

CS162 Operating Systems and Systems Programming Lecture 14

File Systems (Part 2)

March 18, 2013
Anthony D. Joseph
<http://inst.eecs.berkeley.edu/~cs162>

Review: Storage Performance

- Hard (Magnetic) Disk Performance:
 - Latency = Queuing time + Controller + Seek + Rotational + Transfer
 - Rotational latency: on average $\frac{1}{2}$ rotation
 - Transfer time: depends on rotation speed and bit density
- SSD Performance:
 - Read: Queuing time + Controller + Transfer
 - Write: Queuing time + Controller (Find Free Block) + Transfer
 - Find Free Block time: depends on how full SSD is (available empty pages), write burst duration, ...
 - Limited drive lifespan

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.2

Review: File System Goals

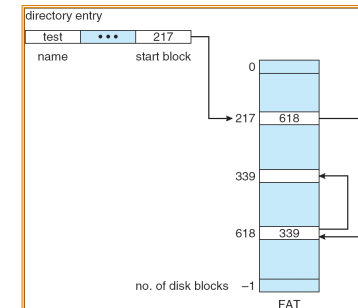
- Maximize sequential performance
- Efficient random access to file
- Easy management of files (growth, truncation, etc)

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.3

Review: Linked Allocation



- MSDOS links pages together to create a file
 - Links not in pages, but in the File Allocation Table (FAT)
 - » FAT contains an entry for each block on the disk
 - » FAT Entries corresponding to blocks of file linked together
 - Access properties:
 - » Sequential access expensive unless FAT cached in memory
 - » Random *really* expensive if FAT not cached

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.4

Goals for Today

- File Systems Structures (cont'd)
- Naming and Directories

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated from my lecture notes by Kubiawicz.

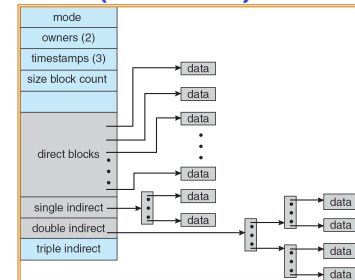
3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.5

Multilevel Indexed Files (UNIX 4.1)

- Multilevel Indexed Files: (from UNIX 4.1 BSD)
 - Key idea: efficient for small files, but still allow big files



- File hdr contains 13 pointers
 - Fixed size table, pointers not all equivalent
 - This header is called an “inode” in UNIX
- File Header format:
 - First 10 pointers are to data blocks
 - Ptr 11 points to “indirect block” containing 256 block ptrs
 - Pointer 12 points to “doubly indirect block” containing 256 indirect block ptrs for total of 64K blocks
 - Pointer 13 points to a triply indirect block (16M blocks)

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.6

Multilevel Indexed Files (UNIX 4.1): Discussion

- Basic technique places an upper limit on file size that is approximately 16Gbytes
 - Designers thought this was bigger than anything anyone would need. Much bigger than a disk at the time...
 - Fallacy: today, Facebook gets hundreds of TBs of logs every day!
- Pointers get filled in dynamically: need to allocate indirect block only when file grows > 10 blocks
 - On small files, no indirection needed

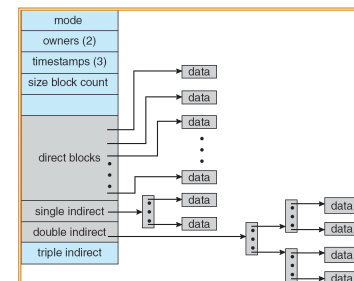
3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.7

Example of Multilevel Indexed Files

- Sample file in multilevel indexed format:
 - How many accesses for block #23? (assume file header accessed on open)?
 - » Two: One for indirect block, one for data
 - How about block #5?
 - » One: One for data
 - Block #340?
 - » Three: double indirect block, indirect block, and data



- UNIX 4.1 Pros and cons
 - Pros: Simple (more or less)
Files can easily expand (up to a point)
Small files particularly cheap and easy
 - Cons: Lots of seeks
Very large files must read many indirect blocks (four I/O's per block!)

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.8

UNIX BSD 4.2

- Same as BSD 4.1 (same file header and triply indirect blocks), except incorporated ideas from Cray-1 DEMOS:
 - Uses bitmap allocation in place of freelist
 - Attempt to allocate files contiguously
 - 10% reserved disk space (mentioned next slide)
 - Skip-sector positioning (mentioned in two slides)
- Problem: When create a file, don't know how big it will become (in UNIX, most writes are by appending)
 - How much contiguous space do you allocate for a file?
 - In BSD 4.2, just find some range of free blocks
 - » Put each new file at the front of different range
 - » To expand a file, you first try successive blocks in bitmap, then choose new range of blocks
 - Also in BSD 4.2: store files from same directory near each other

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.9

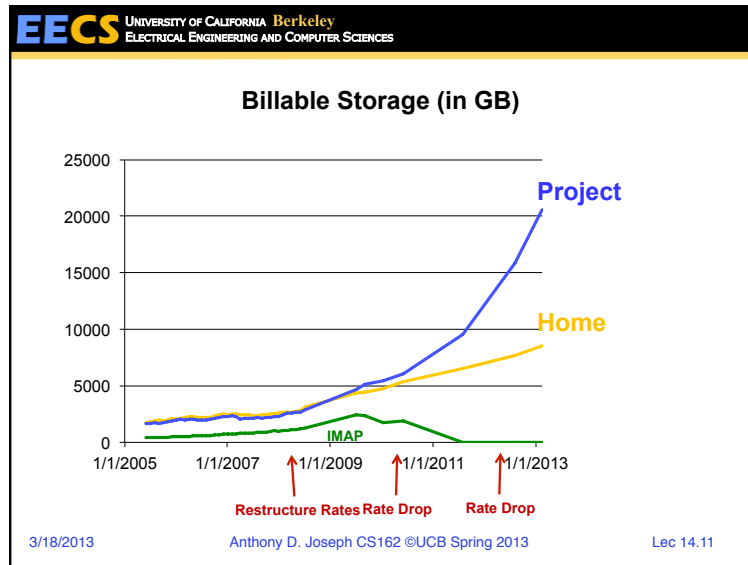
How to Deal with Full Disks?

- In many systems, disks are always full
 - EECS department growth: 300 GB to 1TB in a year (now 10s TB)

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.10



How to Deal with Full Disks?

- In many systems, disks are always full
 - EECS department growth: 300 GB to 1TB in a year (now 10s TB)
 - How to fix? Announce disk space is low, so please delete files?
 - » Don't really work: people try to store their data faster
 - Sidebar: Perhaps we are getting out of this mode with new disks... However, let's assume disks are full for now
- Solution:
 - Don't let disks get completely full: reserve portion
 - » Free count = # blocks free in bitmap
 - » Scheme: Don't allocate data if count < reserve
 - How much reserve do you need?
 - » In practice, 10% seems like enough
 - Tradeoff: pay for more disk, get contiguous allocation
 - » Since seeks so expensive for performance, this is a very good tradeoff

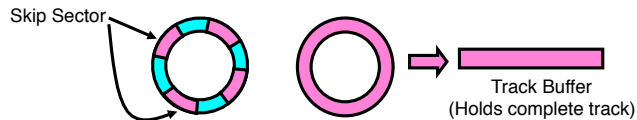
3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.12

Attack of the Rotational Delay

- Problem: Missing blocks due to rotational delay
 - Issue: Read one block, do processing, and read next block. In meantime, disk has continued turning: missed next block!



- Solution 1: Skip sector positioning (“interleaving”)
 - » Place the blocks from one file on every other block of a track: give time for processing to overlap rotation
- Solution 2: Read ahead: read next block right after first, even if application hasn’t asked for it yet
 - » This can be done either by OS (read ahead)
 - » By disk itself (track buffers). Many disk controllers have internal RAM that allows them to read a complete track
- Important Aside: Modern disks+controllers do many complex things “under the covers”
 - Track buffers, elevator algorithms, bad block filtering

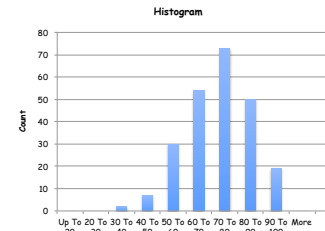
3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.13

Administrivia

- Midterm results: Mean 73.1, Std dev 13



- Regrade request deadline: April 2, 2013
 - We will regrade the entire exam

- Please fill the anonymous course survey at <https://www.surveymonkey.com/s/9DK2VVJ>
 - We'll try to make changes *this* semester based on your feedback

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.14

Quiz 14.1: File Systems

- Q1: True _ False _ With FAT, pointers are maintained in the data blocks
- Q2: True _ False _ Unix file system is more efficient than FAT for random access
- Q3: True _ False _ The “Skip Sector Positioning” technique allows reading consecutive blocks on a track
- Q4: True _ False _ Maintaining the free blocks in a list is more efficient than using a bitmap
- Q5: True _ False _ In Unix, accessing random data in a large file is on average slower than in a small file

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.15

5min Break

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.16

Quiz 14.1: File Systems

- Q1: True _ False **X** With FAT, pointers are maintained in the data blocks
- Q2: True **X** False _ Unix file system is more efficient than FAT for random access
- Q3: True _ False **X** The “Skip Sector Positioning” technique allows reading consecutive blocks on a track
- Q4: True _ False **X** Maintaining the free blocks in a list is more efficient than using a bitmap
- Q5: True **X** False _ In Unix, accessing random data in a large file is on average slower than in a small file

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.17

How do we actually access files?

- All information about a file contained in its file header
 - UNIX calls this an “inode”
 - » Inodes are global resources identified by index (“inumber”)
 - Once you load the header structure, all blocks of file are locatable
- Question: how does the user ask for a particular file?
 - One option: user specifies an inode by a number (index).
 - » Imagine: `open("14553344")`
 - Better option: specify by textual name
 - » Have to map name→inumber
 - Another option: Icon
 - » This is how Apple made its money. Graphical user interfaces. Point to a file and click

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.18

Naming

- **Naming (name resolution)**: process by which a system translates from user-visible names to system resources
- In the case of files, need to translate from strings (textual names) or icons to innumbers/inodes
- For global file systems, data may be spread over globe⇒need to translate from strings or icons to some combination of physical server location and inumber

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.19

Directories

- **Directory**: a relation used for naming
 - Just a table of (file name, inumber) pairs
- How are directories constructed?
 - Directories often stored in files
 - » Reuse of existing mechanism
 - » Directory named by inode/number like other files
 - Needs to be quickly searchable
 - » Options: Simple list or Hashtable
 - » Can be cached into memory in easier form to search
- How are directories modified?
 - Originally, direct read/write of special file
 - System calls for manipulation: `mkdir`, `rmdir`
 - Ties to file creation/destruction
 - » On creating a file by name, new inode grabbed and associated with new file in particular directory

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.20

Directory Organization

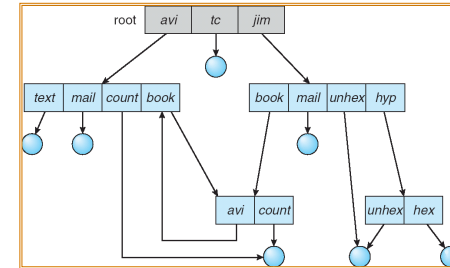
- Directories organized into a hierarchical structure
 - Seems standard, but in early 70's it wasn't
 - Permits much easier organization of data structures
- Entries in directory can be either files or directories
- Files named by ordered set (e.g., /programs/p/list)

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.21

Directory Structure



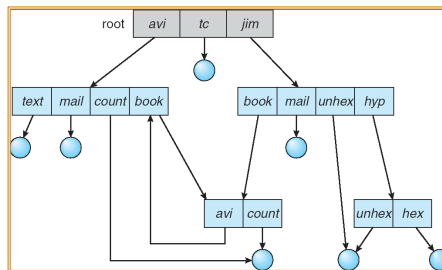
- Not really a hierarchy!
 - Many systems allow directory structure to be organized as an acyclic graph or even a (potentially) cyclic graph
 - Hard Links: different names for the same file
 - » Multiple directory entries point at the same file
 - Soft Links: “shortcut” pointers to other files
 - » Implemented by storing the logical name of actual file

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.22

Directory Structure



- **Name Resolution:** The process of converting a logical name into a physical resource (like a file)
 - Traverse succession of directories until reach target file
 - Global file system: May be spread across the network

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.23

Directory Structure (Con't)

- How many disk accesses to resolve “/my/book/count”?
 - Read in file header for root (fixed spot on disk)
 - Read in first data block for root
 - » Table of file name/index pairs. Search linearly – ok since directories typically very small
 - Read in file header for “my”
 - Read in first data block for “my”; search for “book”
 - Read in file header for “book”
 - Read in first data block for “book”; search for “count”
 - Read in file header for “count”
- **Current working directory:** Per-address-space pointer to a directory (inode) used for resolving file names
 - Allows user to specify relative filename instead of absolute path (say CWD=“/my/book” can resolve “count”)

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.24

Where are inodes stored?

- In early UNIX and DOS/Windows' FAT file system, headers stored in special array in outermost cylinders
 - Header not stored anywhere near the data blocks. To read a small file, seek to get header, seek back to data.
 - Fixed size, set when disk is formatted. At formatting time, a fixed number of inodes were created (They were each given a unique number, called an "inumber")

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.25

Where are inodes stored?

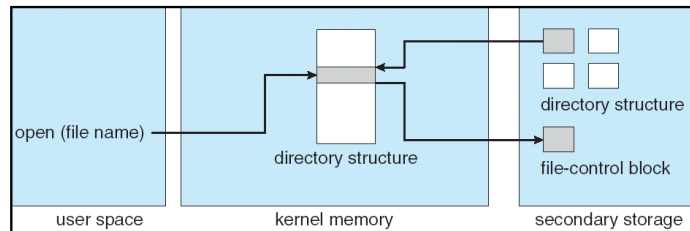
- Later versions of UNIX moved the header information to be closer to the data blocks
 - Often, inode for file stored in same "cylinder group" as parent directory of the file (makes an `ls` of that directory run fast).
 - Pros:
 - » UNIX BSD 4.2 puts a portion of the file header array on each cylinder. For small directories, can fit all data, file headers, etc. in same cylinder \Rightarrow no seeks!
 - » File headers much smaller than whole block (a few hundred bytes), so multiple headers fetched from disk at same time
 - » Reliability: whatever happens to the disk, you can find many of the files (even if directories disconnected)
 - Part of the Fast File System (FFS)
 - » General optimization to avoid seeks

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.26

In-Memory File System Structures



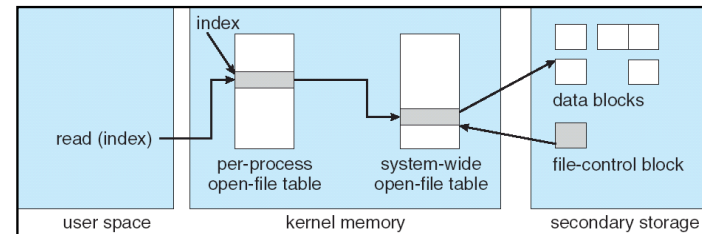
- Open system call:
 - Resolves file name, finds file control block (inode)
 - Makes entries in per-process and system-wide tables
 - Returns index (called "file handle") in open-file table

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.27

In-Memory File System Structures



- Read/write system calls:
 - Use file handle to locate inode
 - Perform appropriate reads or writes

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.28

Quiz 14.2: File Systems

- Q1: True _ False _ A hard-link is a pointer to other file
- Q2: True _ False _ inumber is the id of a block
- Q3: True _ False _ Typically, directories are stored as files
- Q4: True _ False _ Storing file headers on the outermost cylinders minimizes the seek time

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.29

Quiz 14.2: File Systems

- Q1: True _ False **X** A hard-link is a pointer to other file
- Q2: True _ False **X** inumber is the id of a block
- Q3: True **X** False _ Typically, directories are stored as files
- Q4: True _ False **X** Storing file headers on the outermost cylinders minimizes the seek time

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.30

File System Summary (1/2)

- File System:
 - Transforms blocks into Files and Directories
 - Optimize for access and usage patterns
 - Maximize sequential access, allow efficient random access
- File (and directory) defined by header, called “inode”
- Multilevel Indexed Scheme
 - Inode contains file info, direct pointers to blocks,
 - indirect blocks, doubly indirect, etc..

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.31

File System Summary (2/2)

- 4.2 BSD Multilevel index files
 - Inode contains pointers to actual blocks, indirect blocks, double indirect blocks, etc.
 - Optimizations for sequential access: start new files in open ranges of free blocks, rotational Optimization
- Naming: act of translating from user-visible names to actual system resources
 - Directories used for naming for local file systems

3/18/2013

Anthony D. Joseph CS162 ©UCB Spring 2013

Lec 14.32