

CSE 120

Summer, 2005

Day 6
File Systems

Instructor: Neil Rhodes

Overview

File System Interface

- What the user/programmer sees

File System Implementation

- How it works

2

What is a File?

Named collection of related information stored in secondary storage

- Smallest allotment of logical secondary storage Long-term storage
- Must survive process termination (and system reboot!)

3

Typical Filesystem

Unix/Windows model:

- Hierarchical namespace
- create/open/close/read/write/seek
- File is a single collection of bytes

4

Other Possibilities

File as a database

- Records with named keys, types, and values
 - Example: Apple Newton
- Indexing on various keys provided by file system
 - For example, "find all records whose age > 19"

File as array of data chunks

- Palm OS, for example:
 - Records have attributes
 - Modified (Dirty)
 - Unique ID
 - Category
 - Deleted

Files with structure beyond sequence of bytes

- Vax VMS:
 - text files: sequence of lines of data
 - Binary files: sequence of bytes

Files with more than one stream (fork) of data

- Mac OS with resource fork and data fork
- NTFS/HFS+: multiple streams of data in a given file

5

File Metadata

Not the data in the file, but data about the file

- Owner
- Group
- Permissions
- Name
- Creation date
- Modification date
- Last access date
- Type
- Application Creator
- Icon
- Size
- Maximum size
- Locked
- Hidden
- etc.

6

File Types

File extension

- Examples: .c, .h, .doc, .bmp
- Enforced by OS (uses extension to determine what file to run)
- Or, used as convention (Unix)

Magic Number

- Various file have different magic numbers towards the beginning.
 - On Linux, see /usr/share/magic for long list of magic numbers for various files

Stored file type

- Mac OS, for example. File type *and* creator

7

Links

Two possibilities

- Symbolic link. A file "foo" has a reference to file "bar". If bar is deleted, using "foo" gives an error
- Hard link. "Foo" and "bar" both refer to the same file. If either is deleted, the other still refers to it

8

File Namespace

One-level

Two-levels

- Often, one per user

Hierarchical

- Tree

9

File Operations

Common

- Create
- Delete
- Open
- Close
- Read
- Write
- Seek
- Get attributes
- Set attributes

Less common

- Rename
- Append

10

Directory Operations

Common

- Create
- Delete
- OpenDir
- CloseDir
- ReadDir
- Rename

Less Common

- Link
- Unlink

11

Abstraction of the Disk

Sequence of equal-sized blocks: $0..n-1$

Operations

- Read block i
- Write block i

block 0
block 1
block 2
block 3
...
block n-3
block n-2
block n-1

12

File System Metadata

Everything except the contents of the files

- What blocks are free
- What blocks are used
- Which blocks are used for which files
- Directory structure
- Names, attributes, etc.

13

Information Kept About Open Files

System-wide open-file table

- Contains entry for each open file (attributes, disk block locations, etc.)

Per-process open-file table

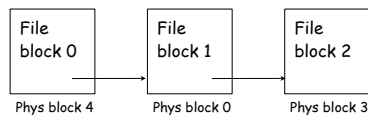
- Each entry contains:
 - Reference to system-wide open file table entry
 - Access mode
 - Current location in the file

14

Finding the Blocks of a File

Contiguous: all blocks are adjacent

- Pros: extremely fast to read
- Cons: must specify max size when creating the file. External fragmentation
- Example: CD-ROM



Linked List

- Pros: no external fragmentation
- Cons:
 - slow to get to block n
 - Uses data in block (no longer a power of two)

External Linked List

- Pros: All data in blocks available to user/program
- Cons: Linked list table must be in memory
 - 20GB disk 1KB block size -> 20,000,000 blocks -> table of size 60-80MB

Extents

- Allocate groups of contiguous blocks
 - For each one, keep start and number



15

Finding the Blocks of a File

index-node (*i-node*)

- Keep data structure for each file, stored in disk block(s).
 - Pointers to disk blocks. If too big, use 1 pointer as single-indirect, 1 as double, 1 as triple.
 - inode table contains location of each inode (stored on disk, but cached in mem).
- Pros: only in memory while the disk is open



i-node

block size: 1024 bytes. Max file size: $1024 \cdot (10 + 256 + 256^2 + 256^3) > 16\text{GB}$

16

Keeping Track of Free Space

Linked list of disk blocks

- Rather than storing one free block number per disk, store as many as will fit
 - Pros: little memory usage
 - Cons: disk access to allocate

Bitmap

- 1-bit per disk block.
- Pros:
 - Quick to access
 - Easy to allocate contiguous blocks
- Cons:
 - Fair amount of memory usage
 - 16GB disk, 1KB blocks -> 2^{24} bits -> 2^{21} bytes -> 2MB
 - Slow to find a free block if there aren't many free blocks

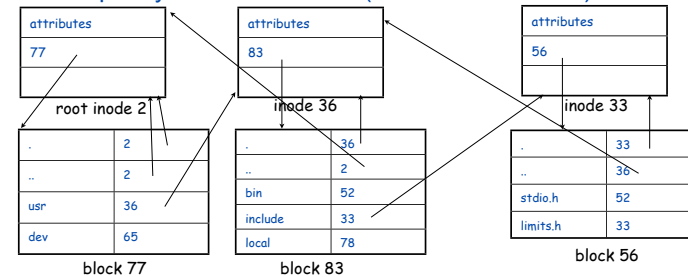
17

Implementing Directories

Keep name and attributes in fixed-size structure

name	attributes	contents
a.out	attributes	first block
main.c	attributes	first block
usr	attributes	first block

Keep only name and inode (/usr/include/stdio.h)



18

Implementing Links

Soft link (symbolic link)

- Contents of data block is name of file linked to

Hard link (multiple directory entries point at same inode)

- Count of links in inode
- When removing an entry from a directory, decrement the inode link count
 - If zero, free inode and blocks associated with inode

19

Caching

Disk access is slow

- Keep cache of recently used disk blocks

LRU is good choice, except:

- What about filesystem metadata? (Filesystem consistency)
 - If we corrupt a disk block used for a file, the file is corrupt
 - If we corrupt a disk block used for filesystem metadata, the filesystem is corrupt
- Write-through cache for filesystem metadata blocks
 - inodes
 - directory blocks
 - free block lists

What about cached written user data

- How long to keep cached?
- Unix approach
 - sync daemon calls sync (flushes cache) every 30 seconds
 - Process can call sync directly

20

Clustering

Group disk blocks that will be accessed close in time to be close in space

- Lower seek time because disk head doesn't need to move as much
- Example:
 - inodes at beginning of disk
 - inodes scattered in groups through the disk