EECS 482
Introduction to Operating Systems

Winter 2018

Harsha V. Madhyastha

Recap: File system structure

- Abstraction:
  - Every file is an array of bytes
  - Can store large number of files
- File header contains
  - Metadata
  - Root of index structure to locate file's contents
- Directory: (name, header disk block#) entries
  - Stored similar to files
  - Limited interface for users/apps to update

Directory Example

/ directory

Name	LBN
“home”	100
“usr”	35
“tmp”	43
“foo.txt”	254

home directory

Name	LBN
“harshavm”	23
“barisk”	99
	0
	0

harshavm directory

Name	LBN
“482.txt”	44
	0
“src”	55
“foo.txt”	33

Example: /home/harshavm/482/notes

1. Read file header for / (root directory)
   - Contains pointers to data blocks of / directory
2. Read data blocks of /
   - Contains list of the files and directories in /. For each entry, contains mapping from name → header's disk block #
   - One of those entries is "home"
3. Read file header for /home
4. Read data blocks for /home
5. Read file header for /home/harshavm
6. Read data blocks for /home/harshavm
7. Read file header for /home/harshavm/482
8. Read data blocks for /home/harshavm/482
9. Read file header for /home/harshavm/482/notes
10. Read first data block for /home/harshavm/482/notes

Eliminated by caching file header for current working directory

Unified view of multiple storage devices

- Combine multiple storage devices into a file system
  - Each device contains own file system (starting with its root)
  - A directory entry can point to the root of a different device
- Example: loginlinux.engin.umich.edu

  / (root)

    bin (same device as /)

    etc (same device as /)

    tmp (separate storage device)

    afs (network storage “device”)
- Directory entry: 1) file, 2) directory, or 3) device

Data types for disk blocks

- File systems store lots of data structures on disk
  - Data blocks
  - Directories
  - File headers (inodes), indirect blocks
  - Free lists (bitmaps of used, unused blocks)

- How can you tell what type a block is?
  - Each is just a fixed size # of bytes
  - By what points to it (just like data structure in memory)

File Cache

- Caches file system blocks in physical memory
  - Each block indexed by (device, LBN)
- Should cache be in phys. or virtual memory?
- Should cache be write-through or write-back?
  - Write-through: poor performance
  - Write-back: loses data on OS crash, power failure
- Current file systems:
  - Write-back but bound time dirty data stays in cache
  - Background daemon writes dirty pages

File cache vs. Virtual memory

- Both use physical memory as a cache for disk
  - Virtual memory: Use disk for increased capacity
  - File systems: Use memory for faster performance

- Both compete for physical memory
  - Another instance of local vs. global replacement
  - Common to use global replacement
  - Why have two separate mechanisms?

Memory-mapped files

- Use the paging system to cache both virtual address space **and** file system data
  - Map file into a virtual address space
  - Point the backing store for that part of the address space at the file's data blocks

- Example: How to load a program executable from disk to memory?

Project 3

- Multi-process test cases
  - Needed even to test swap-backed no fork
  - Call fork() before any calls to vm_map
- To test fork(), write a test for every state a page can be in at the time of fork()
  - Swap-backed vs file-backed
  - Resident vs non-resident
  - Shared vs unshared
  - …
- Reminder about Honor Code

Multiple updates and reliability

- File system must ensure reliability/durability
  - Okay to lose data in address space
  - Data must survive crashes and power outages
  - Assume: update of one block atomic and durable
  - Challenge: Crashes in midst of multi-step updates

- Example: Transfer $100 between accounts
  1. Deduct $100 from savings
  2. Add $100 to checking
- Crash between steps 1 and 2 = lose $100

Other Examples

- Move file from directory a to directory b
  1. Delete file from dir a
  2. Add file to dir b

- Create new (empty) file
  1. Update directory to point to new file header
  2. Write new file header to disk

How to fix these problems?

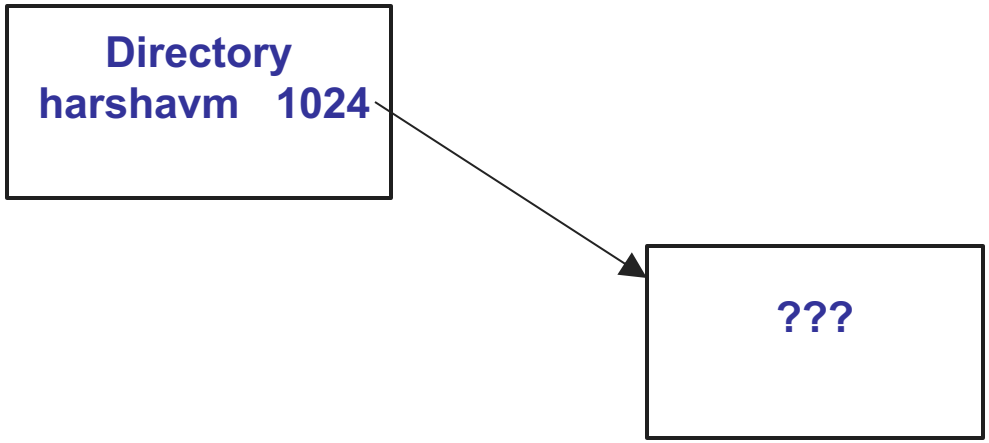
Ordering of updates

- Careful ordering can fix some problems

- Example: Create file 482.txt in dir harshavm
  - Update directory first?
  - Creat inode for new file first?

Ordering of updates

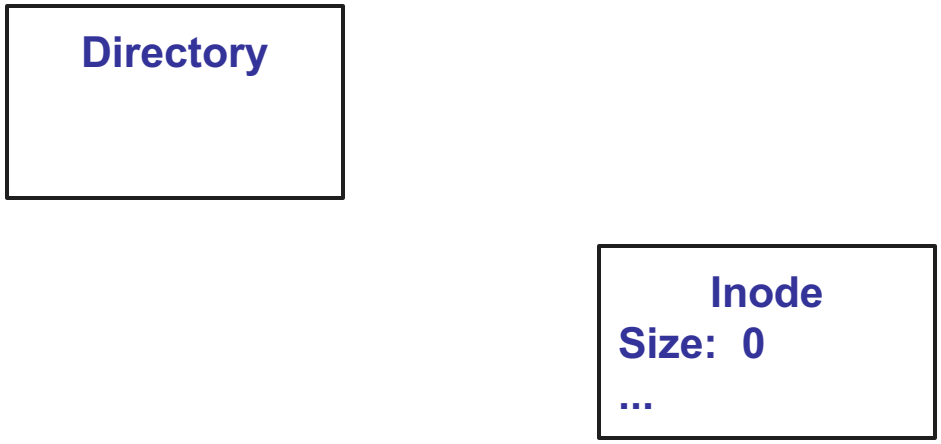
- Careful ordering can fix some problems:
  - For example, creating file 482.txt in dir harshavm
  - Update directory first?



Never have a pointer from valid block to invalid one!

Ordering of updates

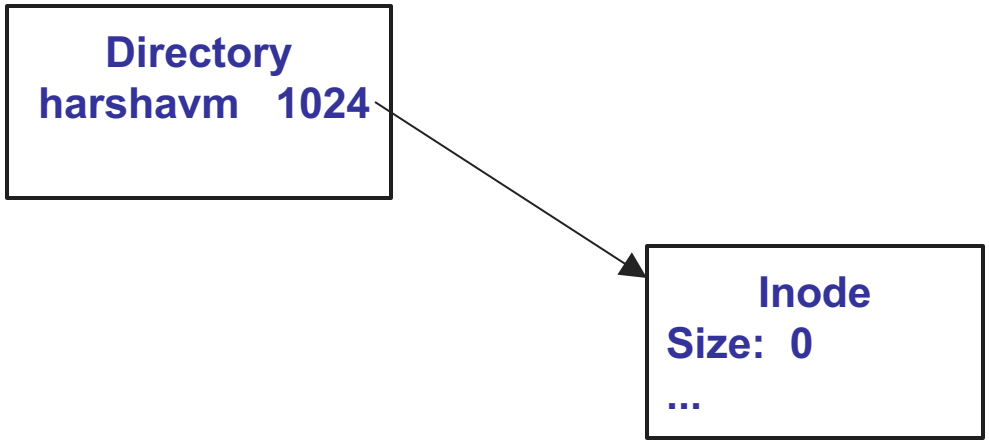
- Careful ordering can fix some problems:
  - For example, creating file 482.txt in dir harshavm
  - Create inode first?



OK to modify unreachable blocks on disk

Ordering of updates

- Careful ordering can fix some problems:
  - For example, creating file 482.txt in dir harshavm
  - Create inode first?



Careful ordering goes from one consistent state to another

Ordering not always enough

- Example: Create file and update free block list
  1. Write new file header to disk
  2. Update directory to point to new file header
  3. Write the new free map

- Is 1, 2, 3 correct?
- What about 3, 1, 2?

- What about bank account example?