Lecture Topics: 12/1

- File System Implementation
  - Space allocation
  - Free Space
  - Directory implementation
  - Caching
- Disk Scheduling
- File System/Disk Interaction
  - FFS
  - LFS

Disk Block Allocation

- The basic unit of storage on a disk is a *block*
  - Block sizes vary, but think 512 bytes
- Each file is stored in one or more blocks
- For simplicity, blocks are not split between files; leftover space at the end of a block is wasted
- When creating or enlarging a file, which disk block(s) should be allocated to the file?

Contiguous Allocation

- In contiguous allocation, a file gets blocks b, b+1, b+2, ...
- Direntry stores starting location, length
- Two blocks with sequential numbers are very likely to be in the same track, so no head movement is required
- We have the dynamic allocation problem again
- And this time it's even worse: harder to predict file size at creation time

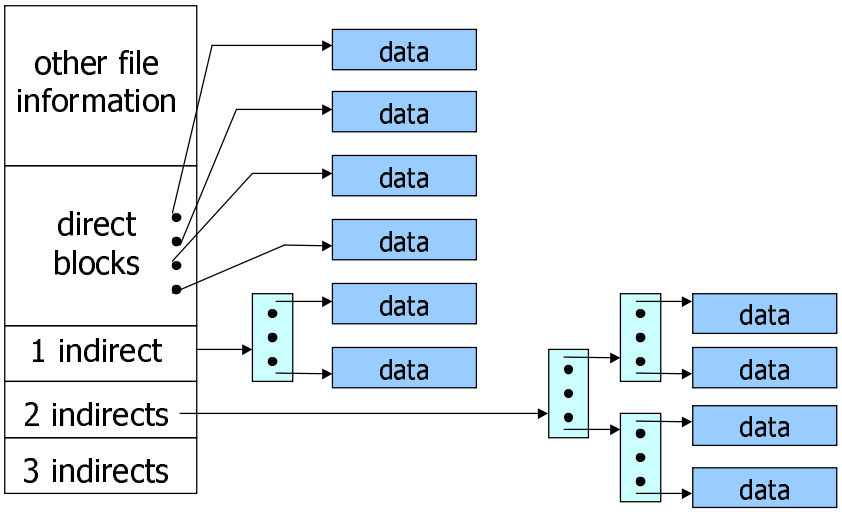
Linked allocation

- In linked allocation, a file gets a linked list of disk blocks
- Direntry stores starting location
- Each block contains data and a pointer to the next block
- Advantages: no dynamic allocation problems or external fragmentation
- Disadvantages: long access times, potential for lost data if pointers are damaged

Indexed allocation

- In indexed allocation, the file gets a list of disk blocks
- An index block contains the block list
- Advantages: same as linked allocation, plus better access times
- Disadvantages: for small files, most of the index block is wasted.
- How do we deal with very large files?

Unix Inodes



Free Space

- How do you find free disk blocks?
- Bitmap: One long string of bits represents the disk, one bit per block
- Linked list: each free block points to the next one (slow!)
- Grouping: list free blocks in the first free block
- Counting: keep a list of streaks of free blocks and their lengths

Directory Implementations

- Linear list
  - One entry after another, each containing the file name, first disk block, etc.
  - Linear search to find a file
- Hash table

File Caching

- Just like everything else in computer systems, we cache file blocks also
- When any part of a file block is read, the entire block is copied into memory
- Some systems have fixed memory partitions, one for file cache and one for the VM system
- Others allow the file cache and VM to dynamically trade off space

Accessing Disk Blocks

- The problem: given a set of disk blocks to access, the order of access really matters
- Example: we reference blocks on tracks 283, 17, 934, 887, 210, 513
- Time to move between tracks is roughly proportional to difference in track numbers

Disk Scheduling

- The solution: break up incoming disk requests into sequences of, e.g., 10
- Within each sequence of 10, mix up the order so that overall latency is improved
- Algorithms:
  - FCFS, Shortest Seek Time First (SSTF), elevator algorithm (SCAN), other variants

FS/Disk Interaction

- Knowing about disk access time issues, we can optimize file systems:
  - smart disk block allocation within a file
  - smart directory layout
- In our first picture of directory layout, the directories went at the top of the disk and all the files went after that
- FFS insight: put directories near the files they index; exploit locality

LFS

- Memory is cheap, and file caches are getting bigger
- If we have an infinitely large cache, and computers stay up infinitely long, eventually all files will be in the cache
- Reads are now really fast, but writes are still slow
- This insight changes how we use the disk

LFS

- Whenever you have to write, use the track the head is currently over
- When the track gets full, move to the next one
- Don't go back and overwrite the disk block that already contains this file block; just make it obsolete
- Makes reads slow and writes fast, but that's OK, because cache gets reads