Operating Systems
ECE344

Lecture 12: File System

Ding Yuan

Happy April Fools'

Linus Torvalds To Join Microsoft To Head Windows 9 Project

APRIL 1, 2013 *By* ABHISHEK 26 COMMENTS

This is breaking bad. This is big. Linus Torvalds, the creator of Linux, and a champion of free and open source software has finally call it a day and has agreed to join Microsoft as the project head of the upcoming Windows 9 project. **According to Bloomberg, Linus will be working on a new Kernel design for Microsoft that will make, usually vulnerable, Windows OS virtually impossible to be infected by viruses and malware.**

Review

- What is a replacement algorithm?
  - What problem does it solve?
- Name a few replacement algorithm
- Optimal algorithm
  - What is it?
- What is Belady's anomaly?

Review (LRU)

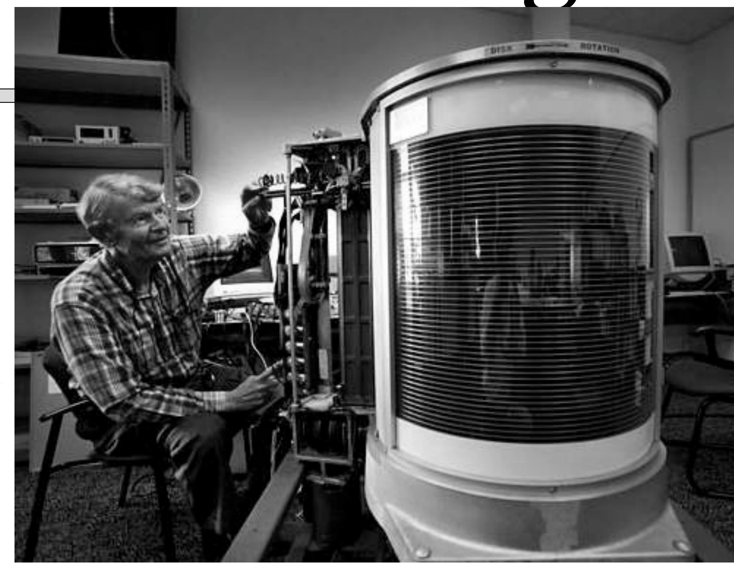
- What is it?
- Why does it work?
- Can you implement it?
  - Compare to Belady's algorithm
- Does VM systems use it in practice? Why?
- What is NRU?
- What is CLOCK?

Review (working set)

- What is the "working set" of a process?
- For multiple processes
  - Local vs. global replacement
  - Working set algorithm

What problem are we solving?

- Data storage & access
  - Super important
- One of the fastest growing industry
  - Why?
  - Driven by technology



1953, IBM, 24 inches, 3.75MB, 1KB/sec, > $150,000

2013, Seagate, 3.5 inches, 4TB, 600MB/sec, < $200

What problem are we solving?

- One of the fastest growing industry
  - Why?
  - Driven by technology
  - Driven by demand
    - Mainframe storage: IBM, Memorex
    - PC storage: Seagate, DEC, Quantum, etc.
    - Enterprise Storage: EMC, NetApp, etc.
    - Cloud Storage: Dropbox, Google Drive, etc.

File Systems

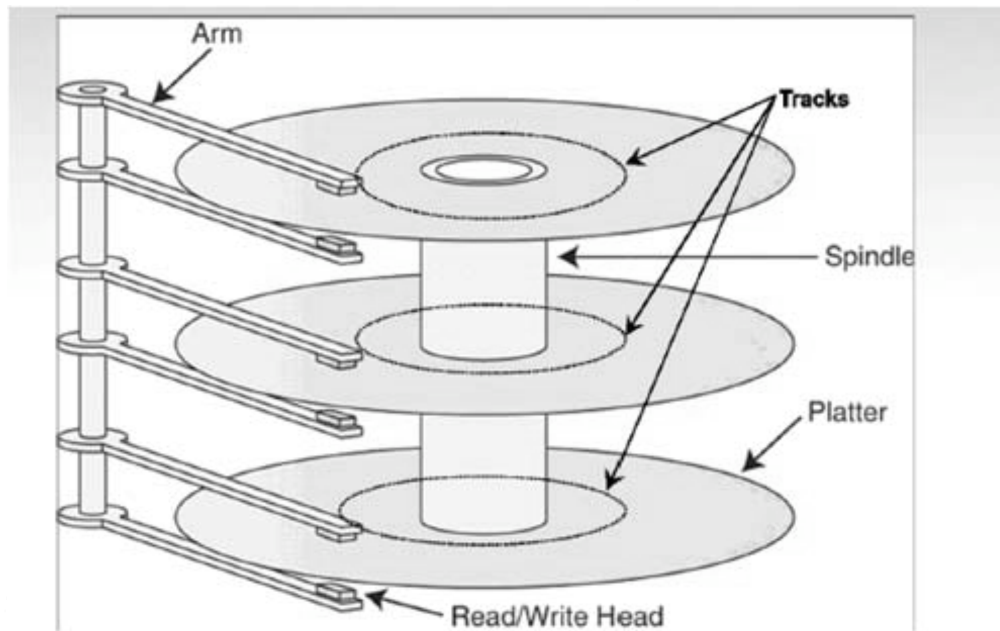
- First we'll discuss properties of physical disks
  - Structure
  - Performance
  - Scheduling
- Then we'll discuss how we build file systems on them
  - Files
  - Directories
  - Sharing
  - Protection
  - File System Layouts
  - File Buffer Cache
  - Read Ahead

Disks and the OS

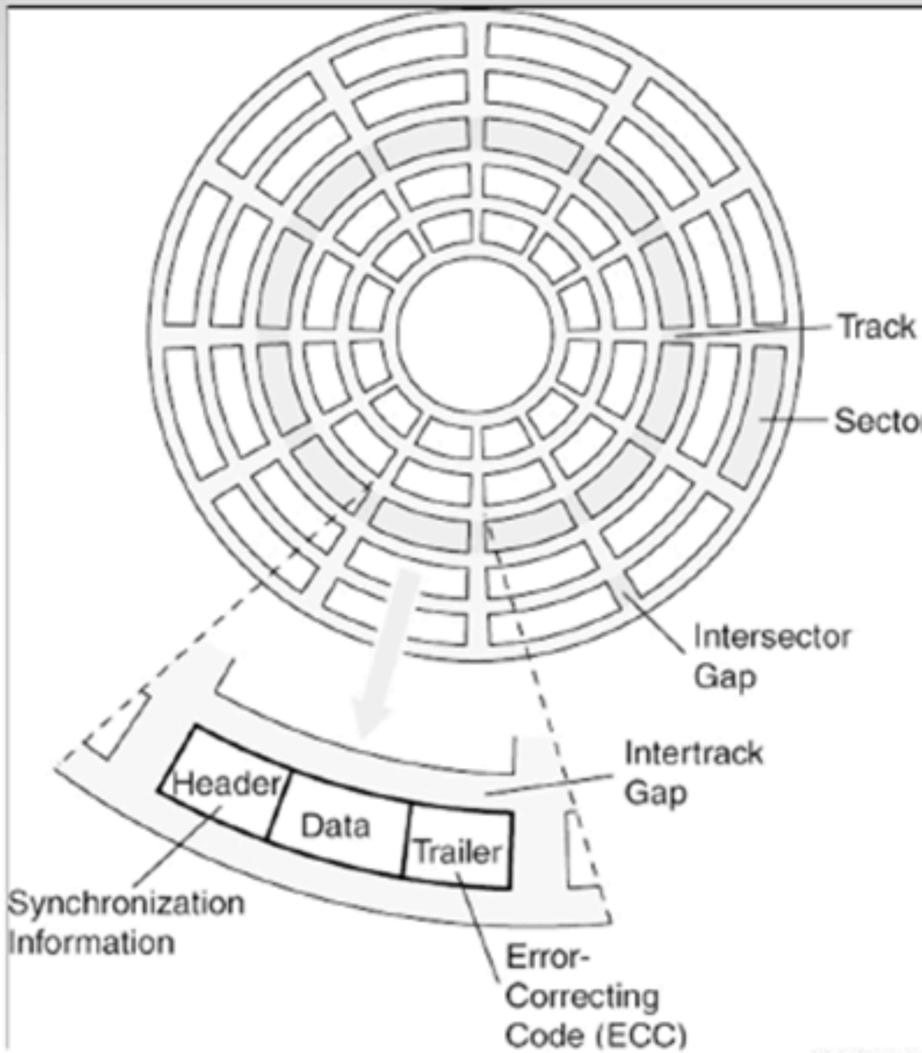
- Disks are messy physical devices
  - Errors, bad blocks, missed seeks, etc.
- The job of the OS is to hide this mess from higher level software
  - Low-level device control (initiate a disk read, etc.)
  - Higher-level abstractions (files, databases, etc.)

How hard disk work?

- http://www.youtube.com/watch?v=kdmLvl1n82U
- Disk components
  - Platters
  - Surfaces
  - Tracks
  - Cylinders
  - Sectors
  - Arm
  - Heads



Another View of Disk

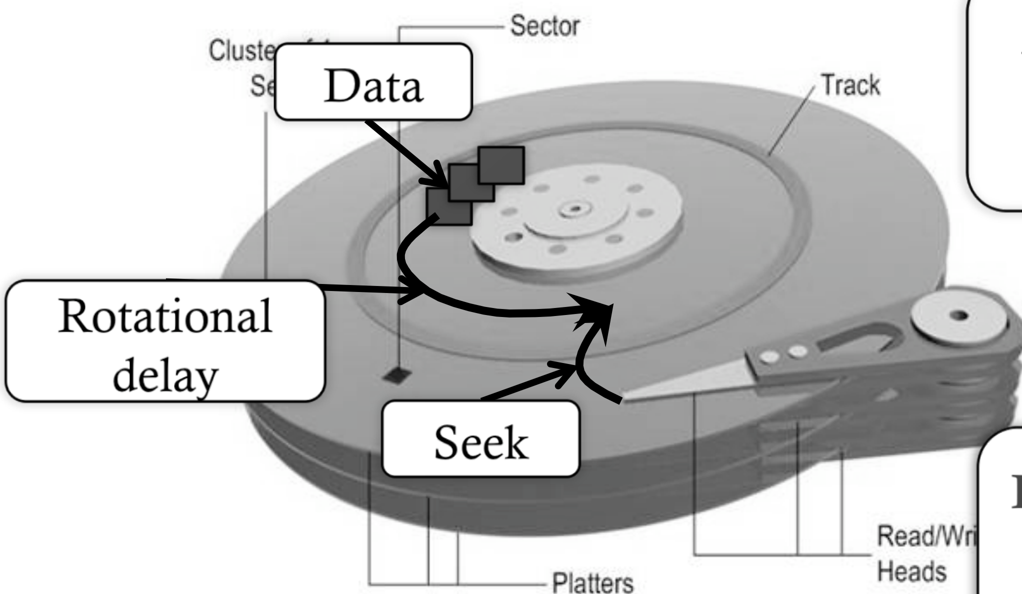


Disk Interaction

- Specifying disk requests requires a lot of info:
  - Cylinder #, surface #, sector #, transfer size…
- Older disks required the OS to specify all of this
  - The OS needed to know all disk parameters
- Modern disks are more complicated
  - Not all sectors are the same size, sectors are remapped, etc.
- Current disks provide a higher-level interface (SCSI)
  - The disk exports its data as a logical array of blocks [0…N]
    - Disk maps logical blocks to cylinder/surface/track/sector
  - Only need to specify the logical block # to read/write
  - But now the disk parameters are hidden from the OS

Disk Performance

- Random disk access is ***SLOW***!



Access data sequentially: only suffer one seek and rotational delay

Random disk access: suffers one seek and rotational delay every time!

Disk Performance

- Disk request performance depends upon three steps
  - Seek – moving the disk arm to the correct cylinder
    - Depends on how fast disk arm can move (increasing very slowly)
  - Rotation – waiting for the sector to rotate under the head
    - Depends on rotation rate of disk (increasing, but slowly)
  - Transfer – transferring data from surface into disk controller electronics, sending it back to the host
    - Depends on density (increasing quickly)
- When the OS uses the disk, it tries to minimize the cost of all of these steps
  - Particularly seeks and rotation

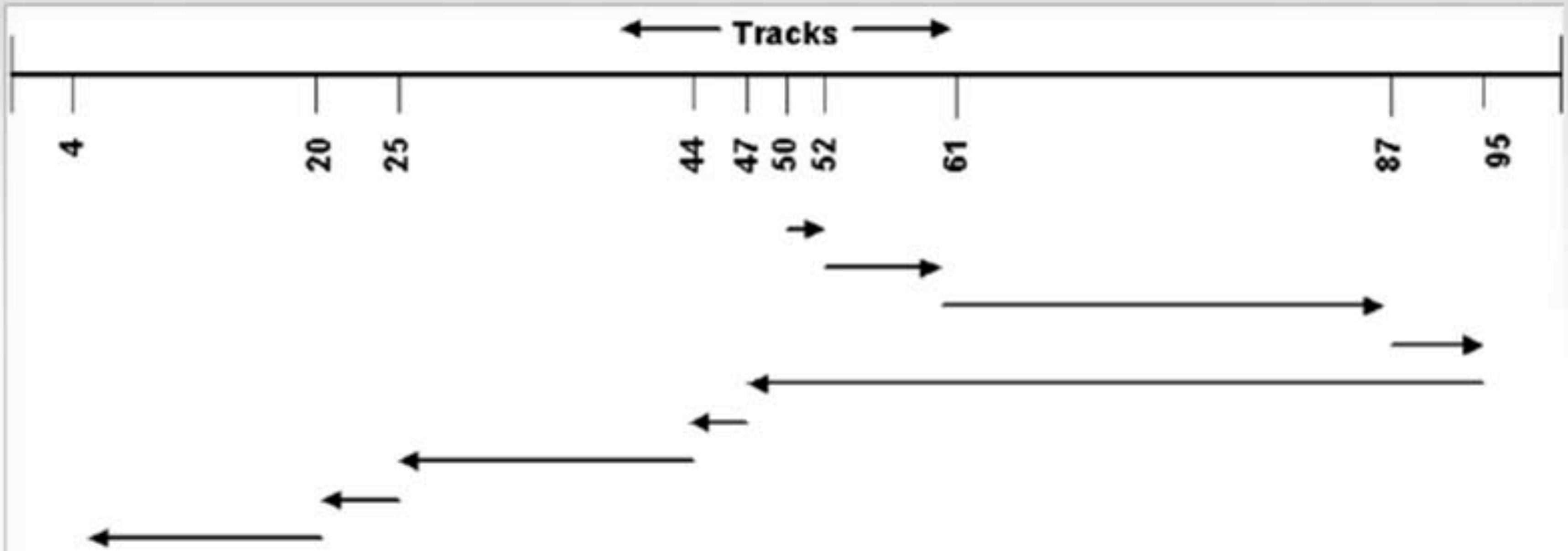
Disks: 2013

- Seagate Cheetah 3.5" (server)
  - capacity: 300 - 600 GB
  - rotational speed: 15,000 RPM
  - sequential read performance: 122 MB/s - 204 MB/s
  - seek time (average): 3.4 ms
- Seagate Barracuda 3.5" (desktop)
  - capacity: 250 GB – 4TB
  - rotational speed: 7,200 RPM
  - sequential read performance: 125 MB/s - 146 MB/s
  - seek time (average): 8.5 ms

Disk Scheduling

- Because seeks are so expensive (milliseconds!), the OS tries to schedule disk requests that are queued waiting for the disk
  - FCFS (do nothing)
    - Reasonable when load is low
    - Long waiting times for long request queues
  - SSTF (shortest seek time first)
    - Minimize arm movement (seek time), maximize request rate
    - Favors middle blocks
  - SCAN (elevator)
    - Service requests in one direction until done, then reverse
  - C-SCAN
    - Like SCAN, but only go in one direction (typewriter)

Example with SCAN Scheduling

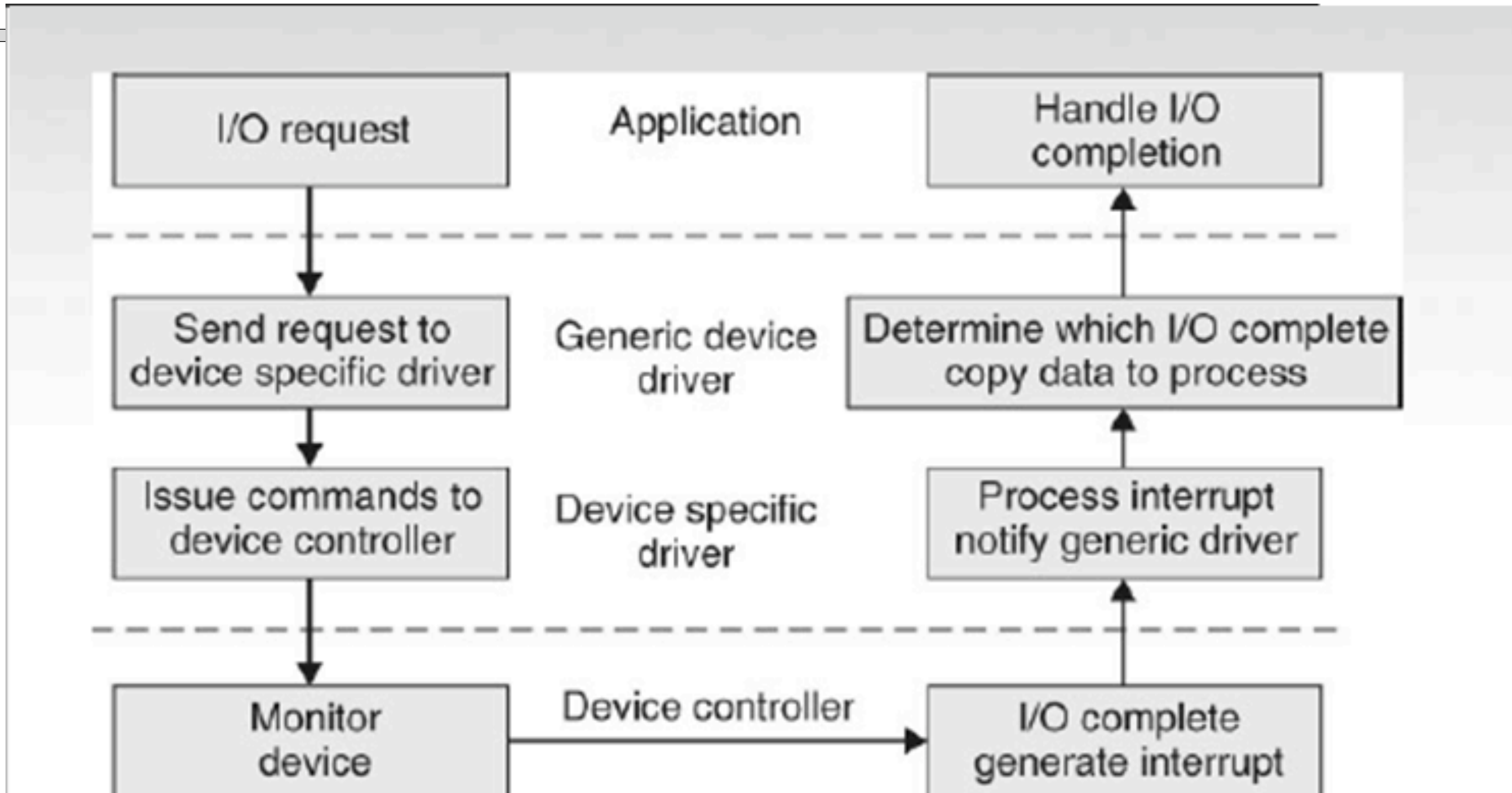


Total Movement = 136

Disk Scheduling (2)

- In general, unless there are request queues, disk scheduling does not have much impact
  - Important for servers, less so for PCs
- Modern disks often do the disk scheduling themselves
  - Disks know their layout better than OS, can optimize better
  - Ignores, undoes any scheduling done by OS

Stages of I/O Request



But do you directly program on “disk”?

Life with an OS

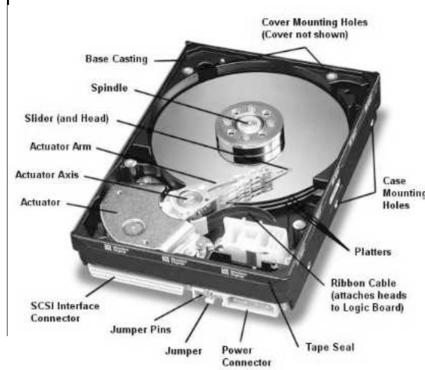
```
file = open (“test.txt”,
   O_WRONLY);

write (file, “test”, 4);

close (file);
```

Life without an OS

- Where is this file on disk? Which platter, track, and sectors?
- Code needs to change on a different system



File Systems

- File systems
  - Implement an abstraction (files) for secondary storage
  - Organize files logically (directories)
  - Permit sharing of data between processes, people, and machines
  - Protect data from unwanted access (security)

Files

- A file is data with some properties
  - Contents, size, owner, last read/write time, protection, etc.
- A file can also have a type
  - Understood by other parts of the OS or runtime libraries
    - Executable, dll, souce, object, text, etc.
  - Understood by the file system
    - Block/character device, directory, link, etc.
- A file's type can be encoded in its name or contents
  - Windows encodes type in name
    - .com, .exe, .bat, .dll, .jpg, etc.
  - Unix encodes type in contents
    - Magic numbers, initial characters (e.g., #! for shell scripts)

Basic File Operations

Unix

- creat(name)
- open(name, how)
- read(fd, buf, len)
- write(fd, buf, len)
- sync(fd)
- seek(fd, pos)
- close(fd)
- unlink(name)

Windows

- CreateFile(name, CREATE)
- CreateFile(name, OPEN)
- ReadFile(handle, …)
- WriteFile(handle, …)
- FlushFileBuffers(handle, …)
- SetFilePointer(handle, …)
- CloseHandle(handle, …)
- DeleteFile(name)
- CopyFile(name)
- MoveFile(name)

Directories

- Directories serve two purposes
  - For users, they provide a structured way to organize files
  - For the file system, they provide a convenient naming interface that allows the implementation to separate logical file organization from physical file placement on the disk
- Most file systems support multi-level directories
  - Naming hierarchies (/, /usr, /usr/local/, …)
- Most file systems support the notion of a current directory
  - Relative names specified with respect to current directory
  - Absolute names start from the root of directory tree

Directory Internals

- A directory is a list of entries
  - <name, location>
  - Name is just the name of the file or directory
  - Location depends upon how file is represented on disk
- List is usually unordered (effectively random)
  - Entries usually sorted by program that reads directory
- Directories typically stored in files

Basic Directory Operations

Unix

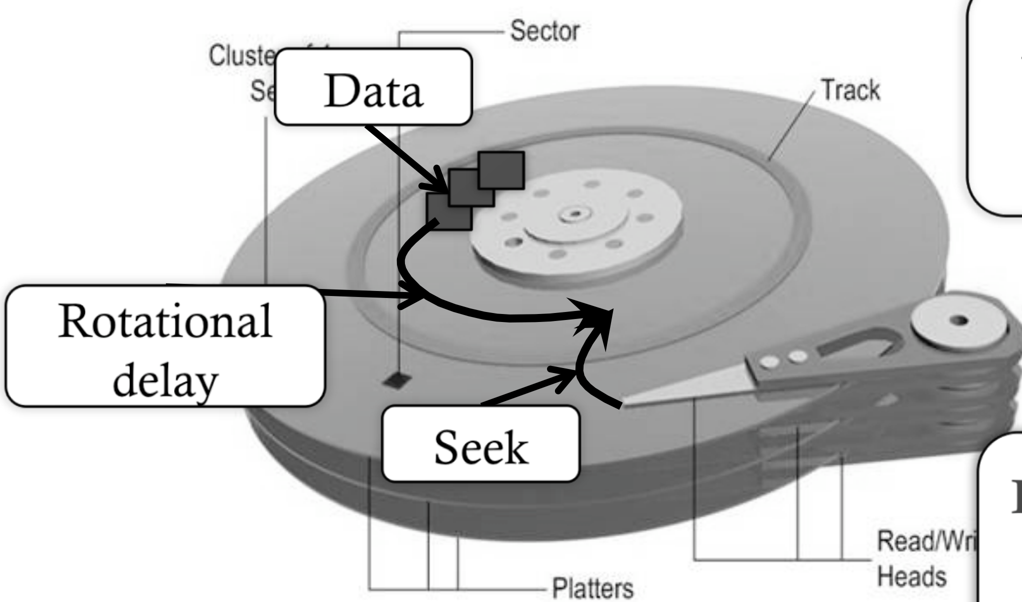
- Directories implemented in files
  - Use file ops to create dirs
- C runtime library provides a higher-level abstraction for reading directories
  - opendir(name)
  - readdir(DIR)
  - seekdir(DIR)
  - closedir(DIR)

NT

- Explicit dir operations
  - CreateDirectory(name)
  - RemoveDirectory(name)
- Very different method for reading directory entries
  - FindFirstFile(pattern)
  - FindNextFile()

Review

- Disk



Access data sequentially: only suffer one seek and rotational delay

Random disk access: suffers one seek and rotational delay every time!

Review: FS

- What is FS
  - Input to FS?
  - "Output" of FS?
- File
- Directory

Path Name Translation

- Let's say you want to open "/one/two/three"
- What does the file system do?
  - Open directory "/" (well known, can always find)
  - Search for the entry "one", get location of "one" (in dir entry)
  - Open directory "one", search for "two", get location of "two"
  - Open directory "two", search for "three", get location of "three"
  - Open file "three"
- Systems spend a lot of time walking directory paths
  - This is why open is separate from read/write
  - OS will cache prefix lookups for performance
    - /a/b, /a/bb, /a/bbb, etc., all share "/a" prefix

File System Layout

How do file systems use the disk to store files?

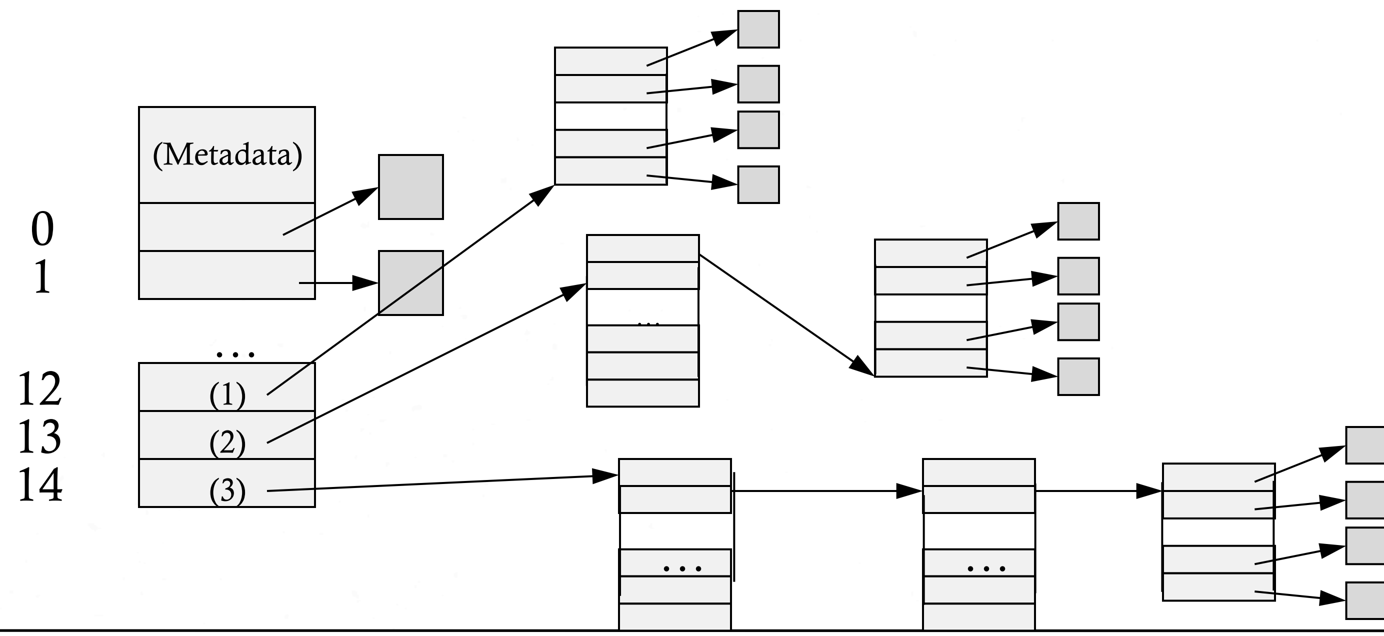
- File systems define a block size (e.g., 4KB)
  - Disk space is allocated in granularity of blocks
- A "Master Block" determines location of root directory
  - Always at a well-known disk location
  - Often replicated across disk for reliability
- A free map determines which blocks are free, allocated
  - Usually a bitmap, one bit per block on the disk
  - Also stored on disk, cached in memory for performance
- Remaining disk blocks used to store files (and dirs)
  - There are many ways to do this

Disk Layout Strategies

- Files span multiple disk blocks
- How do you find all of the blocks for a file?
  1. Contiguous allocation
     - Fast, simplifies directory access
     - Inflexible, causes fragmentation, needs compaction
  2. Linked structure
     - Each block points to the next, directory points to the first
     - Good for sequential access, bad for all others
  3. Indexed structure (indirection, hierarchy)
     - An "index block" contains pointers to many other blocks
     - Handles random better, still good for sequential
     - May need multiple index blocks (linked together)

Unix Inodes

- Unix inodes implement an indexed structure for files
  - Also store metadata info (protection, timestamps, length, ref count…)
- Each inode contains 15 block pointers
  - First 12 are direct blocks (e.g., 4 KB blocks)
  - Then single, double, and triple indirect



Unix Inodes and Path Search

- Unix Inodes are not directories
- Inodes describe where on the disk the blocks for a file are placed
  - Directories are files, so inodes also describe where the blocks for directories are placed on the disk
- Directory entries map file names to inodes
  - To open "/one", use Master Block to find inode for "/" on disk
  - Open "/", look for entry for "one"
  - This entry gives the disk block number for the inode for "one"
  - Read the inode for "one" into memory
  - The inode says where first data block is on disk
  - Read that block into memory to access the data in the file

Sharing Files btw. Directories

- Links (or hard links)
  - ln source_file target_dir
    - Simply create another link from target_dir to the **inode** of source_file (the inode is not duplicated)
    - Now two directories have links to source_file
    - What if we remove one?
    - Now you understand why the system call to remove a file is named “unlink”?
  - What if we duplicate the inode
    - Symbolic link

File Buffer Cache

- Applications exhibit significant locality for reading and writing files
- Idea: Cache file blocks in memory to capture locality
  - This is called the file buffer cache
  - Cache is system wide, used and shared by all processes
  - Reading from the cache makes a disk perform like memory
  - Even a 4 MB cache can be very effective
- Issues
  - The file buffer cache competes with VM (tradeoff here)
  - Like VM, it has limited size
  - Need replacement algorithms again (LRU usually used)

Caching Writes

- On a write, some applications assume that data makes it through the buffer cache and onto the disk
  - As a result, writes are often slow even with caching
- Several ways to compensate for this
  - “write-behind”
    - Maintain a queue of uncommitted blocks
    - Periodically flush the queue to disk
    - Unreliable
  - Battery backed-up RAM (NVRAM)
    - As with write-behind, but maintain queue in NVRAM
    - Expensive

Read Ahead (prefetch)

- Many file systems implement “read ahead”
  - FS predicts that the process will request next block
  - FS goes ahead and requests it from the disk
  - This can happen while the process is computing on previous block
    - Overlap I/O with execution
  - When the process requests block, it will be in cache
  - Compliments the disk cache, which also is doing read ahead
- For sequentially accessed files can be a big win
  - Unless blocks for the file are scattered across the disk
  - File systems try to prevent that, though (during allocation)

Performance Issues

Original Unix FS had two placement problems:

1. Data blocks allocated randomly in aging file systems
   - Blocks for the same file allocated sequentially when FS is new
   - As FS "ages" and fills, need to allocate into blocks freed up when other files are deleted
   - Problem: Deleted files essentially randomly placed
   - So, blocks for new files become scattered across the disk
2. Inodes allocated far from blocks
   - All inodes at beginning of disk, far from data
   - Traversing file name paths, manipulating files, directories requires going back and forth from inodes to data blocks

Both of these problems generate many long seeks

Fast File System

- BSD FFS addressed these problems using the notion of a cylinder group
  - Disk partitioned into groups of cylinders
  - Data blocks in same file allocated in same cylinder
  - Files in same directory allocated in same cylinder
  - Inodes for files allocated in same cylinder as file data blocks
- Free space requirement
  - To be able to allocate according to cylinder groups, the disk must have free space scattered across cylinders
  - 10% of the disk is reserved just for this purpose

Summary

- Files
  - Operations, access methods
- Directories
  - Operations, using directories to do path searches
- Sharing
  - Link
- File System Layouts
  - Unix inodes
- File Buffer Cache
  - Strategies for handling writes
- Read Ahead