CS 134
Operating Systems

March 4, 2019

File System

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!)

Typical filesystem

- Unix/Windows model:
  - Hierarchical namespace
  - `create/open/close/read/write/seek`
  - File is a single collection of bytes

Other possibilities

- File as a database
  - Records with named keys, types, and values
    - Example: Apple Newton, Be OS
  - Indexing provided by filesystem
    - For example, “Find all records where age > 19”
- File as array of data chunks
  - Palm OS, for example
    - Records have attributes:
      - Modified (Dirty)
      - Unique ID
      - Category
      - Deleted

Other possibilities (cont.)

- Files with structure beyond sequence of bytes
  - Vax VMS
    - text files: sequence of lines of text
    - 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

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.

File types

- File extension
  - Example: `.c`, `.h`, `.doc`, `.pdf`
  - Enforced by OS (uses extension to determine what program to execute for that file)
  - Or, used as convention (Unix)
- Magic number
  - Various files have different magic numbers toward the beginning
    - for example, `ELF_MAGIC` (`"\x7FELF"` at offset 0)
  - On Linux, see `man magic` for pointer to long list of magic numbers for various file types
- Stored file type
  - Classic Mac OS, for example. File type *and* creator

Links: two possibilities

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

File namespace

- One-level
- Two-levels
  - Often, one per user
- Hierarchical
  - Tree

File operations

- Common
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write
  - Seek
  - Get attributes
  - Set attributes
- Less common
  - Append
  - Rename

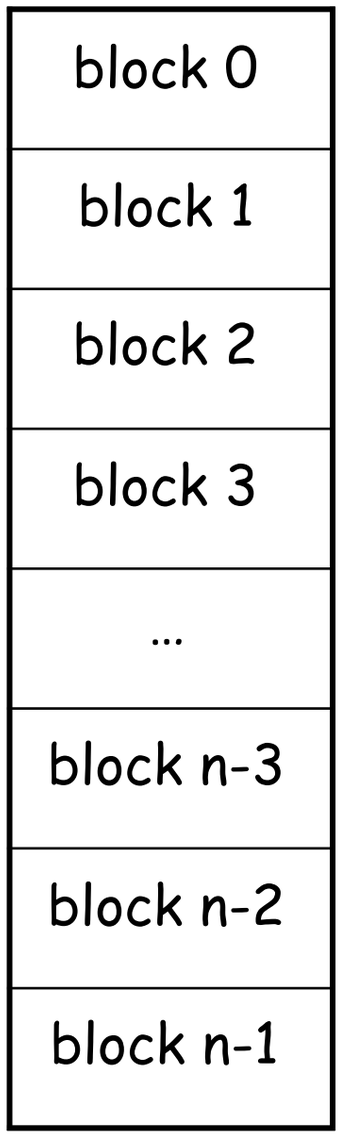
Directory operations

- Common
  - Create
  - Delete
  - OpenDir
  - CloseDir
  - ReadDir
- Less common
  - Rename
  - Link
  - Unlink

Abstraction of the disk

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

- Operations
  - Read block *i*
  - Write block *i*



Filesystem metadata

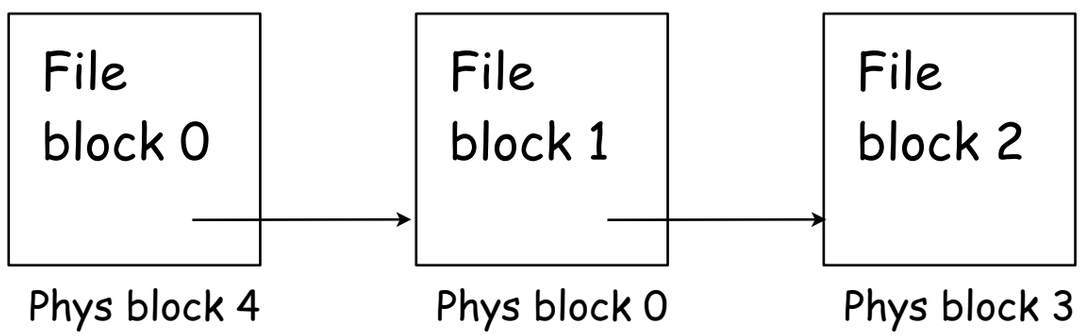
- Everything except the contents of the files themselves
  - What blocks are free
  - What blocks are in use
  - Which blocks are used (in what order!) for which files
  - Directory structure
  - Names, attributes, etc.

Information kept about open files

- System-wide open-file table
  - Contains entry for each opened file (attributes, disk block locations, current location within file, access mode, reference count)
    - Same file may be present more than once with different:
      - Current location within file
      - Access mode
- Per-process open-file table
  - Each entry contains:
    - Reference to system-wide open-file table

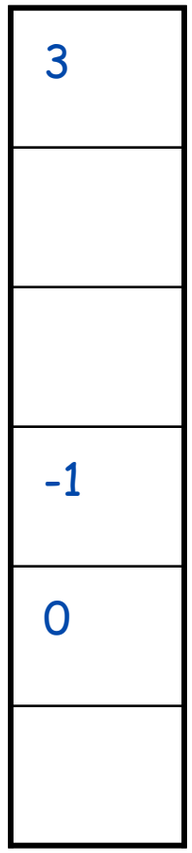
Finding the blocks of a file

- 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)



Finding the blocks of a file (cont.)

- 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



Finding the blocks of a file (cont.)

- Contiguous: all blocks are adjacent
  - Pros: extremely fast to read
  - Cons: must specify max size when creating a 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)

Finding the blocks of a file (cont.)

- 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 file is open

block size: 1024 bytes. Max file size: $1024*(10+256+256^2+256^3) > 16GB$

Keeping track of free space

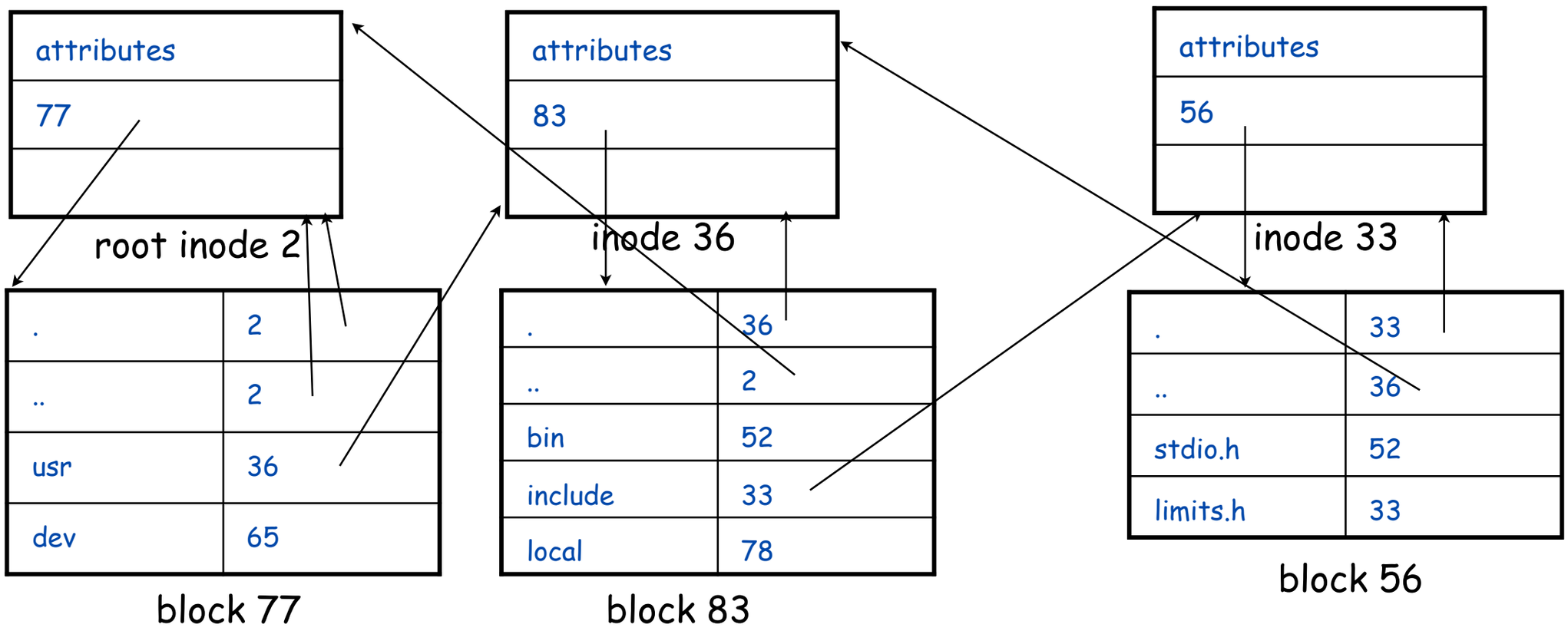
- Linked list of disk blocks
  - Rather than storing one free block number per disk, store as many as will fit
  - Pros: little memory usage
  - Cons: disk access to allocate
- Bitmap
  - 1-bit per disk block.
  - Pros:
    - Quick to access
    - Easy to allocate contiguous blocks
  - Cons:
    - Fair 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

Implementing directories

- Keep name, attributes and inode # in fixed-size structure

name	attributes	contents
a.out	attributes	inode #
main.c	attributes	inode #
usr	attributes	inode #

- `/usr/include/stdio.h`



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