File System Usability

Bringing It Together

- Virtual address spaces are mapped onto physical address spaces in physical memory
  - Pages are mapped to frames!
  - Lots of policies to consider: demand paging, page replacement, working set, …
- The heap is part of a process's virtual address space (and is thus paged!)
  - Can be explicitly or automatically managed
- I/O devices are important for system usability.
  - Each one consists of the device, the controller, a bus, and a port
  - One of the most important I/O devices is the disk, which provides stable storage.
- File systems make the disk usable
  - Goals include: efficiency, usability, reliability
  - File system metadata is in a superblock
  - File metadata is in a file header (inode in Linux/Unix)
  - File data is on disk in some format (contiguous, …)
  - File systems demonstrate mechanical sympathy, including through reserved space, block groups, first-free allocation…)

Directories

OR

So we can store the file, but can the user find the file?

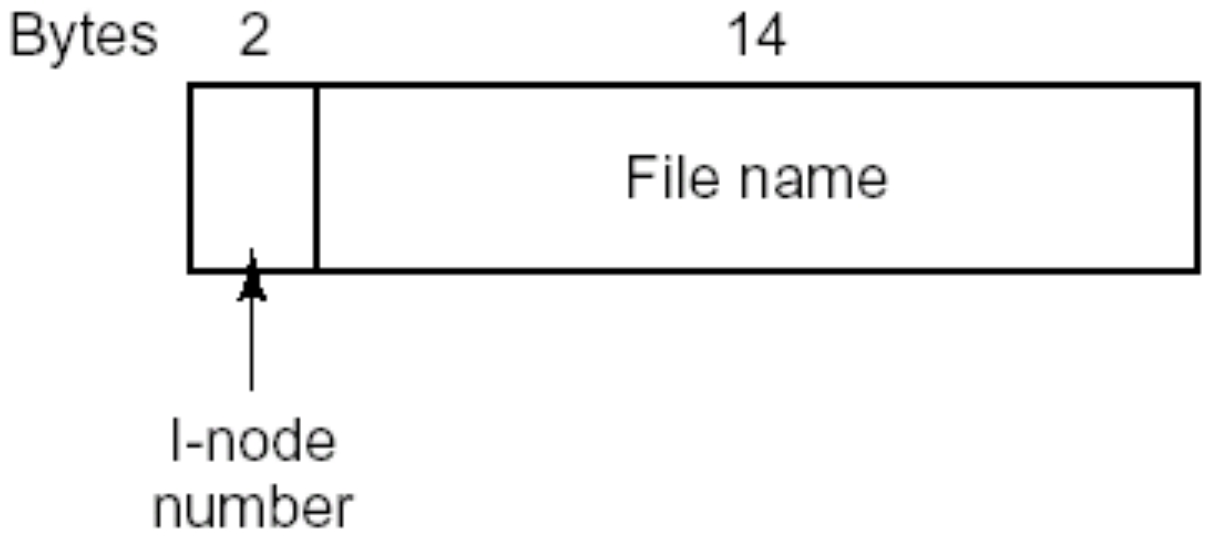
So a directory is a folder.

Computer Scientists call them directories.

The rest of the world calls them folders.

Directories

- Directories create a *name space* for the files
- Look to us like a list of file names
- Really a collection of mappings from file name to file number
  - Each mapping is known as a *directory entry*
  - The *file number* indicates the location of the file header in the file system
  - Stored in a file
- Directories can only be modified in kernel mode
  - Ensure integrity of mapping
  - Application programs *can* read directories



A simple UNIX directory entry

Directory Strategies: Make-It-Work Strategy

One name space for the entire disk.

- Use a special area of the disk partition to hold the directory
- Directory contains <name, file number> pairs
- If one user uses a name, no one else can

File Name	File Number
.user1_bashrc	27
.user2_bashrc	30
firefox	3392
hw2.txt	7

Directory Strategies: Simple User-Based Strategy

- Each user has a separate directory
  - But each user only has that one directory
- Directory contains <name, file number> pairs
- Once a user uses a name, that user cannot reuse the name

User1 Directory

File Name	File Number
.bashrc	27
firefox	3392

User2 Directory

File Name	File Number
.bashrc	30
hw2.txt	7

Directory Strategies: Multi-level Directories

Tree-structured hierarchical name space (all modern OSs)

- There is one special root directory for the system
- Each user has a home directory and may create subdirectories
- Directory contains <name, file number> pairs
  - The file referred to by <name> may be another directory
- Each file name must be unique within that directory

Question

Every directory has a file header.

A. True
B. False

Given only the inode number (*inumber*) the OS can find the inode on disk.

A. True
B. False

So... given all that, how do we find the blocks of a file?

Let’s assume FFS…

Finding a File

How do you find the blocks of a file?

- Where do we find the pointers to the data blocks?
  - Find the file header (inode); it contains pointers to file blocks
- How do we find the file header?
  - Use its inumber
- How do we find its inumber?
  - Read the directory that contains the file
- To find the directory…
- But wait! The directory is a file…
- So where do we start?

Example:
Read file `/users/ans/wisdom.txt`

How do we find it?

- `wisdom.txt` is a file
- `ans/` is a directory that contains the inumber for `wisdom.txt`
  - Locate `ans/`, read directory
- `users/` is a directory that contains the inumber for `ans`
  - Locate `users/`, read directory
- How do you find the inumber for `Users/`?
  - `/` is a directory that contains the inumber for `Users/`
  - In Unix, `/`'s inumber is 2
    - (whew! At least we know how to find that! Or do we? … disk layout up soon!)

How much work was that?

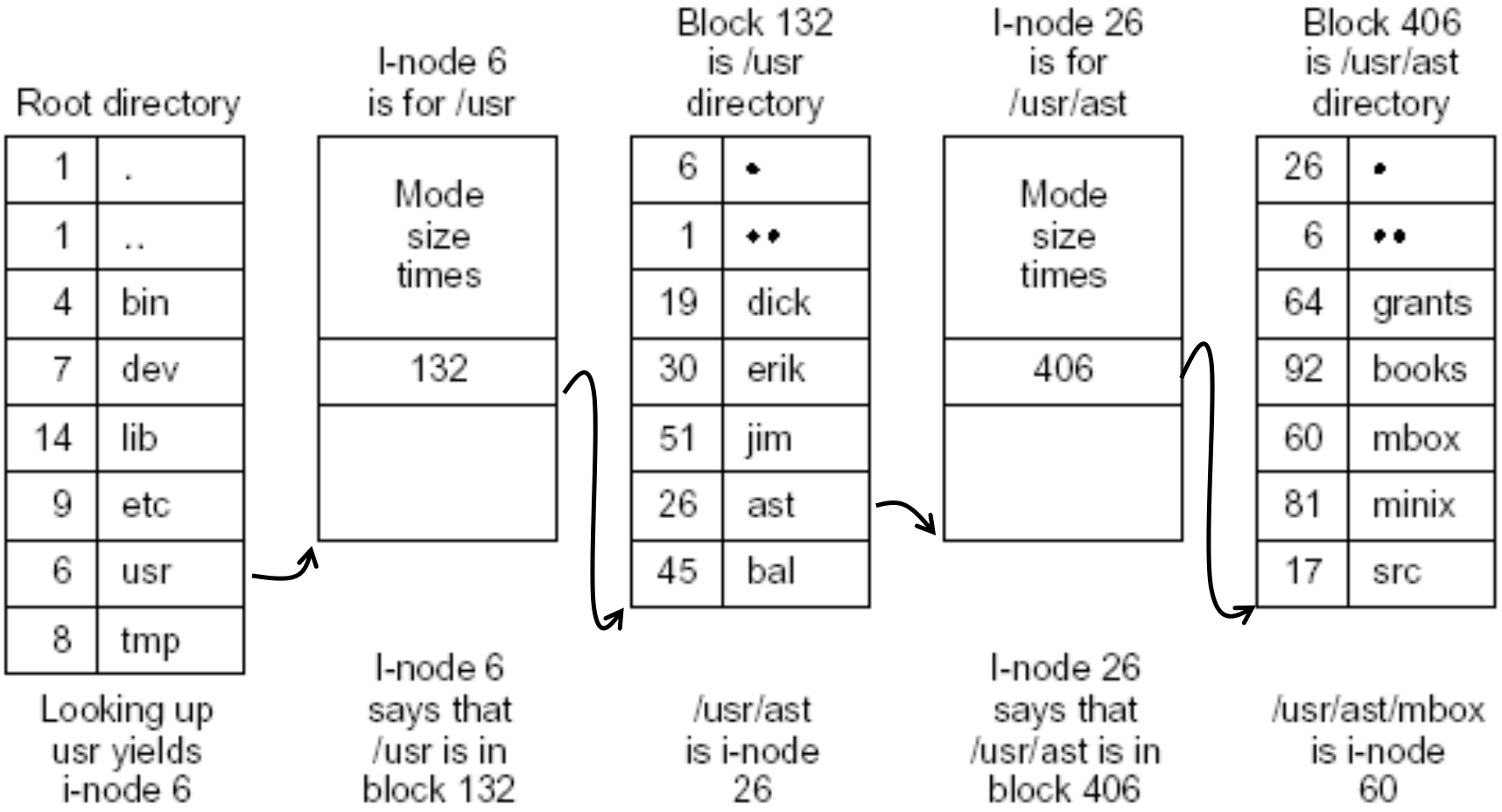
How many disk accesses are needed to access file `/users/ans/wisdom.txt`?

1. Read the inode for `/` from a fixed location
2. Read the data block for root
3. Read the inode for `users/`
4. Read the data block for `users/`
5. Read the inode for `ans/`
6. Read the data block for `ans/`
7. Read the inode for `wisdom.txt`
8. Read the data block for `wisdom.txt`

"To make anything faster, add a (buffer) cache!"

Finding a File: Another Example

Another Example: Reading a File



The steps in looking up */usr/ast/mbox*.

Root directory is a file itself and its inode is at a fixed location on disk.

Another Example: Reading a File Text Description

The steps in looking up /usr/ast/mbox on the disk

1. Retrieve root directory inode from fixed location on disk
2. Retrieve root directory data block from location given in inode.
3. Look up usr in root directory data block; its information is in inode 6
4. Go to inode 6 and determine which block holds the data for /usr
   - This is block 132
5. Get block 132, the /usr directory data, and lookup ast; its information is in inode 26
6. Go to inode 26 and determine which block holds the data for /usr/ast
   - This is block 406
7. Get block 406, the /usr/ast data, and lookup mbox; its information is in inode 60
   - Woot! We have found our email!

Question

Assuming the cache is empty, how many disk accesses would it take to access the first data block of /classes/cs439/problem_sets/8_ps.txt?

You may assume each directory is a single data block.

A. 8
B. 14
C. 16
D. 22

Putting It All Together

Putting It All Together: A Typical Read Call

1. User process requests a read from a device
2. File system checks if data is in the buffer cache. If not,
   a) File system will call the device driver with a request for input
   b) Device driver tells the DMA controller what to do and blocks itself
   c) DMA controller transfers the data to the kernel buffer when it has all been retrieved from the device
   d) DMA controller interrupts the CPU when the transfer is complete
   e) Interrupt handler signals thread in the device driver, which is then moved to the ready queue
3. Data is transferred to the user address space
4. System call completes and returns

Each Step *is* a read()

How many disk accesses are needed to access file `/users/ans/wisdom.txt`?

1. Read the inode for `/` from a fixed location
2. Read the data block for root
3. Read the inode for `users/`
4. Read the data block for `users/`
5. Read the inode for `ans/`
6. Read the data block for `ans/`
7. Read the inode for `wisdom.txt`
8. Read the data block for `wisdom.txt`

"To make anything faster, add a (buffer) cache!"

Optimizations

The Path

path: string that identifies a file or directory

- absolute (if it starts with “/”, the root directory)
- relative (with respect to the current working directory)

CWD

- Maintain the notion of *current working directory* (CWD)
- Users can now specify relative file names
- Filesystem can cache the data blocks of CWD
  - In its *buffer cache*

File System API

Or Making the File System Usable for the Programmer

What We Have

We now know how to do the following:

- Given a file header, find the data blocks of a file
- Given a file header, extend the file (if necessary)
- Find a file header from a human-readable path

But we don’t want our users to have to know how to do all of this!

We know a lot of common file system operations, and how they work. (Thanks, Project 2!)

FS API: Ones We Know

- open()/close()
- read()/write()
- create()
- seek()
- …

link()

Creates a hard link---a new name for some underlying file

On link(), the file system:

- Adds entry to the directory with new name
- Increments counter in file header that tracks the number of links a file has

This new name points to the same underlying file.

unlink()

Removes an existing hard link, decrementing the link count in the file metadata.

If link count is now zero, file and its resources are deleted.

To delete(), the file system:

- Finds directory containing the file, and removes the file entry from the directory
- Clears headers
- Frees the disk blocks used by the file and its headers

Also fsync()

- fsync() does not return until all data is written to persistent storage
- Flushes the buffer cache

But does it really?

Summary

- File System Efficiency
  - Space
    - File layout and allocation
  - Time
    - Reserved space, placement of metadata structures
    - File layout and allocation
    - Disk choice (sorta)
- File System Usability
  - Named data
  - Directories
  - API