

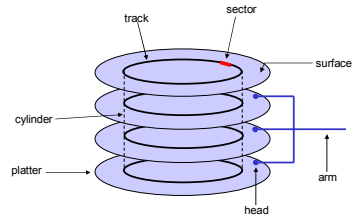
CSE 451: Operating Systems
Spring 2006

Module 14
From Physical to Logical:
File Systems

John Zahorjan
zahorjan@cs.washington.edu
Allen Center 534

Physical disk structure

- Disk components
 - platters
 - surfaces
 - tracks
 - sectors
 - cylinders
 - arm
 - heads



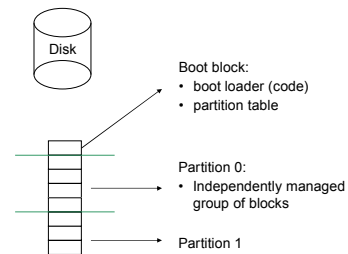
5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 2

Disk performance

- Performance depends on a number of steps
 - **seek**: moving the disk arm to the correct cylinder
 - depends on how fast disk arm can move
 - seek times aren't diminishing very quickly (*why?*)
 - **rotation (latency)**: waiting for the sector to rotate under head
 - depends on rotation rate of disk
 - rates are increasing, but slowly (*why?*)
 - **transfer**: transferring data from surface into disk controller, and from there sending it back to host
 - depends on density of bytes on disk
 - increasing, and very quickly
- When the OS uses the disk, it tries to minimize the cost of all of these steps
 - particularly seek and rotation

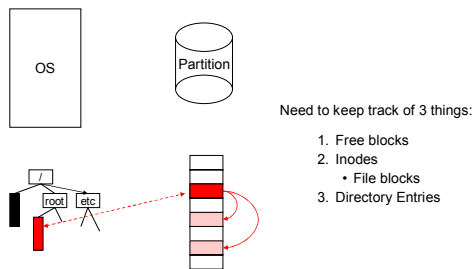
5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 3

From Physical To Logical: Low Level



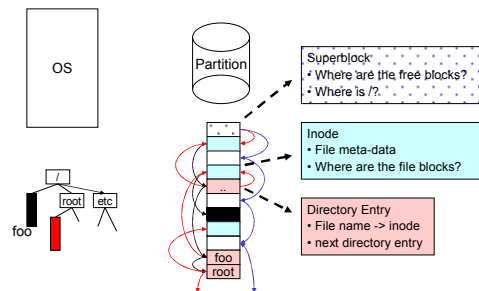
5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 4

From Physical To Logical: File Systems



5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 5

A Strawman Approach



5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 6

Formatting: Preparing the Empty File System

5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 7

Evaluation

Positives:

- Simple
- No preset limits on:
 - File size
 - Number of files
 - Disk size

Negatives:

- Incredibly slow:
 - Many block transfers to read a directory
 - Many seek / latency delays
 - Direct access to file bytes requires walking linked list of data blocks
- Internal fragmentation
 - 1KB allocated for every inode
 - 1KB allocated for every directory entry

5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 8

Solutions

- Performance
 - Pack logical items into physical blocks
 - Inodes
 - Directory entries
 - 1 seek / latency retries many items
- Keep items small
 - Fewer files than blocks \Rightarrow fewer bits in an inode name than a block name

5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 9

The original Unix file system

- Dennis Ritchie and Ken Thompson, Bell Labs, 1969
- “UNIX rose from the ashes of a multi-organizational effort in the early 1960s to develop a dependable timesharing operating system” – Multics
- Designed for a “workgroup” sharing a single system
- Did its job exceedingly well
 - Although it has been stretched in many directions and made ugly in the process
- A wonderful study in engineering tradeoffs

5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 10

All disks are divided into five parts ...

- Boot block
 - can boot the system by loading from this block
- Superblock
 - specifies boundaries of next 3 areas, and contains head of freelists of inodes and file blocks
- i-node area
 - contains descriptors (i-nodes) for each file on the disk; all i-nodes are the same size; head of freelist is in the superblock
- File contents area
 - fixed-size blocks; head of freelist is in the superblock
- Swap area
 - holds processes that have been swapped out of memory

5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 11

Disk Layout

Direct Access to inodes:

Inode K is in block $K / (\text{BLOCK_SIZE} / \text{sizeof}(\text{inode}))$
 At offset $K \% (\text{BLOCK_SIZE} / \text{sizeof}(\text{inode}))$

Directory entries are packed into blocks in a manner similar to inodes

5/19/2006 © 2006 Gribble, Lazowska, Levy, Zahorjan 12

The tree (directory, hierarchical) file system

- A directory is a flat file of fixed-size entries
- Each entry consists of an i-node number and a file name

i-node number	File name
152	.
18	..
216	my_file
4	another_file
93	oh_my_god
144	a_directory

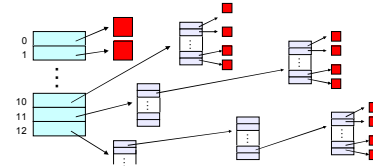
5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

13

The "block list" portion of the i-node (Unix Version 7)

- Must be able to represent very small and very large files...
 - with minimal chaining...
 - and leaving inodes small
- Each inode contains 13 block pointers
 - first 10 are "direct pointers" (pointers to blocks of file data)
 - then, single, double, and triple indirect pointers



5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

14

So ...

- Data pointers occupy only 13 x 4B in the inode
- Can get to 10 x 512B = a 5120B file directly
 - (10 direct pointers, blocks in the file contents area are 512B)
- Can get to 128 x 512B = an additional 65KB with a single indirect reference
 - (the 11th pointer in the i-node gets you to a 512B block in the file contents area that contains 128 4B pointers to blocks holding file data)
- Can get to 128 x 128 x 512B = an additional 8MB with a double indirect reference
- Can get to 128 x 128 x 128 x 512B = an additional 1GB with a triple indirect reference
- Maximum file size is 1GB + a smidge

5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

15

- A later version of Bell Labs Unix utilized 12 direct pointers rather than 10
 - Why?
- Berkeley Unix went to 1KB block sizes
 - What's the effect on the maximum file size?
 - 256x256x256x1K = 17 GB + a smidge
 - What's the price?
- Suppose you went to 4KB blocks?
 - 1Kx1Kx1Kx4K = 4TB + a smidge

5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

16

File system consistency

- Both i-nodes and file blocks are cached in memory
- The "sync" command forces memory-resident disk information to be written to disk
 - system does a sync every few seconds
- A crash or power failure between sync's can leave an inconsistent disk
- You could reduce the frequency of problems by reducing caching, but performance would suffer big-time

5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

17

i-check: consistency of the flat file system

- Is each block on exactly one list?
 - create a bit vector with as many entries as there are blocks
 - follow the free list and each i-node block list
 - when a block is encountered, examine its bit
 - If the bit was 0, set it to 1
 - if the bit was already 1
 - if the block is both in a file and on the free list, remove it from the free list and cross your fingers
 - if the block is in two files, call support!
 - if there are any 0's left at the end, put those blocks on the free list

5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

18

d-check: consistency of the directory file system

- Do the directories form a tree?
- Does the link count of each file equal the number of directories links to it?
 - I will spare you the details
 - uses a zero-initialized vector of counters, one per i-node
 - walk the tree, then visit every i-node

5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

19

File System Performance 1: Disk scheduling

- Seeks are very expensive, so the OS attempts to schedule disk requests that are queued waiting for the disk
 - FCFS (do nothing)
 - reasonable when load is low
 - long waiting time for long request queues
 - SSTF (shortest seek time first)
 - minimize arm movement (seek time), maximize request rate
 - unfairly favors middle blocks
 - SCAN (elevator algorithm)
 - service requests in one direction until done, then reverse
 - skews wait times non-uniformly (why?)
 - C-SCAN
 - like scan, but only go in one direction (typewriter)
 - uniform wait times

5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

20

File System Performance 2: Layout

- Disk scheduling attempts to minimize the impact of blocks needed at the moment located widely over the disk
 - How effective do you imagine it is / can be?
- An alternative (complementary) approach is to allocate blocks likely to be needed together near each other?
 - Which blocks might be needed together?
- A related approach is to observe block usage patterns and move them near each other
 - The "pipe organ" layout is the simplest example

5/19/2006

© 2006 Gribble, Lazowska, Levy, Zahorjan

21