Part1

Wednesday, May 22, 2019 12:05 PM

Operations on directories
- Create: make a new directory
- Delete: remove a directory, usually must be empty
- Opendir: open a directory to allow searching it
- Closedir: close a direstory, done searching
- Readdir: read a directory entry
- Rename: change the name of a directory
  - Similar to renaming a file

File system implementation issues

Contiguous allocation for file blocks
- Contiguous allocation requires all blocks of a file to be consecutive on disk
- Problem: deleting files leaves "holes"
  - Similar to memory allocation issues
  - Compacting the disk can be a very slow procedure…

Contiguous allocation
- Data in each file is stored in consecutive blocks on disk
- Simple and efficient indexing
  - Starting location, block #, on disk, start
  - Length of the file in blocks, length
- Random access well-supported
- Difficult to grow files

Linked allocation
- File is a linked list of disk blocks
  - Blocks may be scattered around the disk drive
  - Blok contains both pointer to next block and data
  - Files may be as long as needed
- New blocks are allocated as needed
  - Linked into list of blocks in file
  - Removed from list, bitmap, of free blocks

Finding blocks with linked allocation
- Directory structure is simple

Linked allocation using a table in RAM
- Links on disk are slow
- Keep linked list in memory
- Advantages
  - Faster
  - Disk blocks aren't an odd size
- Disadvantages
  - Have to read it from disk at some point, startup?
  - Have to keep in-memory and on-disk copy consistent

Finding blocks with indexed allocation

Larger files with indexed allocation
- How can indexed allocation allow files larger than a single index block?
- Linked index blocks: similar to linked file blocks, but using index blocks instead
- Logical to physical mapping is done by
- …
- File size is now unlimited
- Random access slow, but only for very large files

Block allocation with extents
- Reduce space consumed by index pointers
    - Often, consecutive blocks in file are sequential on disk
    - Store <block, count> instead of just <block> in index
    - At each level, keep total count for the index for efficiency
- Lookup procedure is:
    - Find correct <block, count> entry by running through index block, keeping track of how far into file the entry is
    - Find correct block in <block, count> pair
- More efficient if file blocks tend to be consecutive on disk
- Allocating blocks like this allows faster reads and writes
- Lookup is somewhat more complex

Managing free space: linked list
- Use a linked list to manage free blocks
    - Similar to linked list for file allocation
    - No wasted space for bitmap
    - No need for random access unless we want to find consecutive blocks for a single file
- Difficult to know how many blocks are free unless its tracked elsewhere in the file system
- Difficult to group nearby blocks together if they're freed at different times
    - Less efficient allocation of blocks to files
    - Files read an written more because consecutive blocks not nearby

Managing free space: bit vector
- Keep a bit vector, with one entry per file block
    - Number bits from 0 through n-1, where n is the number of blocks available for files on the disk
    - If bit[j] == 0, block j is free
    - If bit[j] == 1, block j is in use by a file (for data or index)
- If words are 32 bits long, calculate appropriate bit by:
    - wordnum = block / 32
    - bitnum = block % 32
- Search for tree blocks by looking for words with bits unset: words != 0xffffffff
- Easy to find consecutive blocks for a single file
- Bit map must be stored on disk, and consumes space
    - Assume 4KB blocks, 256 GB disk => 64M blocks
    - 64M bits = 2^26 bits = 2^23 bytes = 8MB overhead

Big or small file blocks?
- Larger blocks are
    - Faster: transfer more data per seek
    - Less efficientL waste space

Part2

Friday, May 24, 2019 12:09 PM

What's in a directory?
- Two types of information
  - File names
  - File metadata (size, timestamps, etc.)
- Basic choices for directory information
  - Store all information in directory
    - Fixed size entries
    - Disk addresses and attributes in directory entry
  - Store name and pointers to index nodes (i-nodes)

Directory structure
- Structure
  - Linear list of files (often itself stored in a file)
    - Simple to program
    - Slow to run
    - Increase speed by keeping it sorted (insertions are slower!)
  - Hash table: name hashed and looked up in file
    - Decreases search time: no linear searches!
    - May be difficult to expand
    - Can result in collisions (two files hash to same location)
  - Tree
    - Fast for searching
    - Easy to expand
    - Difficult to do in on-disk directory
- Name length
  - Fixed: easy to program
  - Variable: more flexible, better for users

Solution: use links
- A creates a file, and inserts into her directory
- B shares the file creating a link to it
- A unlinks the file
  - B still links to the file
  - Owner is still A (unless B explicitly changes it)

Log-structured file systems
- Trends in disk and memory
  - Faster CPUs
  - Larger memories
- Result
  - More memory -> disk caches can also be larger
  - Increasing number of read requests can come from cache
  - Thus, most disk accesses will be writes
- LFS structures entire disk as a log
  - Al writes initially buffered in memory
  - Periodically write these to the end of the disk log
  - When file opened, locate i-node, the find blocks

- Issue: what happens when blocks are deleted?
- Divide disk into segments
  - Write data sequentially to segments
  - Read data from wherever it's stored
- Periodically, "clean" segments
  - Go through a segment and copy live data to a new location
  - Mark the segment as free

Backing up a file system
- Goal: create an extra copy of each file in the file system
  - Protect against disk failure
  - Protect against human error (remove * .o)
  - Allow the system to track changes over time
- Two basic types of backups
  - Full backup: make a copy of every file in the system
  - Incremental backup: only make a copy of files that have changed since the last backup
    - Faster: fewer file to copy
    - Smaller

Backup mechanics
- Actually copy blocks from one file system to another
  - Safe if original FS fails
  - Safe if original FS is corrupted
  - May be difficult to find modified files
  - Somewhat slow
- Snapshot
  - New data doesn't overwrite old data
  - Easy to recover "deleted" files
  - Fast
  - Not as helpful for failed devices or corrupted FS
  - Snapshots can be done with hard links