# CSE 153
# Design of Operating Systems

## Winter 2023

Lecture 19: File Systems (2)—Abstractions and implementation

# Plan for today

- Abstractions for the disk drive that:
  - Store information persistently
  - Allow users to organize information
  - Provide tools for controlling access

- How to implement the abstractions
  - We saw the structure of disk drives
    - Sea of blocks
    - Seeks are costly
    - How to support abstractions?

# 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 a sequence of bytes with some properties
  - 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, souce, 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)

NT

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

# File Access Methods

- Different file systems differ in the manner that data in a file can be accessed
  - 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)
- Older systems provide more complicated methods
- What file access method do Unix, Windows provide?

# 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
  - 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

- 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 Sharing

- File sharing is important for getting work done
  - Basis for communication between processes and users

- 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?
  - Protection

# Protection

- File systems implement some kind of 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 to it

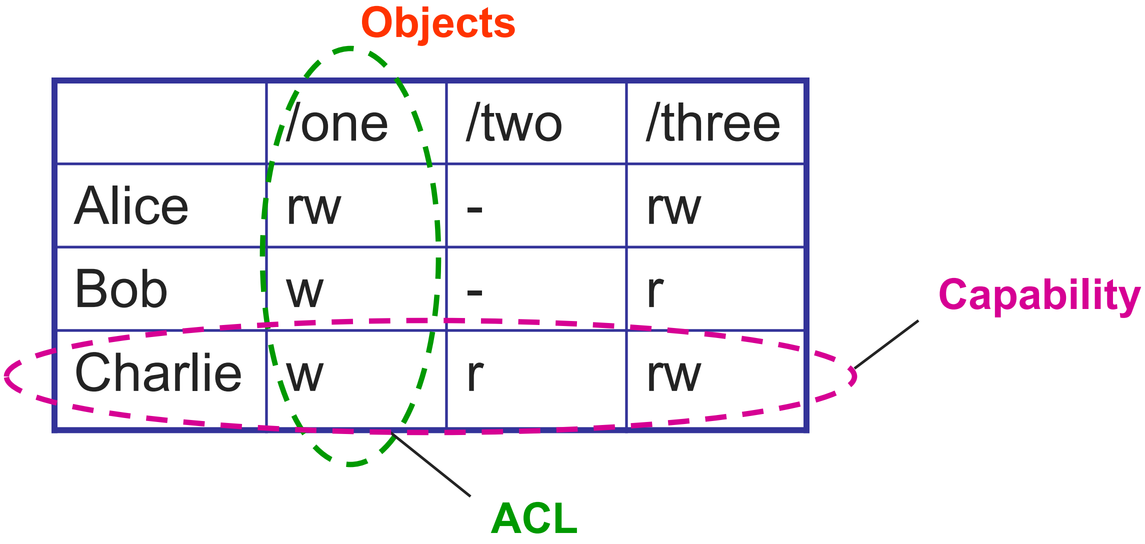
# Representing Protection

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 |

# ACLs and Capabilities

- The approaches differ only in how table is represented
  - What approach does Unix use?
- 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)

# 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
  - At fixed disk location, sometimes replicated 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 blocks store files (and dirs), and swap!

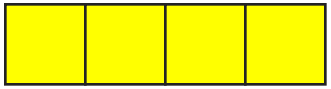
# File systems

- File system design: how to allocate and keep track of files and directories
- Does it matter?  What is the difference?
  - Performance, reliability, limitations on files, overhead, …
- Many different file systems have been proposed and continue to be proposed
- Lets talk about some general ideas first

# Disk Layout Strategies

- Files span multiple disk blocks
- How do you find all of the blocks for a file?

1. Contiguous allocation

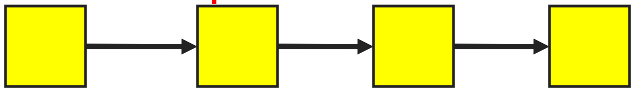


   » Like memory

   » Fast, simplifies directory access

   » Inflexible, causes fragmentation, needs compaction

2. Linked structure



   » Each block points to the next, directory points to the first

   » Bad for random access patterns

3. Indexed structure (indirection, hierarchy)

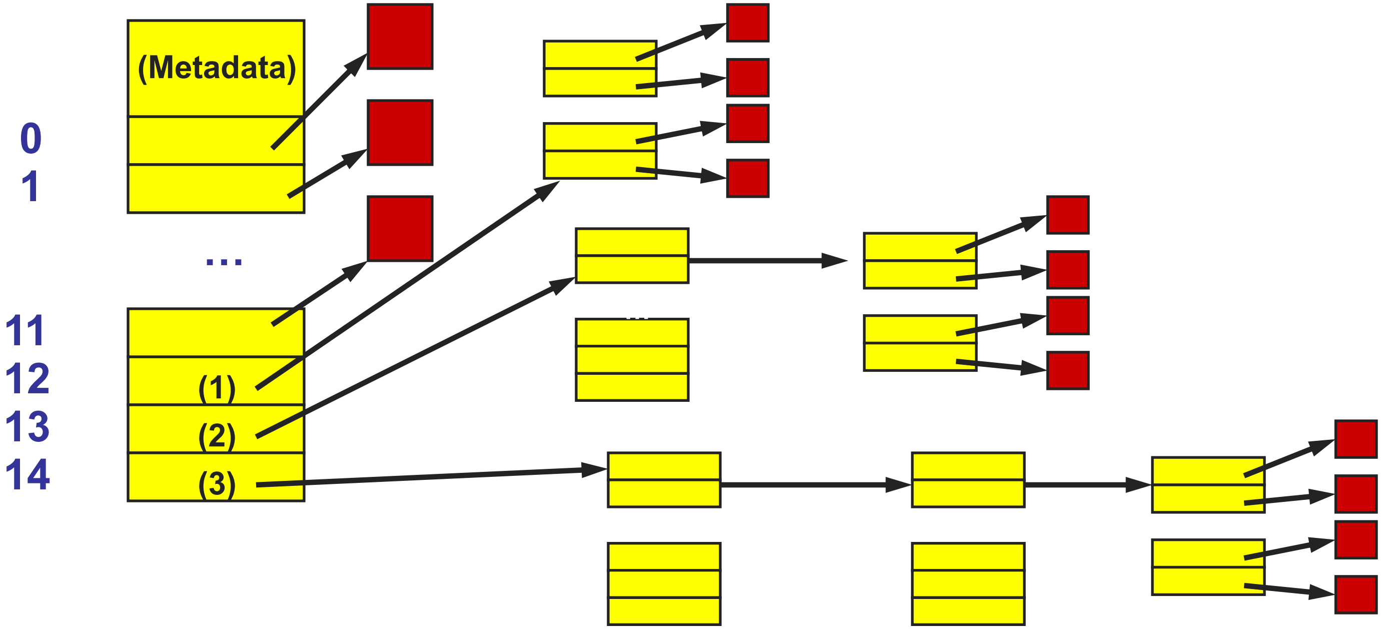
   » An “index block” contains pointers to many other blocks

   » 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 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
- This is why we have *open* in addition to *read* and *write*