File System Hierarchy, Unix File System

ECE595
Nov 8

Y. Charlie Hu

Roadmap

- Functionality (API)
  - Basic functionality
    - Disk layout
    - File operations (open, read, write, close)
  - Directories
- Performance
  - Disk allocation (file header design)
  - Disk scheduling
  - Buffer cache
    - interactions with VM
    - File system interface

Review: Disk Allocation

- Context:
  - A file has logical bytes/lblocks
  - Different files share physical blocks on disk
- Original goals are
  - How to allocate blocks for a file – for access performance
    - Random vs. sequential accesses
  - How to index the blocks for a file – for finding the blocks on disk
- Disk allocation methods:
  - Contiguous
  - Extent-based
  - Linked files / FAT
  - Single-level indexing
  - Multi-level indexing

Analogy

Memory Management	Disk Allocation
Virtual address per process	1-D logical bytes per file
Page/Seg table maps virtual addr to physical addr	File header maps logical bytes to disk blocks
Different schemes: Base& bound; Segmentation; 1-level paging; 2-level paging; Segment+paging; Inverse paging	Different schemes: Contiguous; Extent-based; Linked files / FAT; Single-level indexing; Multi-level indexing

Why File Hierarchy?

- Very flexible
- Lets people group related items
- Expands to hold large # of files easily
- Some notion of where you are "working"
- Drawback?
- Alternatives
  - everyone in one directory
  - single level of "partitioned" files (by user, usually)

So What's a Directory? (visualize this!)

- In Unix, just a specially-formatted file!
  - File metadata for a dir kept in inode
  - Yes, a kernel process can read/write a dir just like a file
  - It's a directory because "we treat it that way"
- Directory inode: set flag indicating it's a directory
- Directory data block contains names of files
  - <name, inode #> pairs in no particular order
  - The file pointed to by inode# may be another dir
  - A special dir, called root, has no name

Implications of Directory Format

Once having the data block of a dir

- Finding names is easy
  - Can get a list of files/subdirectories
  - Can tell if file/subdirectory does/doesn't exist
- What about these operations?
  - Determining if name in dir is a file or directory
  - File size info, ownership, create time, etc.
  - Why is doing 'ls –l' on big directory slow?
- How about "." and ".."?

So How Do You Find Files?

- Directory contains inode # of file
- Inodes seen as a logically contiguous array
- What happens when two dir entries with different names share the same inode #?
  - Allows same file to exist in two directories
  - Called a "hard link"

How Do You Really Find a File?
“/dir1/dir2/file”

- Want: inode number for “file”
  - Inode # provides inode, then data block #’s
- Have: absolute pathname
- Question: what contains entry for ‘dir1’?
- Answer: the ‘/’ directory, called ‘root’
- How do you find the inode for ‘/’?
  - Bootstrapping – this is your well-known # (inode 2 on UNIX)
  - “You'll have to ask the engineers at AT&T OR Berkeley I suppose. It's just convention as to why they chose the number 2.”

[lec20]
A Disk Layout for A File System

Boot block	Super block	File metadata (i-node in Unix)	File data blocks

- Boot block: contains info to boot OS
- Superblock defines a file system
  - Size of the file system
  - Free metadata (inode) count and pointers
  - Free block count and pointers (or pointer to bitmap)
  - Location of the metadata of the root directory
- What if the superblock is corrupted?
  - What can we do?

How Does This Work in Reality?

- Fetch root inode
  - Start loading root directory data blocks
  - Walk directory data until you find dir1
  - Get inode # for dir1
- Fetch dir1’s inode
  - Start loading dir1’s directory data blocks
  - Walk directory data until you find dir2
  - Get inode # for dir2
- …
- Repeat process until you have inode # for file

What People Noticed

- Name lookup is a slow process
  - Required CPU time
  - Required lots of disk block fetches (inodes, data blocks)
- Lots of common lookups
- Optimization? Caching!
  - Keep cache of name lookups
  - Keep cache of recently-used inodes, dir blocks
  - Reserve some kernel mem for this

File & Directory Creation

- File Creation
  - Get an inode
  - In directory data block, add entry with new name and inode #
- Directory Creation
  - Same as file, but in inode, set flag indicating it's a directory
- How do you rename a file or dir?
- How do you delete a file?

Kinds of Links

- Why links?
  - Sharing
- Hard link
  - Two name entries with same inode #
  - Always tied to that same inode #
- Implementation
  - How to delete a file?
  - Needs reference counting

Why can go wrong with hard links?

- Consider hard links that may point to dirs: may get cyclic dir graphs
  - Search/dir traversal: need to avoid infinite loops
- Unix solution
  - part 1: users can create hard links to plain files only
  - Part 2: provide symbolic links to dirs

Soft Links

- Soft link (symbolic link)
  - Specify name of file to use
    - Absolute or relative names work
  - Access requires evaluating link-specified name

Why support both types of Links? (cont)

- Symbolic links may refer to dirs and thus form cycles as well
  - Since they are special, can avoid infinite loop in dir searches by ignoring or limiting the # of symbolic links traversed
- Symbolic links can refer to dirs in other file systems, or residing on different disks
- Implementation
  - Not reference-counted
  - How can we make it more efficient?

Softlink implementation

- Once upon a time, it is a new file, data block contains the path of the file pointed
- To optimize, the path is stored in inode!
- inode also gives other info, such as
  - owner,
  - last modified time

UFS concurrency semantics

- What happens when two processes try to write to the same file?
  - What is the programmer's intent?
- What needs to be made atomic?
  - AllocateBlock(); FreeBlock()
  - Write(): AllocateBlock() and update inode
- What are naturally atomic?
  - WriteRawData(); ReadRawData()

What if a system has multiple FS?

- System boots from primary boot partition on primary disk (configuration parameter)
- In UNIX systems:
  - Root dir in boot partition becomes "/"
  - Other filesystems are "mounted" into the namespace
  - For instance: (assume /dev/rzla is boot partition)
    - mount /dev/rzlb /usr1
    - mount /dev/rz2a /usr2
    - mount /dev/rz2b /usr3
- In Microsoft systems:
  - Each filesystem is assigned a "drive letter" in range [A-Z], prepended to filenames

Implementation of mount on UNIX

- Setting a flag in in-memory copy of inode for dir
  - Indicating the dir is a mount point
- A field in inode points to an entry in mount table
  - Indicating which device is mounted here
- The mount table entry contains a pointer to the superblock of the file system on that device

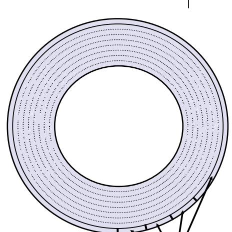
[lec20] Disk Layout of a FS

- Disk layout of a file system

Boot block	Super block	File metadata (i-node in Unix)	File data blocks

Early Unix Disk Layout

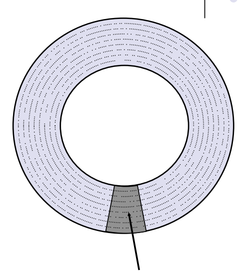
- An array of i-nodes in outermost cylinders
- i-node number is index into the i-node array
- Problems
  - i-nodes are far away from data blocks
  - fixed max number of files
  - What happens to ls –R?



i-node array

Unix BSD 4.2 Disk Layout

- How often do you read i-nodes (of same dir) without reading any file blocks?
  - 4 times more often than reading together
  - examples: ls –l, ls -R
- Solution:
  - A portion of the i-node array on each cylinder
  - Multiple i-nodes of same dir are on the same block



i-node subarray

Why do inodes start from 2?

- Usually, inode 0 is reserved because a return value of 0 usually signals an error.
- In ext2:
  - #define EXT2_BAD_INO 1 /* Bad blocks inode */
    #define EXT2_ROOT_INO 2 /* Root inode */
    #define EXT2_BOOT_LOADER_INO 5 /* Boot loader inode */
    #define EXT2_UNDEL_DIR_INO 6 /* Undel. Dir. inode */
- In ext3:
  - #define EXT3_BAD_INO 1 /* Bad blocks inode */
    #define EXT3_ROOT_INO 2 /* Root inode */
    #define EXT3_BOOT_LOADER_INO 5 /* Boot loader inode */
    #define EXT3_UNDEL_DIR_INO 6 /* Undelete directory inode */
    #define EXT3_RESIZE_INO 7 /* Res. group descriptors inode */
    #define EXT3_JOURNAL_INO 8 /* Journal inode */

Readings

UNIX File System

- UNIX (1969)
  - One of the most popular operating systems
  - Evolving since escaping from Bell lab early 70's
  - Written in C with small kernel
- Other important events in 1969?
  - Man landed on the Moon
  - Internet was born (4 nodes!)
  - Linus Torvalds was born

UNIX File System (UFS)

- Overall structure of the file storage and control on UNIX
- One of the most significant aspects of UNIX

UFS Overview

- Anything can be viewed as a file: devices, networking
- All files as streams of bytes in UNIX kernel
- Hierarchical, directory-based
- Four types of files
  - regular file: ASCII files
  - directory-type file: map file names to the contents in a directory
  - special file: printers, terminals, other devices
  - named pipe: FIFO

UFS Overview (cont)

- inode: index node representing a file
  - Every access to the file must make use of the information of the inode.
- UNIX supports multiple file systems
  - one in charge of UNIX system startup
  - others can be "mounted" or "removed"
    - on disk, CD-ROM, floppy, over network

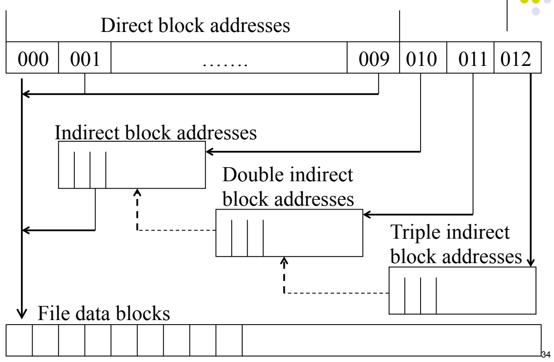
Inode Structure

- Administrative information and physical data addresses of the file:
  - file mode (access and type info)
  - count of hard links
  - owner id
  - group id
  - time of last file access
  - time of last file modification
  - time of last inode modification
  - file size
  - file addresses
  - semaphore

File Addresses in an Inode

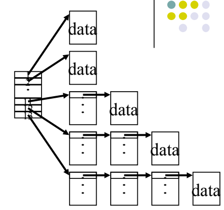
- Block as the basic address unit, BLOCKSIZE is constant
- 13 three-byte pointers point either directly or indirectly to the disk blocks containing the data contents of the file.
  - Pointers 0-9: addresses of direct blocks containing file data
  - Pointer 10: address of a single indirect block, a block containing the addresses of direct blocks
  - Pointer 11: address of a double indirect block, a block containing the addresses of single indirect blocks which contain the addresses of direct blocks
  - Pointer 12: address of a triple indirect block

File Addresses in an INODE



File Addresses in an INODE

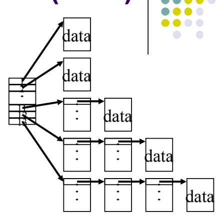
- Assume blocksize = 1K
  - a block contains 1024 / 4 = 256 block addresses
- direct block address: 10K
  - indirect block addresses: 256K
  - double indirect block addresses: 256 * 256K = 64M
  - tripe indirect block addresses: 256 * 64M = 16G



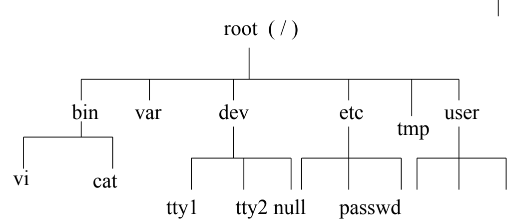
What happens in accessing block 23, 5, 340?

File Addresses in an INODE (cont)

- addresses for special files
  - only first two pointers are used: device channel number, minor device number
- addresses for named pipes
  - named pipes are limited to 10K



External View



Internal View of a File System

Block 0	Boot Block
1	Super Block
2 . . .	i-list Blocks ……
FD-1	Last i-list Blocks
1	First Data Block
. . .	Data Blocks
T	Last Data Blocks

Internal View of a File System

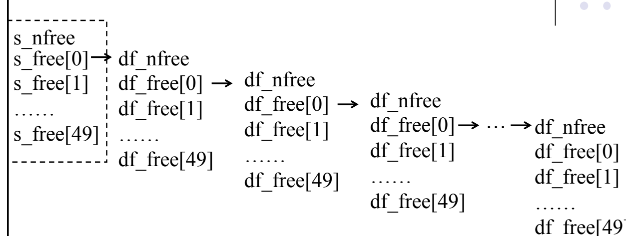
- Boot Block: the first block in a UNIX file system, contains the boot program and other initialization information or unused
- Super Block: always the second block, contains specific information about the file system
- i-list Blocks:
  - list of inodes for the file system
  - Contiguous
  - always follows the super block
  - number specified by the system admin at format time
- Data Blocks: immediately follow the i-list and consume the rest of the blocks

Free Blocks Organization

- All free blocks appear in *free-block chain*.
- free-block chain is a linked list of *free-block address blocks*.
- Each free-block address block holds up to 50 4-byte addresses of free blocks.
- The superblock entry s_free[0] heads the free-block chain
- Free-block address block structure:

```
struct {
        int df_nfree;
        d_addr_t df_free[NICFREE];
}
```


Free-Block List Structure



Free-Block List Maintenance

- *s_flock* flag in the super block to prevent two processes from updating the free list at the same time.
- Block allocation:
  - If s_nfree > 1, allocate s_free[s_nfree-1], s_nfree--
  - If s_nfree = 1, return block s_free[0], copy over content of block pointed by s_free[0]
- Block de-allocation:
  - if s_nfree < 50, place free block to s_free[s_nfree], s_nfree++
  - If s_nfree = 50, use the de-allocated block as new free block address block, copy block content, set s_free[0], s_nfree = 1

Free-Inode List Maintenance?

Block 0	Boot Block
1	Super Block
2 . . .	i-list Blocks ……
FD-1	Last i-list Blocks
1	First Data Block
. . .	Data Blocks
T	Last Data Blocks

Free-Inode List Maintenance

- A free inode is an unused inode, i.e. with zero mode.
- The superblock contains a cache of indices of free inodes in an array s_node, 100 elements
- s_ilock for concurrency control
- inode allocation:
  - case 1: s_ninode > 0, superblock cache contains a free inode address
  - case 2: s_ninode = 0, search the i-list for free inode, refill the s_node array
- inode de-allocation:
  - case 1: s_ninode < 100, put it in the s_node array
  - case 2: s_ninode = 100, do nothing

File System Hierarchy

- Unix has a tree structure
  - Directories can contain files
  - Directories can contain other directories
  - Naming a file looks like /dir1/dir2/filename
  - That' s called an absolute pathname
  - What' s a relative pathname?
    - You are "in" /dir1
    - You say "dir2/filename"
    - Note absence of leading '/' in relative pathname

Readings

- Chapters 10-11