ILLINOIS INSTITUTE OF TECHNOLOGY

File Systems Basics

Anthony Kougkas

akougkas@iit.edu

Outline

- File System Overview
- File System Architecture
- File System Semantics
- File System Operations

File System Overview

- System that permanently stores data
- Usually layered on top of a lower-level physical storage medium
- Divided into logical units called “files”
  - Addressable by a *filename* (“foo.txt”)
  - Usually supports hierarchical nesting (directories)
- A file *path* joins file & directory names into a **relative** or **absolute** address to identify a file (“/home/aaron/foo.txt”)

File (an abstraction)

- A (potentially) large amount of information or data that lives a (potentially) very long time
  - Often *much* larger than the memory of the computer
  - Often *much* longer than any computation
  - Sometimes longer than life of machine itself
- (Usually) organized as a linear array of bytes or blocks
  - Internal structure is imposed by application
  - (Occasionally) blocks may be variable length
- (Often) requiring concurrent access by multiple processes
  - Even by processes on different machines!

Directory – A Special Kind of File

- A tool for users & applications to organize and find files
  - User-friendly names
  - Names that are meaningful over long periods of time

- The data structure for OS to locate files (i.e., containers) on disk

File System Basics

- *File*:
  - Named collection of logically related data
  - Unix file: an uninterpreted sequence of bytes

- *File system:*
  - Provides a logical view of data and storage functions
  - User-friendly interface
  - Provides facility to create, modify, organize, and delete files
  - Provides sharing among users in a controlled manner
  - Provides protection

File Types and Attributes

- *File types:*
  - Regular files
  - Directories
  - Character special files: used for serial I/O
  - Block special files: used to model disks [buffered I/O]
- *File attributes*: varies from OS to OS
  - Name, type, location, size, protection info, password, owner, creator, time and date of creation, last modification, access
- *File operations*:
  - Create, delete, open, close, read, write, append, get/set attributes
- *File access:*
  - Sequential, random

Attributes of Files

- *Name:*
  - Although the name is not always what you think it is!
- *Type:*
  - May be encoded in the name (e.g., *.cpp*, *.txt*)
- *Dates:*
  - Creation, updated, last accessed, etc.
  - (Usually) associated with container
  - Better if associated with content
- *Size:*
  - Length in number of bytes; occasionally rounded up
- *Protection:*
  - Owner, group, etc.
  - Authority to read, update, extend, etc.
- *Locks:*
  - For managing concurrent access
- …

Definition — *File Metadata*

- Information *about* a file
  - Maintained by the file system
  - Separate from file itself
  - Usually attached or connected to the file
    - E.g., in block # –1
  - Some information visible to user/application
    - Dates, permissions, type, name, etc.
  - Some information primarily for OS
    - Location on disk, locks, cached attributes

Observation – some attributes are not visible to user or program

- E.g., *location*
  - Location is stored in metadata
  - Location can change, even if file does not
  - Location is not visible to user or program

File System

- Components: directory, authorization, file service and system service
  - **Authorization service**: between file and directory services
  - **Directory service**: used to keep track of the location of all resources in the system
  - **File service** provides a transparent way of accessing any file in the system in the same way
  - **System service**: file system's interface to hardware

File Systems

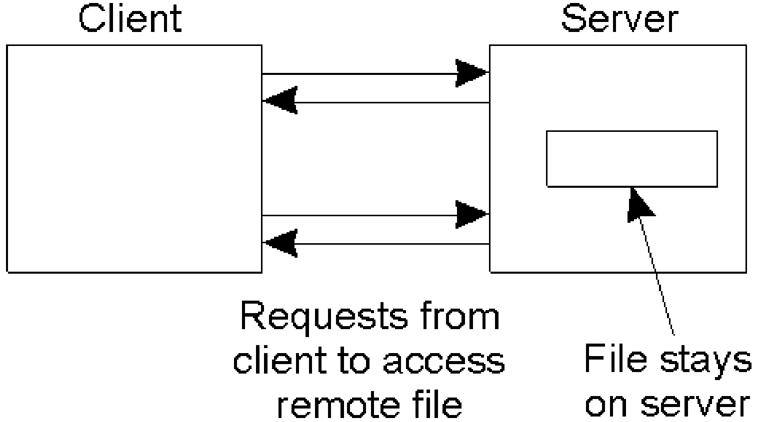
- ***File service:***
  - Specification of what the file system offers to client
    - Actions
    - Client primitives
    - Parameters, application programming interface (API)
  - Does not include how service is implemented

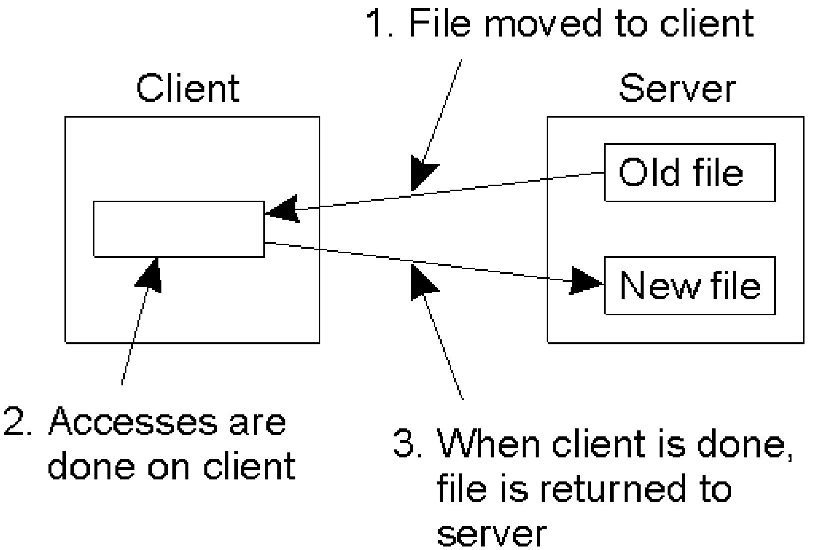
- ***File server:***
  - Process that runs on a machine and implements file service
  - Can have several servers on one machine (UNIX, DOS,…)
  - ideally, clients do not know the distributed nature

Architectures

- How are FS generally organized?
  - Client-server architectures
    - Example: Sun Microsystem's NFS
    - File servers with a standardized view of its local file system; clients can access these files
  - Cluster-based distributed file systems
    - Example: file striping, partitioning the whole file system, GFS
  - Symmetric architectures
    - Fully symmetric organization based on p2p
    - Example: Ivy

Client-Server Architectures

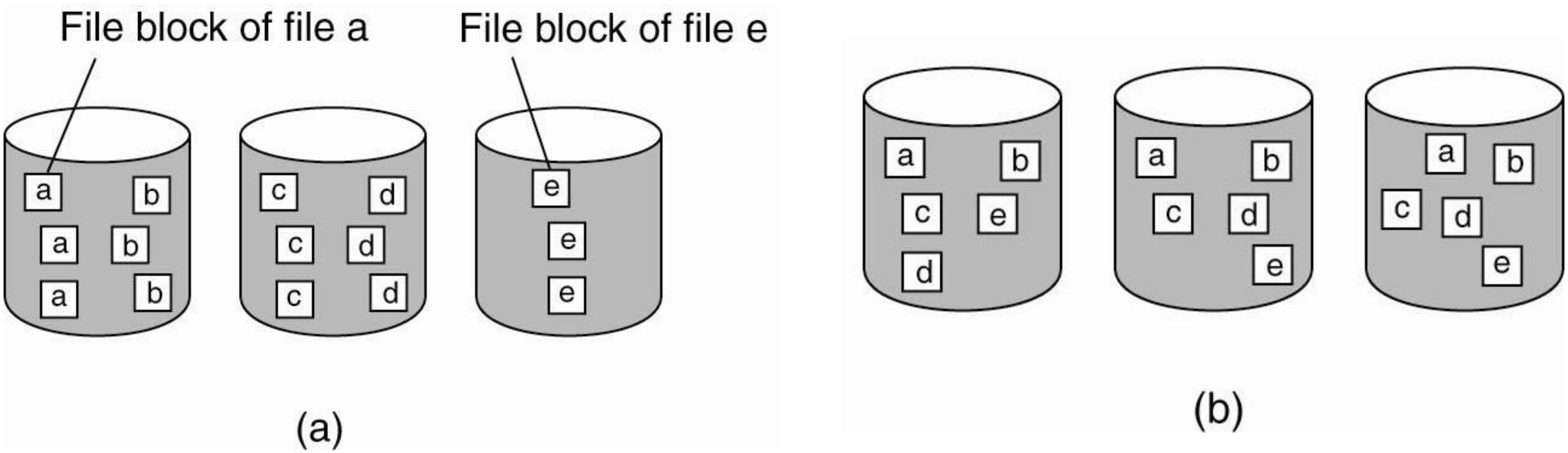




- **Remote access model**
  - Work done at the server
- Stateful server (e.g., databases)
- Pros & cons?

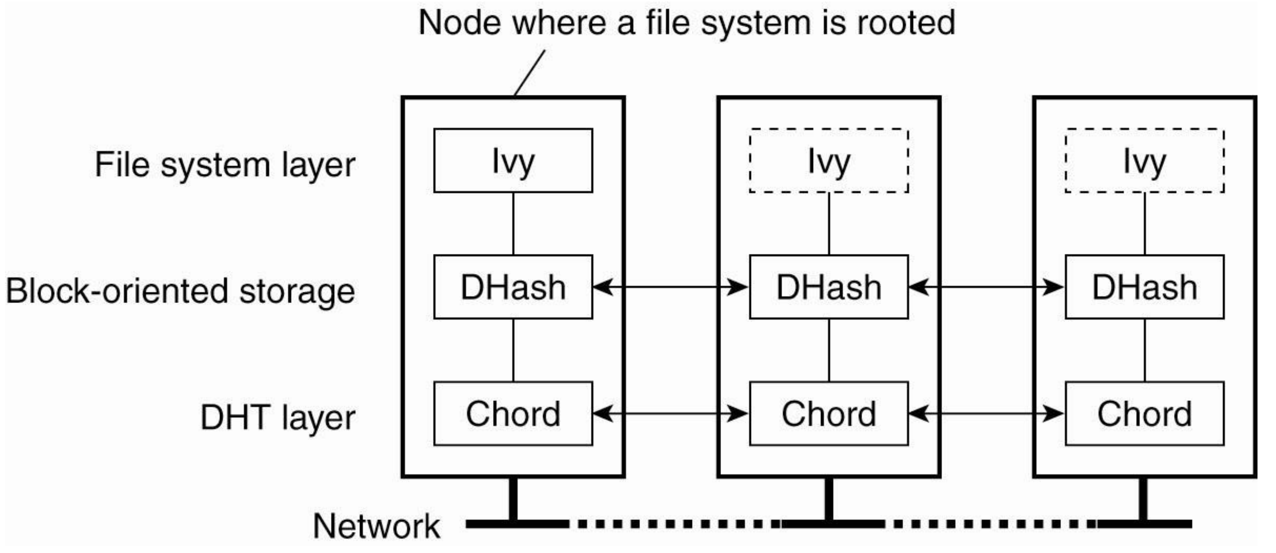
- **Upload/download model**
  - Work done at the client
- Stateless server
- Pros & cons?

Cluster-based DFSs



- Figure (b): When server clusters are used for parallel applications
  - File-striping techniques, a single file is distributed across multiple servers
- Figure (a): For general-purpose applications, when file striping may not be effective
  - Partition the file system as a whole and simply store different files on different servers

Symmetric Architectures



- Commonly used DHT-based systems for distributing data, combined with a key-based lookup mechanism
- Key difference: whether build a file system on top of a distributed storage layer
- Example: Ivy

System Structure

- How should the system be organized?
  - Are clients and server different?
    - Same process implements both functionality
    - Different processes, same machine
    - Different machines (a machine can either be client or server)
  - How are file and directory services organized-same server?
    - Different server processes: cleaner, more flexible, more overhead
    - Same server: just the opposite
  - Caching/no caching
    - server
    - client
  - How are updates handled?
  - File sharing semantics?
  - Server type: stateful vs. stateless

Semantics of File Sharing

- **Unix semantics**: used in centralized systems
  - Read after write returns value written
    - System enforces absolute time ordering on all operations
    - Always returns most recent value
    - Changes immediately visible to all processes
- Issues in distributed systems
  - Single file server (no client caching): easy to implement UNIX semantics
  - Client file caching: improves performance by decreasing demand at the server; updates to the cached file are not seen by other clients
  - Conclusion:?

Semantics of File Sharing

- **Session semantics** (relaxed semantics)
  - Local changes only visible to process that opened file
  - File close => changes made visible to all processes
  - Allows local caching of file at client
- Problem:
  - What if two or more clients are caching and modifying a file?

Semantics of File Sharing

- **No file update semantics** (Immutable files):
  - Files are never updated/modified
  - Allowed file operations: CREATE and READ
  - Files are atomically replaced in the directory
  - Problems:
    - what if two clients want to replace a file at the same time?
    - Delete file in use by another process

Semantics of File Sharing

- **Atomic transactions**
  - All file changes are delimited by a Begin and End transaction
  - All files requests within the transaction are carried out in order
  - The complete transaction is either carried out completely or not at all (atomicity)
  - Serializable access
  - Problem: ?

Operations on Files

- *Open, Close*
  - Gain or relinquish access to a file
  - OS returns a *file handle* – an internal data structure letting it cache internal information needed for efficient file access
- *Read, Write, Truncate*
  - *Read:* return a sequence of *n* bytes from file
  - *Write:* replace *n* bytes in file, and/or append to end
  - *Truncate:* throw away all but the first *n* bytes of file
- *Seek, Tell*
  - *Seek:* reposition *file pointer* for subsequent reads and writes
  - *Tell:* get current *file pointer*
- *Create, Delete:*
  - Conjure up a new file; or blow away an existing one

Methods for Accessing Files

- *Sequential* access
- *Random* access
- *Keyed* (or indexed) access