File Systems

Mendel Rosenblum

File Systems

Optional readings: Operating Systems: Principles and Practice:

- Chapter 11
- Section 13.3 (up through page 567)

Key challenge for file systems: Disks

- Disk attributes: high latency, fast sequential access

Problems addressed by modern file systems

- Disk space management: How do we organize files on disk?
  - Sharing disk space between users
  - Fast access to files (minimize seeks)
  - Efficient use of disk space
- Naming: How do users select files?
  - Going from a file name to the location of its blocks on disk
- Reliability: Want to never lose data
  - Permanent storage, unlike most other memory OS manages
  - Information must survive OS crashes and hardware failures. Recovery important.
- Protection: isolation between users, controlled sharing

What is a file?

- User's view of file:
  - Named collection of bytes
  - Stored durably
- OS Kernel's view of file:
  - Collection of disk blocks
  - Some attribute storage (**metadata**)

File access patterns

- Sequential: information is processed in order, one byte after another
  - By far most common pattern (~90%) on today's machines
  - Examples: editor reads and writes, compiler reads and reads, etc.
- Random Access: can address any byte in the file by position
  - Data set for demand paging
  - Database systems
- Keyed (or indexed): search for blocks with particular contents
  - Examples: hash table, associative database, dictionary
  - Usually provided by databases, not operating system

Issues to consider

- Most files are small (a few kilobytes or less)
  - Per-file overheads must be low
- Most of the disk space is in large files
  - Most of the I/O operations are for large files
  - Performance must be good for large files
- Files may grow unpredictably over time
  - Don't know how big file will get when it is created

Inodes

- Index Node - **inode**
  - Operating system data structure, one per file
  - Kept in memory when file is open
  - Stored on disk along with file data
- Info in inode:
  - File size
  - Sectors occupied by file
  - Access times (last read, last write)
  - Protection information (owner id, group id, rwx)

File Structure

How should disk sectors be used to represent the bytes of a file?

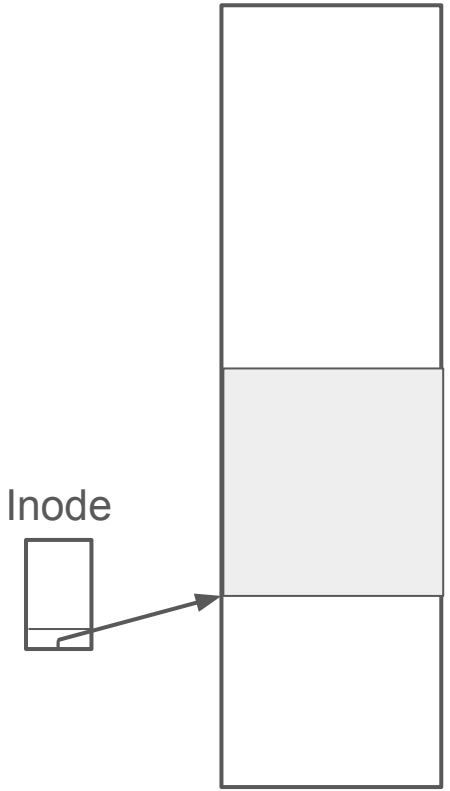
Desirable properties:

- Be able to find and access bytes of a file quickly
- Not take a lot of space (**metadata**)

Let's review the different approaches used historically

Contiguous allocation

- Allocate file as contiguous groups of sectors called extents
  - Also called extent-based allocation
  - Like base-and-bound and segmentation in virtual memory
- Inode contains number of first sector, file length in sectors
  - Compact space requirement
- User must specify length when creating a file.
- OS must keep a free list of unused areas of the disk.

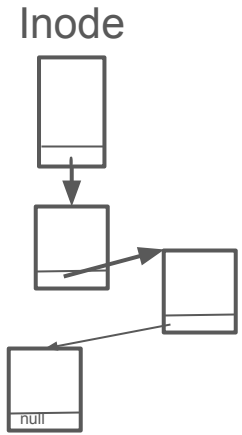


Contiguous allocation evaluation

- Advantages
  - Simple
  - Easy access, both sequential and random
  - Few seeks for I/O (optimal for sequential)
- Drawbacks
  - Fragmentation will make it hard to use disk space efficiently;
    - Large files may be impossible
  - Must predict size at file creation time
  - Can't extend files (leads to overallocation)
- Example: IBM OS/360

Linked files

- Divide disk into fixed-sized blocks (4096 bytes?)
  - Multiple of sector size
- Files contained a linked list of blocks
  - Inode: pointer to first block.
  - Each block of file contains pointer to next block
- Free list: keep a linked list of all free blocks



Linked files evaluation

- Advantages
  - Files can be extended, no fragmentation problems
  - Sequential access is easy: just chase links
  - Relatively small metadata
- Drawbacks
  - Random access too expensive to be practical
  - Lots of seeks, even in sequential access
- Examples (more or less): TOPS-10, Xerox Alto.

MS-DOS (Windows) FAT File System

- Used on the original IBM PC (1983)
- Keep the links for all files in a single table: **File Allocation Table (FAT)**
  - Stored on disk, kept in memory resident during normal operation
  - FAT entry per block on disk:
    - Disk sector number of next block in file
    - Special values for "last block in file"
    - Special values for "free block"
- Directory entry stores number of first block in file
- Find free space: search FAT for "free block"
  - Can start search from last block of the file

DOS FAT

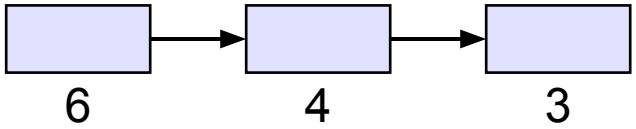
Directory (5)

A: 6
B: 1
...

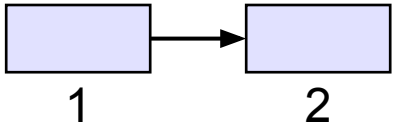
File Allocation Table

0	free
1	2
2	end
3	end
4	3
5	end
6	4
7	free
	…

File A:



File B:



FAT history

- Original FAT used 16 bit integers at 512 byte block
  - Biggest supported disk was 32 megabytes
- In 1996 Microsoft introduced FAT32
  - 32 bit integers support 28 bits of sector number
  - Add **clusters**: groups of adjacent sectors
    - Cluster sizes 2 - 32 kilobyte (fixed at file system creation)
  - 4 kilobyte clusters: Supported disks up to 1 TB
  - 32 kilobyte clusters: Support disks up to 8 TB

FAT evaluation

- Advantages
  - Sequential access is easy, and fast if disk isn't fragmented
    - Can achieve contiguous allocation
  - Random access fairly fast (chase links in memory)
  - FAT is also the free list
  - No pointers in blocks
- Disadvantages
  - Disk free space tends to become fragmented
  - FAT must be kept in memory
- Still used today for flash sticks, digital cameras, many embedded devices