

CSE 120

Principles of Operating Systems

Fall 2019

Lecture 12: File Systems

Geoffrey M. Voelker

File Systems

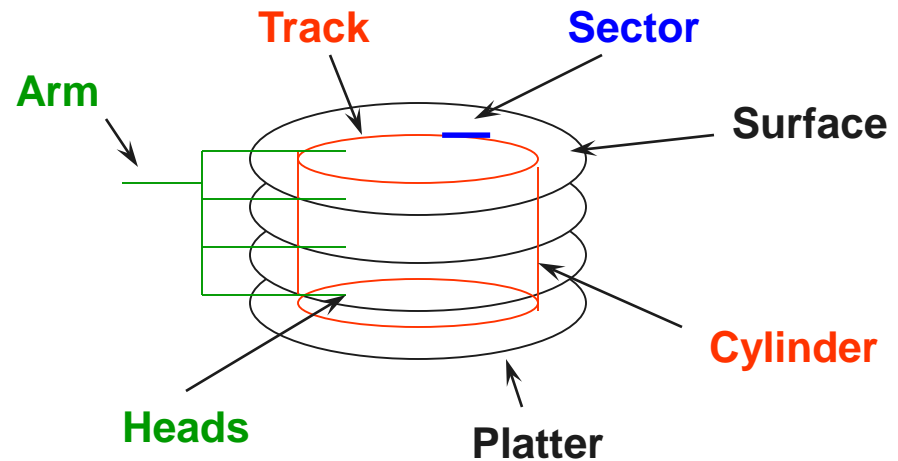
- First we'll discuss properties of physical disks
 - ◆ Structure
 - ◆ Performance
- Then how file systems support users and programs
 - ◆ Files
 - ◆ Directories
 - ◆ Sharing
 - ◆ Protection
- End with how file systems are implemented
 - ◆ File System Data Structures
 - ◆ File Buffer Cache
 - ◆ Read 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.)
- The OS may provide different levels of disk access to different clients
 - ◆ Physical disk (surface, cylinder, sector)
 - ◆ Logical disk (disk block #)
 - ◆ Logical file (file block, record, or byte #)

Physical Disk Structure

- Disk components
 - ◆ Platters
 - ◆ Surfaces
 - ◆ Tracks
 - ◆ Sectors
 - ◆ Cylinders
 - ◆ Arm
 - ◆ Heads



Disk Interaction

- Specifying disk requests requires a lot of info:
 - ◆ Cylinder #, surface #, track #, 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
 - ◆ The disk exports its data as a logical array of blocks [0...N]
 - » Disk maps logical blocks to cylinder/surface/track/sector
 - » Block size can be configured via low-level formatting
 - ◆ Only need to specify the logical block # to read/write
 - ◆ But now the disk parameters are hidden from the OS

Disk Specifications

- Seagate Enterprise Performance 3.5" ([server](#))
 - ◆ capacity: 600 GB
 - ◆ rotational speed: 15,000 RPM
 - ◆ sequential read performance: 233 MB/s (outer) – 160 MB/s (inner)
 - ◆ seek time (average): 2.0 ms
- Seagate Barracuda 3.5" ([workstation](#))
 - ◆ capacity: 3000 GB
 - ◆ rotational speed: 7,200 RPM
 - ◆ sequential read performance: 210 MB/s - 156 MB/s (inner)
 - ◆ seek time (average): 8.5 ms
- Seagate Savvio 2.5" ([smaller form factor](#))
 - ◆ capacity: 2000 GB
 - ◆ rotational speed: 7,200 RPM
 - ◆ sequential read performance: 135 MB/s (outer) - ? MB/s (inner)
 - ◆ seek time (average): 11 ms

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

Solid State Disks

- SSDs are a relatively new storage technology
 - ◆ Memory that does not require power to remember state
- No physical moving parts → faster than hard disks
 - ◆ No seek and no rotation overhead
 - ◆ But...more expensive, not as much capacity
- Generally speaking, file systems have remained unchanged when using SSDs
 - ◆ Some optimizations no longer necessary (e.g., layout policies, disk head scheduling), but basically leave FS code as is
 - ◆ Initially, SSDs have the same disk interface (SATA)
 - ◆ Increasingly, SSDs used directly over the I/O bus (PCIe M.2/3)
 - » Much higher performance

Non-Volatile Memory (NVM)

- Under development are new technologies that provide non-volatile memory
 - ◆ Phase change (PCM), spin-torque transfer (STTM), resistive
- Performance close to DRAM
 - ◆ But persistent!
- Byte-addressable
 - ◆ SSD is in units of a page
- Unlike SSDs, NVM will have a dramatic effect on both operating systems and applications
 - ◆ See Steve Swanson's and Jishen Zhao's research!
- Intel/Micron Optane first commercially available product

Disk Scheduling

- Because seeks are so slow (milliseconds!), the OS tries to schedule disk requests that are queued waiting for the disk
 - ◆ Many algorithms for doing so
- In general, unless there are many requests in 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

File Systems

- File systems
 - ◆ Implement an abstraction (**files**) for secondary storage
 - ◆ Organize files logically (**directories**)
 - ◆ Permit sharing of data among processes, people, and machines
 - ◆ Protect data from unwanted access (**protection**)
- OSes abstract file systems behind an interface
 - ◆ Unix: **virtual file system (VFS)**
 - ◆ Windows: **installable file system (IFS)**
- Interface defines set of methods and data types
 - ◆ OS implemented to use any file system through this interface
 - ◆ Linux (**ext3, ext4, xfs, btrfs**), Windows (**FAT16, FAT32, NTFS**)
 - ◆ Another example of level-of-indirection in systems design

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 the file system
 - » Block, character, device, portal, link, etc.
 - ♦ Understood by other parts of the OS or runtime libraries
 - » Executable, dll, source, object, text, 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)`
- `rename(old,new)`

Windows

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

File Access Methods

- Some file systems provide different **access methods** that specify different ways for accessing data in a file
 - ♦ **Sequential access** – read bytes one at a time, in order
 - ♦ **Direct access** – random access given block/byte number
 - ♦ **Record access** – file is array of fixed- or variable-length records, read/written sequentially or randomly by record #
 - ♦ **Indexed access** – file system contains an index to a particular field of each record in a file, reads specify a value for that field and the system finds the record via the index (DBs)
- **What file access method does Unix, Windows provide?**
- Older systems provide the more complicated methods

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 OSes support the notion of a current directory
 - ◆ Relative names specified with respect to current directory
 - ◆ Absolute names start from the root of directory tree
 - ◆ Maintained on a per-process basis

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
 - ◆ Try “ls -U /bin”
- Directories typically stored in files
 - ◆ Only need to manage one kind of secondary storage unit

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)`

Windows

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

Path Name Translation (v1)

- 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
 - ◆ 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 Sharing

- File sharing has been around since timesharing
 - ♦ Easy to do on a single machine
 - ♦ Networks give us remote sharing (mostly)
- File sharing is important for getting work done
 - ♦ Basis for communication and synchronization
- Two key issues when sharing files
 - ♦ Semantics of concurrent access
 - » What happens when one process reads while another writes?
 - » What happens when two processes open a file for writing?
 - » **What are we going to use to coordinate?**
 - ♦ Protection

Protection

- File systems implement a protection system
 - ◆ Who can access a file
 - ◆ How they can access it
- More generally...
 - ◆ Objects are “what”, subjects are “who”, actions are “how”
- A protection system dictates whether a given **action** performed by a given **subject** on a given **object** should be allowed
 - ◆ You can read and/or write your files, but others cannot
 - ◆ You can read “/etc/motd”, but you cannot write it

Representing Protection

Access Control Lists (ACL)

- For each object, maintain a list of subjects and their permitted actions

Capabilities

- For each subject, maintain a list of objects and their permitted actions

	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

Subjects

Objects

ACL

Capability List

Root & Administrator

- The user “root” is special on Unix
 - ♦ It bypasses all protection checks in the kernel
- Administrator is the equivalent on Windows
- **Always running as root can be dangerous**
 - ♦ A mistake (or exploit) can harm the system
 - » “rm” will always remove a file
 - ♦ Create a user account on Unix even if you have root access
 - » You only run as root when you need to modify the system
 - ♦ If you have Administrator privileges on Windows, then you are effectively always running as root (for the most part)
 - » Need additional protection mechanisms (**User Account Control**)

ACLs and Capabilities

- Approaches differ only in how the table is represented
 - ♦ What approach does Unix use in the FS?
- Capabilities are easier to transfer
 - ♦ They are like keys, can handoff, does not depend on subject
- In practice, ACLs are easier to manage
 - ♦ Object-centric, easy to grant, revoke
 - ♦ To revoke capabilities, have to keep track of all subjects that have the capability – a challenging problem
- ACLs have a problem when objects are heavily shared
 - ♦ The ACLs become very large
 - ♦ Use groups (e.g., Unix)
- We will revisit protection in more depth again

Next time...

- Chapters 37, 39, 40