1 File System (cont')

1.1 Disk Layout Strategies

- How do you find all of the blocks for a file

    1. Contiguous Allocation
        - Using starting block + length (extent)
            - All blocks of file are located together on disk
    2. Linked Allocation
        - Using Pointer
            - Each block points to the next
            - Directory entry points to the first
    3. Indexed-based allocation
        - Using pointer
            - An "index" contains pointers to many other blocks
            - May require multiple, linked index blocks

1.2 Data Block Allocation: Pros and cons

- Contiguous based allocation

    - Pros
        - Fast sequential access
        - Fast allocation
        - Fast deallocation
        - Small amount of metadata
    - Cons

      Question What does it mean when moving whole files around?
        - External fragmentation
        - Needs compaction
        - Inflexible
        - Need to move whole files around

- Linked-based allocation

    - Pros
        - Easy sequential access
        - Disk blocks can be anywhere
        - No external fragmentation
    - Cons

    - Direct access is expensive
    - If a data block is corrupted, could lose the rest of file

- Index-based allocation
  - Pros
    - Handles direct access well (say block # 15 on file ”I love corgi.txt”)
    - Disk blocks can be anywhere
    - No external fragmentation
      - Data is placed randomly
  - Cons
    - Limits file size (∼ 4TB)
    - Cost of access bytes near the end of large files grows (i.e Triple indirect pointers)
      - pointer (in inode) → data block of addresses (triple indirect pointers) → data block of addresses (double indirect pointers) → data block of addresses (single indirect pointers) → data block

2 Implementation of Very Simple File System

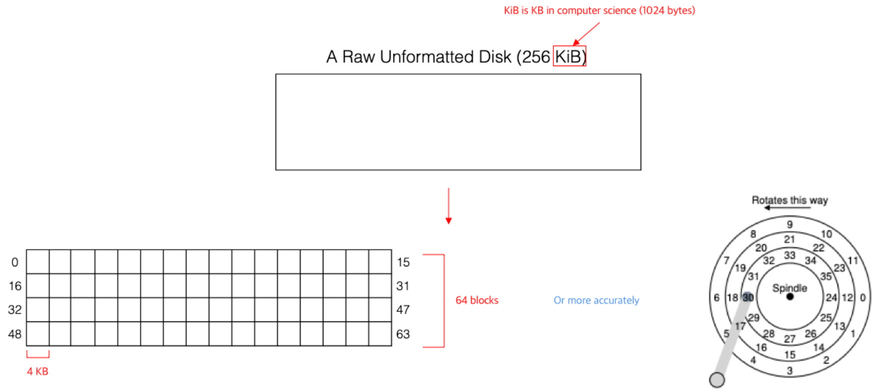
2.1 Existing File System

2.2 The main idea

- Goal: create a file system for an unformatted disk
- Key question: What does it mean to format a hard drive?
  - create some structure so that data will be easy to find and organize
- Other key questions:
  - Where do we store file data and metadta structures (answer: inode)
  - How do we keep track of data allocations
  - How do we locate file data and metaddata
  - What are the limitations (max file size, etc.)?

2.3 File System Implementation

- Key Question: How does file system use the disk to store files?
  - By defining a block size (4 KB)
  - By allocating disk space in granularity of blocks



2.4 Superblock

- Key question: What do we need to know to connect the disk to a computer?
  - Using superblock!!
- Determines the location of root directory
- Is duplicated across disk for reliability (in case of corruption)
- Is at well known block
  - Is always read first (on boot) before attaching to file system
    - Is to know where free map, and inodes are
    - Is to know what kind of file system is being stored
    - Is to know where the inode table begins
    - The number of inodes and data blocks in a particular file system
    - Other parameters

2.5 Freemap

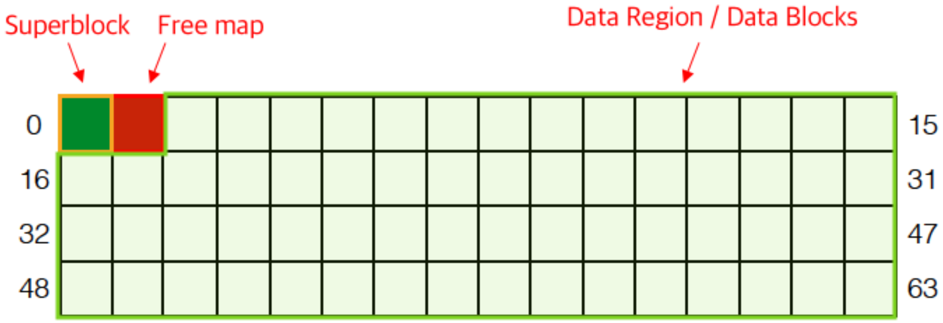
- Key question: How do we keep track of data allocations? That is, how do we know which blocks are free and which blocks are in use?
  - Free map!!
- Determines which blocks are free
  - Usually bitmap, 1 bit per block on disk
  - Stored on disk, cached in memory for performance
    - when read in memory, entries stored in L1 or L2 cache, so entries can be re-retrived quickly
- How many blocks can 4 Kib bitmap track of?
  - 4 KiB = 32 Ki bits ⇒ 32Ki blocks

2.6 Data blocks

- Is used to store files and directories
- Is region used to store user data

2.7 Data Region

- Is where data blocks are located



2.8 Metadata: inode table

- Lets FS to keep track of information about each file
- **inode** has size of 128 bytes in VFSF
- How many files can VSFS hold at most?
  - $(5 \cdot 4\text{KiB})/128\text{B} = 160$ files

2.9 Allocation Structure

- keeps track of which blocks are being used and which ones are free
- data structure
  - Is called **bitmap**
  - 0 - free
  - 1 - not free
- Two types
  1. Inode bitmap (for inode)
  2. Data bitmap (for data blocks)

2.10 Indirection

- block of addresses - 4 bytes per pointer
- single indirect block
  - can fit 1024 data blocks
- double indirect block
  - can fit 1024 * 1024 blocks
- triple indirect block
  - can fit 1024 * 1024 * 1024 blocks
- How big can a file be?
  - $4\text{KiB} \cdot (12 + 1024 + 1024^2 + 1024^3) \approx 4\text{TiB}$

2.11 Why an imbalanced tree?

- Designed based on evidence
  - Because most files are small (2KiB)
  - Files are usually accessed sequentially
  - Directories are typically small (20 or fewer entries)

2.12 Another Approach: extent-based

- An extent == a disk pointer plus a length (in # of blocks)
- Instead of a pointer to every block of file, a pointer to every few blocks needed

  Example

  ext4, HFS+, NTFS, XFS

2.13 Yet another approach: Link-Based

- Uses in memory File Allocation Table, indexed by address of data block
  - Faster in finding a block
- Works poorly if we want to access the last block of a big file

  Example

  Microsoft's FAT file system

2.14 Summary

- Inodes
  - Data structure rerpesenting a FS object (file, dir, etc)
  - Attributes + disk location
  - No file name, just metadata
- Directory
  - List of (name, inode) mappings
  - Each directory entry: a file, other directory, link, itself (.), parent(..), etc