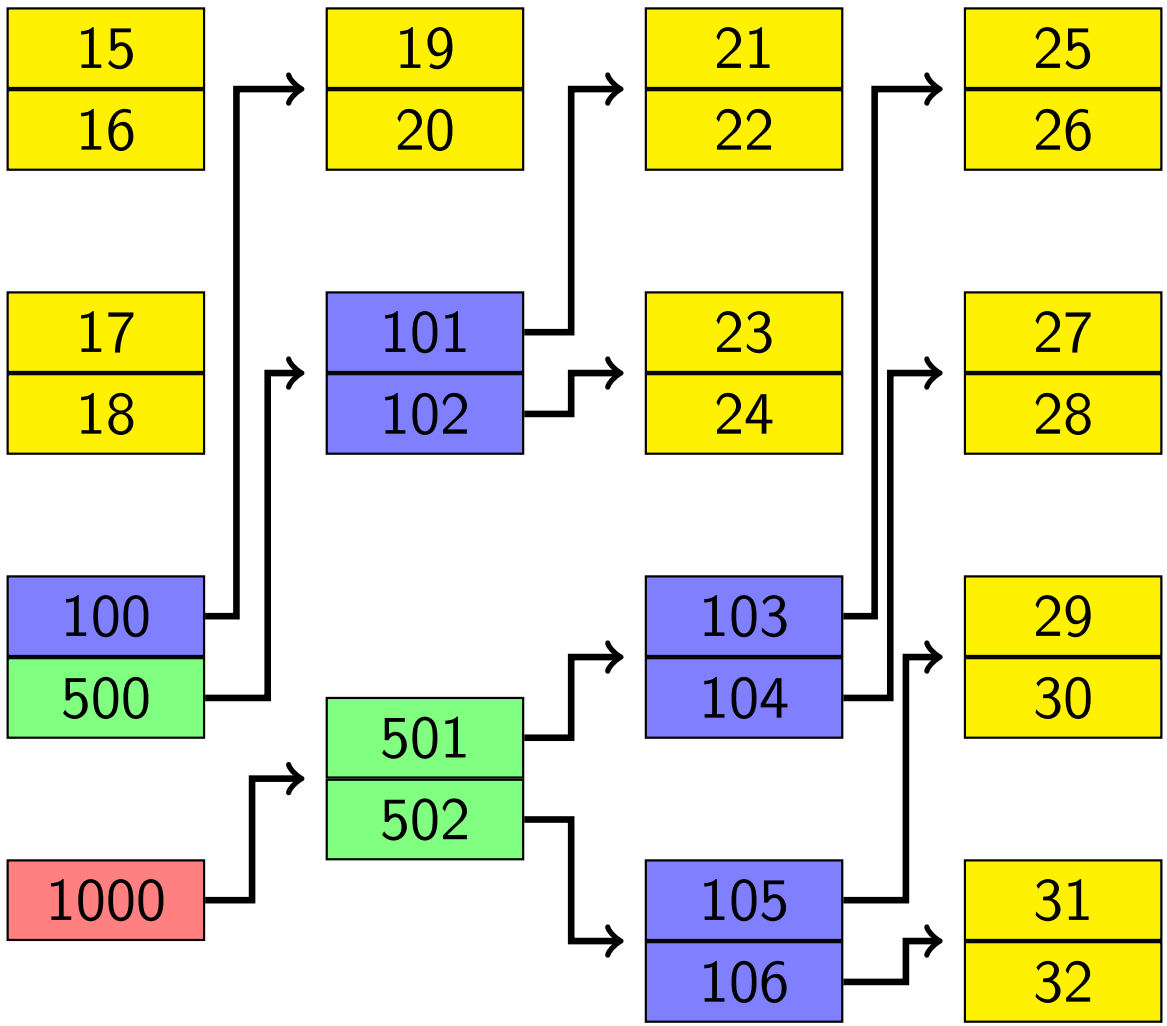
- Bloques de índice de bloques de índices
- ¿Suena familiar? (revisar notas y diapositivas de memoria virtual)
- Permite grandes agujeros



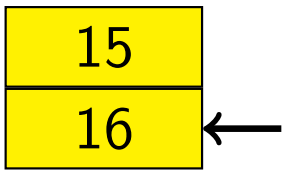
Bloques de índices de Unix

- Intuición
  - Muchos archivos son pequeños
  - Algunos archivos son muy grandes (gigabytes, incluse terabytes)
- ¿Cómo resolvemos este problema?
  - La magia no existe, y somos ingenieros
  - Así que sabemos que 57 niveles de indirección no serán rápidos
- "Heurísticas ingeniosas" de el Unix FFS inode
  - Estructura de inodos contiene 12 punteros "directos" a bloques
    - 12 números de bloques $\times$ 8 KB/bloque $=$ 96 KB
    - Disponibilidad es "gratis" —debe leer el inodo para abrir el archivo de cualquier manera
  - Estructura de inodos también mantiene 3 punteros indirectos a bloques
    - Indirecto, doble indirecto, y triple indirecto (puede acceder a $\gg 2^{32}$)

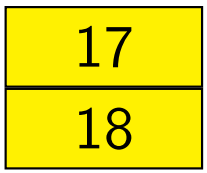
Bloques de índices de Unix

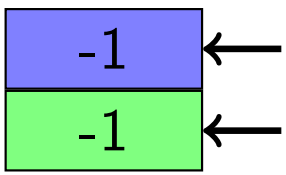


Bloques de índices de Unix



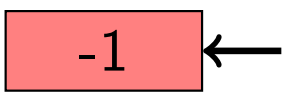
← Bloques “directos”, archivo alojado en 15 al 18





← Puntero indirecto

← Puntero indirecto doble



← Puntero indirecto triple

Rastreando espacio libre

- Bit-vector
  - 1 bit por bloque: boolean “libre”
  - Revisar cada palabra vs. 0
  - Utilizar instrucción de “establecer primer bit”
  - Ejemplo de texto
    - Disco 1.3 GB, sectores 512 B: 332 KB bit vector
- Necesita mantenerla en RAM (la mayor parte)

¿Cómo?

- Lista enlazada
  - El superbloque apunta al primer bloque libre
  - Cada bloque libre apunta al siguiente
- Costo de asignar N bloques es lineal
  - Bloque libre puede apuntar a múltiples bloques libres
    - 512 bytes $=$ 128 (4 byte) números de bloques
  - Enfoque de FAT provee de una lista de bloques libres de manera gratuita
- Mantener listas de espacios libres
  - `(block, sequential-block-count)`

Buffer de cache unificado

- Tradicionalmente hay dos enfoques
  - Página de cache, cache del sistema de archivos a menudo es totalmente independiente
    - Pedazos de las páginas de cache acorde al tamaño de página del hardware
    - Pedazos del cache de archivos acorde al tamaño del bloque del sistema de archivos
    - Diferente código, diferentes espacios de RAM
  - ¿Cuánta RAM podemos asignar a cada una?
- Observación
  - ¿Por qué no tener solo un cache?
  - Mezclar automáticamente varia acorde a la carga
    - cc quiere más cache de disco
    - Firefox quiere más VM cache

Trucos del cache

- Leer por adelantado

```
for (i = 0; i < filesize; i++)
  putc(getc(infile), outfile);
```

  - El sistema observa las lecturas secuenciales
  - Bloque de archivos 0, 1, 2, ...
  - Puede generar un conducto que superponga las lecturas y la computación (leer latencia)
    - Pedir dos bloques dispara una lectura de disco de 3 bloques
- Libre detrás, reemplazar por detrás
  - Descartar el buffer del cache cuando el siguiente sea solicitado
  - Bueno para archivos grandes
  - Anti-LRU (least recent used): elimina el MRU (most recent used) en lugar del (least recent used)

Recuperación

- Caída del sistema, ¿ahora qué?
  - Algún contenido de la RAM se pierde
  - Lista del espacio libre en el disco puede estar mal
  - Escanear el sistema de archivos
    - Revisar invariantes
    - Archivos no referencidos
    - Bloques doble alojados
    - Bloques sin asignar
  - Resolver problemas, ¿usuario experto?
- Enfoque moderno
  - Cambios en un “journal” (diario)

Backups

- Enfoque tradicional “Torres de Hanoi” incremental
  - Mensual: volcar el sistema de archivos entero
  - Semanal: volcar cambios desde el último mes
  - Diario: volcar los cambios desde el semanal anterior
  - Restaurar un archivo
    - El backup mensual tiene una copia (puede ser antigua pero...)
    - Alguna de las copias semanales puede tener una copia
    - Alguna de las copias diarias puede tener una copia

Enfoque de mezcla

- Las cintas corren más rápido cuando están en un flujo (velocidad máxima continua, no iniciar o detener)
- Recolectar cambios desde ayer
  - Escanear el sistema de archivos por modificaciones
- La cinta de salida está en blanco
- La cinta de entrada envía la información de ayer
  - Algunos archivos no han cambiado: enviar a la cinta de salida
  - Algunos archivo son viejos: cambiarlos en la cinta de salida
- Mantener cuantas cintas como sean necesarias, reciclar el resto
- Restaurar es rápido (enviar la cinta al disco)
- Archivos almacenados son muy redundantes —buena confiabilidad

Enfoque de instantáneas (snapshot)

- A la media noche, detener las escrituras al sistema de archivos
- Nuevas escrituras van al nuevo sistema de archivos
  - La mayoría punteros a la data de ayer
  - Cambios almacenados en el sistema de archivos en vivo
    - Tal vez archivos completos (copiar al escribir)
    - Tal vez solo bloques de datos nuevos
- Bueno para usuarios
  - Instantáneas antiguas pueden ser montadas (modo lectura)
  - Archivo borrado accidentalmente, podemos obtenerlo de la de ayer
  - AFS soporta versiones simples de archivos

Resumen

- Problema del mapeo de bloques
  - Similar al mapeo de memoria virtual a física
  - espacio de direcciones grande, a menudo disperso
    - Agujeros no son el caso común pero son posibles
  - Mapear cualquier dirección lógica a cualquier dirección física
  - Diferencia principal: el mapeo de archivos no cabe en memoria
- Nivel de indirección
  - Múltiples tipos de sistemas de archivos en una máquina
  - Crece el mapa de asignación de bloques
  - ...

Lecturas extra

- Journaling
  - Prabhakaran et al., Analysis and Evolution of Journaling File Systems (USENIX 2005)
- Algo interesante que no es journaling
  - McKusick y Ganger: “Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem” (USENIX 1999)