Files and File Systems

Major OS Themes

Virtualization

- Present physical resource as a more general, powerful, or easy-to-use form of itself
- Present illusion of multiple (or unlimited) resources where only one (or a few) really exist
- Examples: CPU, Memory, Hard drives

Concurrency

- Coordinate multiple activities to ensure correctness

Persistence

- Some data needs to survive crashes and power failures

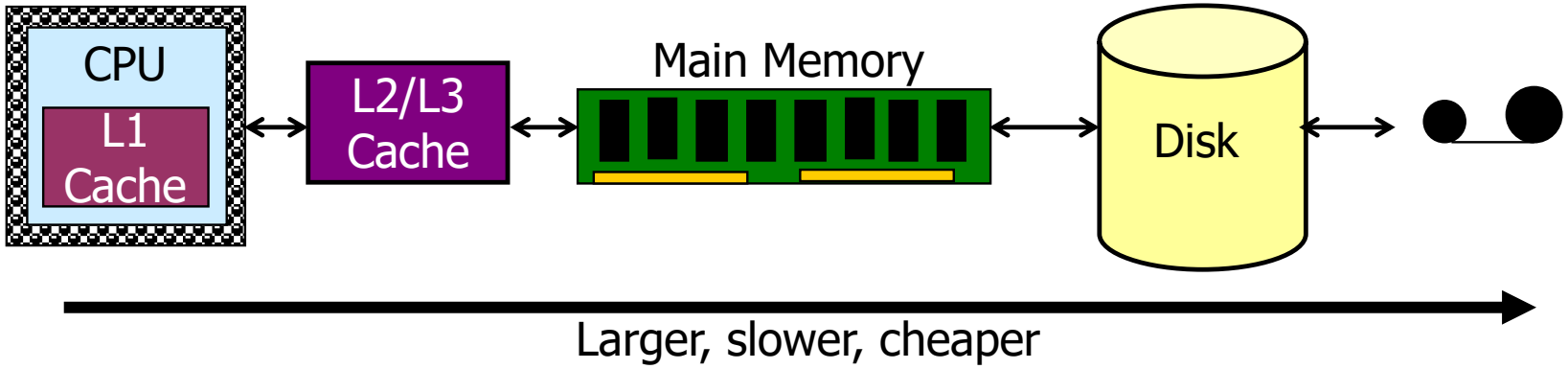
- Need abstractions, mechanisms, policies for all

How to virtualize persistent storage?

- Recall OS Goals:
  - Convenience for the user and efficient use of machine
- Virtualization and abstraction helps with first goal
  - Files and directories abstract away the hard drive
- Efficiency:
  - File system controls when and how data is transferred to persistent storage

Storage Hierarchy

- processor registers, main memory, and auxiliary memory form a rudimentary memory hierarchy
- the hierarchy can be classified according to memory speed, cost, and volatility
- caches can be installed to hide performance differences when there is a large access-time gap between two levels



Outline

- Persistent storage of data
- Files
- Directories
- File Operations
- Inodes
- Traversing Paths

File Systems

- Provide long-term information storage
- Requirements:
  - Store very large amounts of information
  - Information must survive the termination of process using it
  - Multiple processes must be able to access info concurrently
- Two views of file systems:
  - User view – convenient logical organization of information
  - OS view – managing physical storage media, enforcing access restrictions

File (Management) 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)

File Operations

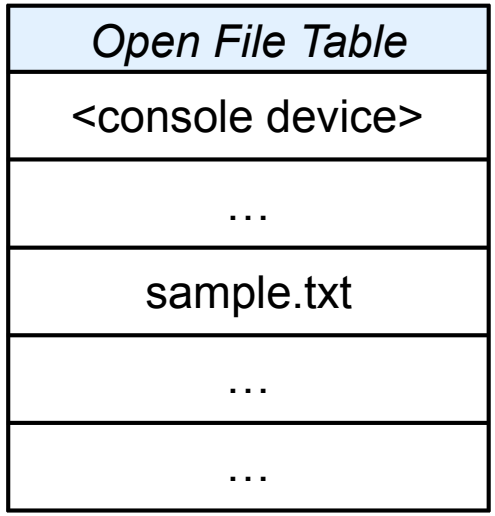
- Creation
  - Find space in file system, add entry to directory
    - map file name to location and attributes
- Writing
- Reading
  - Dominant abstraction is “file as stream”
- Repositioning within a file
- Deleting a file
- Truncation and appending
  - May erase the contents (or part of the contents) of a file while keeping attributes

Handling File Operations

- Must search the directory for the entry associated with the named file
- When the file is first used, store its attribute info in a system-wide open-file table
  - The index into the open-file table is used on subsequent operations, so no searching is required

```
Unix example (open, read, write are syscalls):

main() {
    char onebyte;
    int fd = open("sample.txt", "r");
    read(fd, &onebyte, 1);
    write(STDOUT, &onebyte, 1);
    close(fd);
}
```



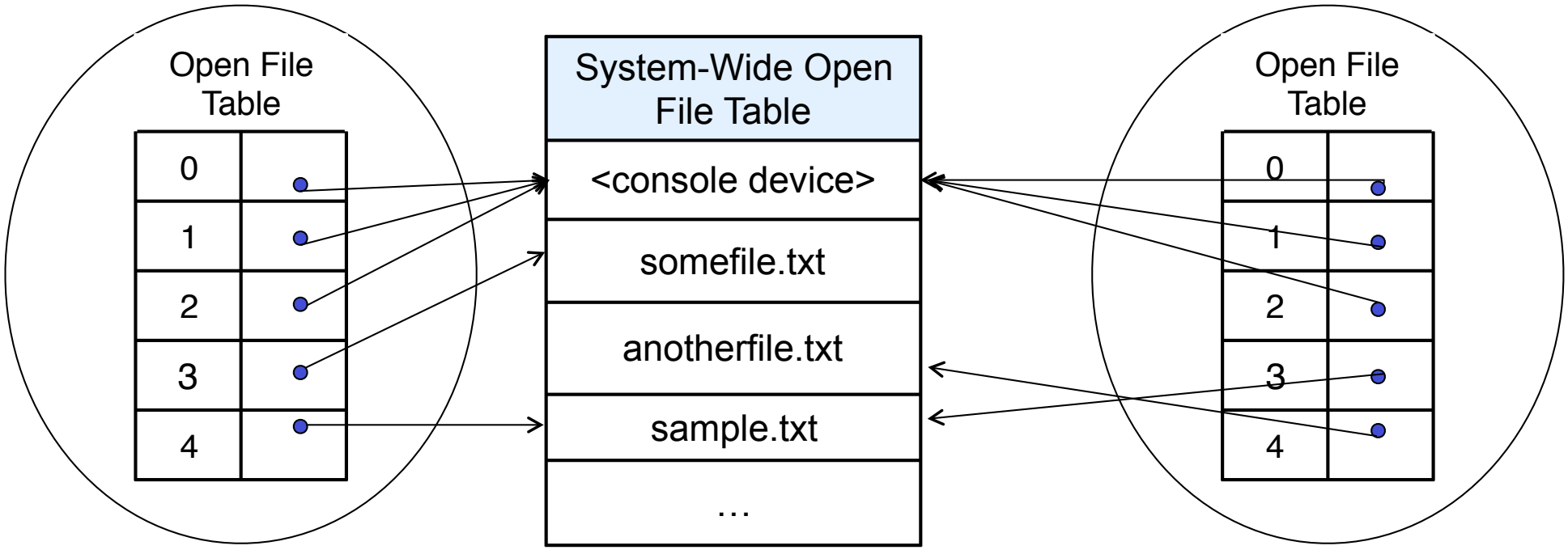
File Sharing

- File sharing is incredibly 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?
  - Protection

Shared Open Files

- There are two levels of internal tables

  1. A per-process table of all files that each process has open that contains the current file position for that process
  2. Each entry in the per-process table has a pointer to an entry in the system-wide open-file table for process independent info



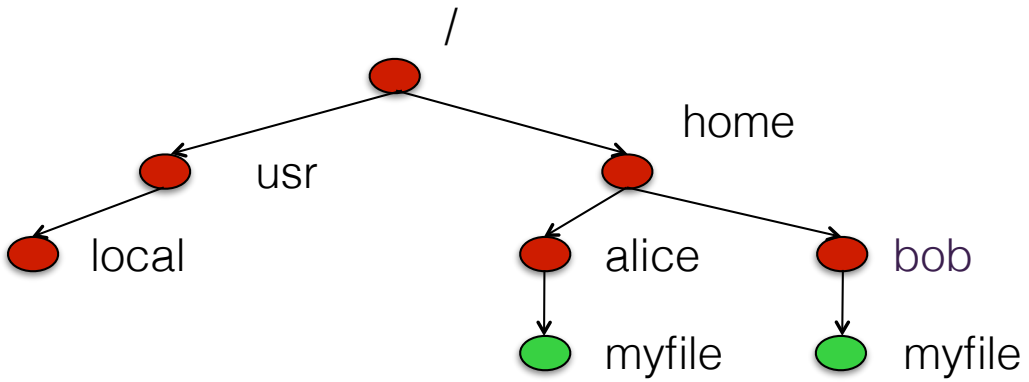
File Access Methods

- General-purpose file systems support simple methods
  - Sequential access – read bytes one at a time, in order
  - Direct access – random access given block/byte number

- Database systems support more sophisticated methods
  - Record access - fixed or variable length
  - Indexed access
- What file access method(s) does Unix/Linux, Windows provide?
- Older systems provide more complicated methods
  - Modern systems typically only support simple access

Directories

- Directories provide logical structure to file systems
  - For users, they provide a means to organize files
  - For the file system, they provide a convenient naming interface
    - Allows the implementation to separate logical file organization from physical file placement
    - Stores information about files (owner, permission, etc.)
- Most file systems support multi-level directories
  - Naming hierarchies (/, /usr, /usr/local/, /home, …)



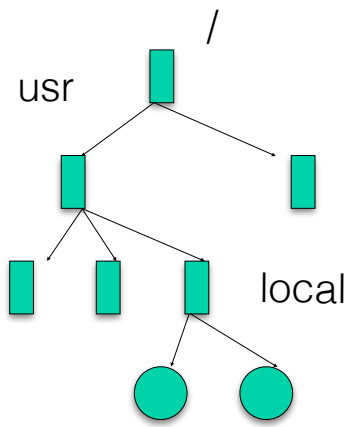
Directory Structure

- A directory is a list of entries – names and associated metadata
  - Metadata is not the data itself, but information that describes properties of the data (size, protection, location, etc.)
- 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

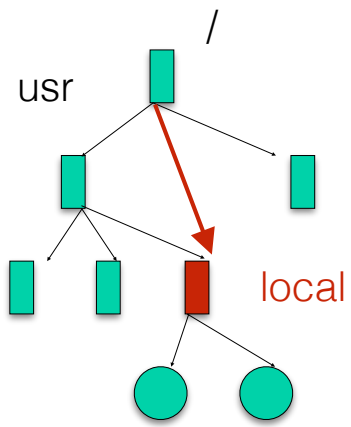
Directory Implementations

- Single-level, two-level, or tree-structured
- Acyclic-graph directories: allows for shared directories
  - The same file or subdirectory may be in 2 different directories

Tree-structured:



Acyclic graph:



Directory Implementation

- Option 1: List
  - Simple list of file names and pointers to data blocks
  - Requires linear search to find entries
  - Easy to implement, slow to execute
    - And directory operations are frequent!
- Option 2: Hash Table
  - Create a list of file info structures
  - Hash file name to get a pointer to the file info structure in the list
  - Hash table takes space

File Links

Sharing can be implemented by creating a new directory entry called a link: a pointer to another file or subdirectory

- Symbolic, or soft, link
  - Directory entry refers to file that holds "true" path to the linked file
- Hard links
  - Second directory entry identical to the first

'/' directory

File Name	Start Block	Type
…	…	…
local	42	dir
usr	150	dir

'usr' directory (hard link)

File Name	Start Block	Type
…	…	…
local	42	dir
…	…	…

'usr' directory (soft link)

File Name	Start Block	Type
…	…	…
local	215	link
…	…	…

Issues with Links

- With links, a file may have multiple absolute path names
  - Traversing a file system should avoid traversing shared structures more than once
- Maintaining consistency is a problem
  - How do you update permissions in directory entry with a hard link?
- Deletion: When can the space allocated to a shared file be deallocated and reused?
  - Somewhat easier to handle with symbolic links
    - Deletion of a link is OK; deletion of the file entry itself deallocates space and leaves the link pointers dangling
  - Keep a reference count for hard links
- Sharing: How can you tell when two processes are sharing the same file?

Next Up

- Implementing File Systems
  - Disk layout
  - File metadata
  - Directory data

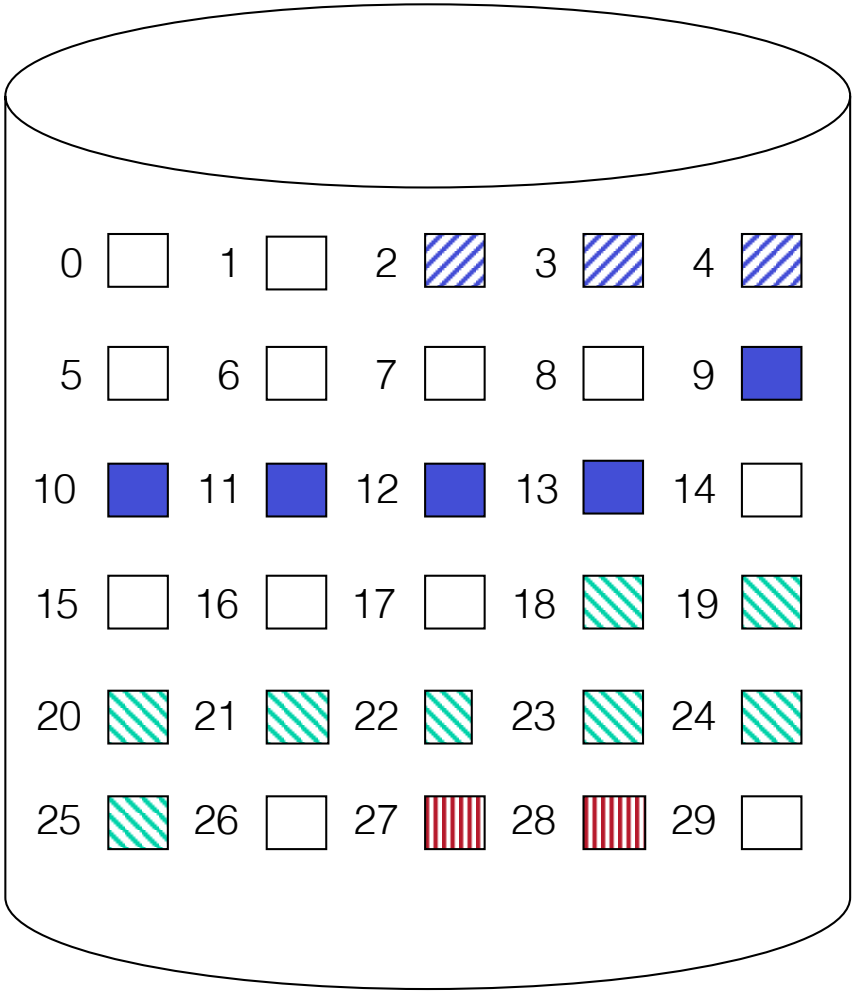
File System Implementation

- 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 (aka *partition control block, superblock)*
  - Always at a well-known disk location
  - Often replicated across disk for reliability
- A free map determines which blocks are free
  - 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

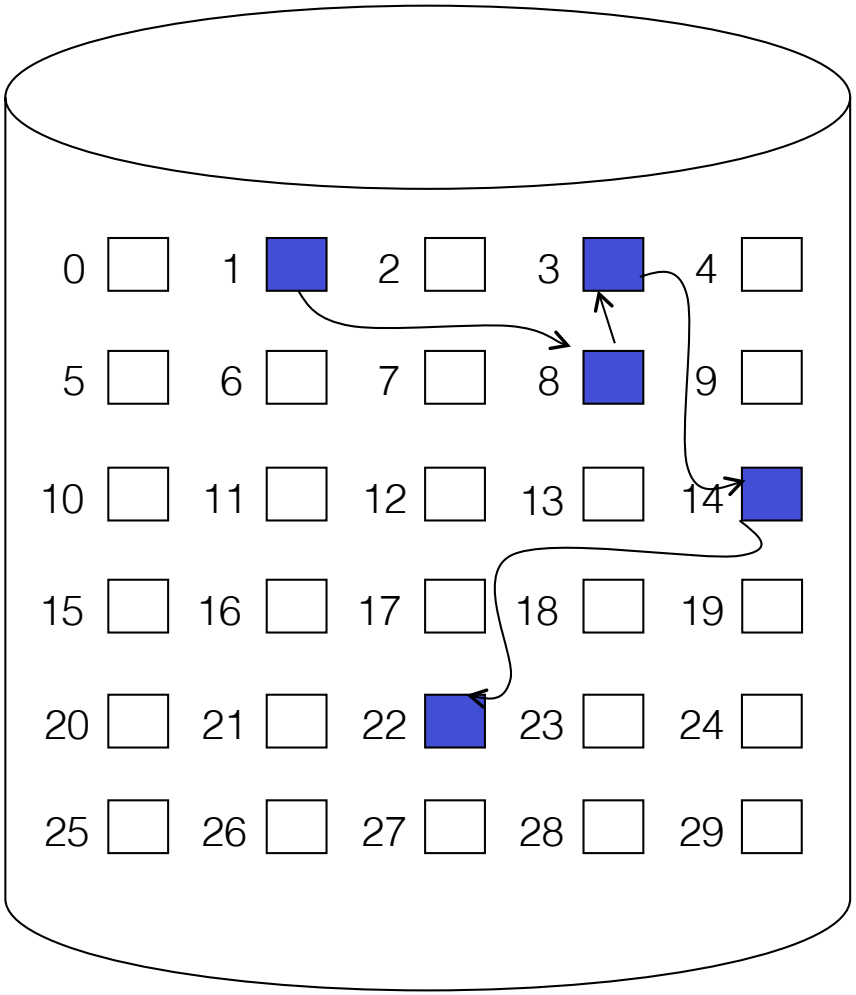
- Files often span multiple disk blocks (think “processes often span multiple pages ...”)
- How do you find all of the blocks for a file?
  1. Contiguous allocation
     - Fast, simplifies directory access and allows indexing
     - Inflexible, causes external fragmentation, requires compaction
  2. Linked, or chained, structure
     - Each block points to the next, directory points to the first
     - Good for sequential (streaming) access, bad for all others
  3. Indexed structure (kind of like address translation)
     - An “index block” contains pointers to many other blocks
     - Handles random access better, still good for sequential
     - May require multiple, linked index blocks

Contiguous Allocation



File Name	Start Blk	Length
File A	2	3
File B	9	5
File C	18	8
File D	27	2

Linked Allocation



File Name	Start Blk	Last Blk
…	…	…
File B	1	22
…	…	…

Unix Inodes

- Unix inodes implement an indexed structure for files
- All file metadata is stored in an inode
  - Unix directory entries map file names to inodes
- Each inode contains 15 block pointers
  - First 12 are direct block pointers
    - Disk addresses of first 12 data blocks in file
  - The 13th is a single indirect block pointer
    - Address of block containing addresses of data blocks
  - Then the 14th is a double indirect block pointer
    - Address of block containing addresses of single indirect blocks
  - Finally, the 15th is a triple indirect block pointer

Example UNIX Inode

- Ext2 Linux file system Inodes are 72 bytes