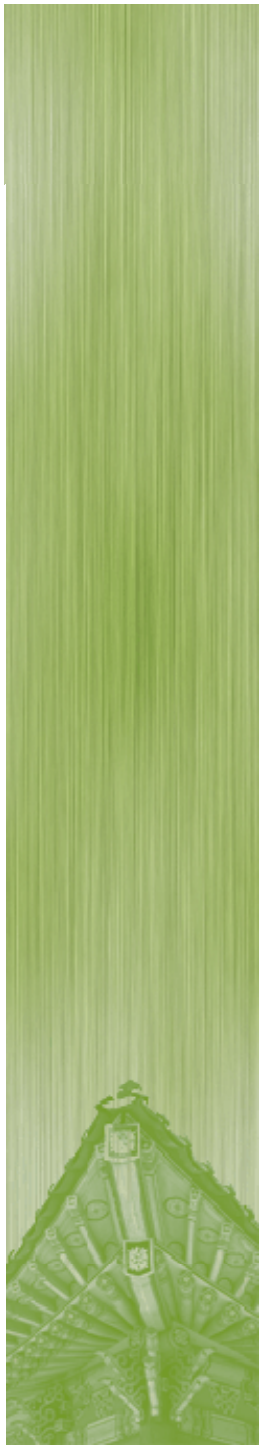


File Systems Overview

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

SKK UNIVERSITY

Today's Topics

- File system basics
- Directory structure
- File system mounting
- File sharing
- Protection

Basic Concepts

- **Requirements for long-term information storage**
  - Store a very large amount of information
  - Survive the termination of the process using it
  - Access the information concurrently by multiple processes
- **File system**
  - Implement an abstraction for secondary storage (files)
  - Organizes files logically (directories)
  - Permit sharing of data between processes, people, and machines.
  - Protect data from unwanted access (security)

Files

- **File**
  - A named collection of related information that is recorded on secondary storage.
    - persistent through power failures and system reboots
  - OS provides a uniform logical view of information storage via files.
- **File structures**
  - Flat: byte sequence
  - Structured:
    - Lines
    - Fixed length records
    - Variable length records

Storage: A Logical View

- **Abstraction given by block device drivers:**

- **Operations**
  - Identify(): returns N
  - Read(start sector #, # of sectors)
  - Write(start sector #, # of sectors)

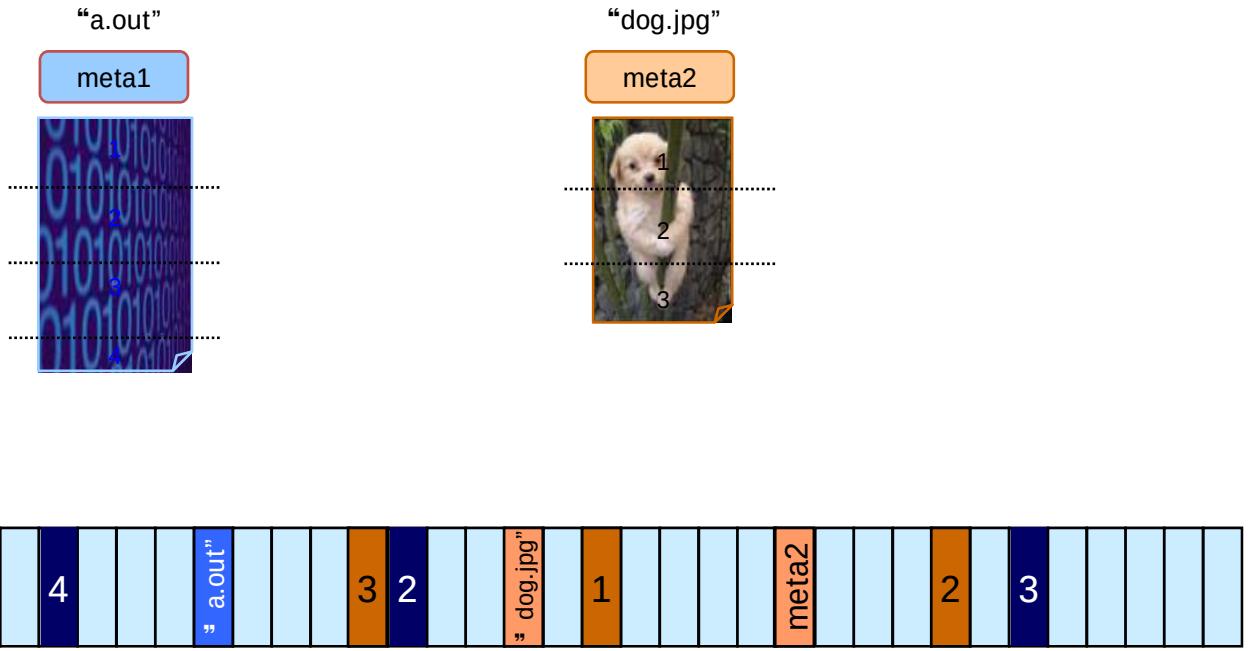
Source: Sang Lyul Min (Seoul National Univ.)

File System Basics (1)

- **For each file, we have**
  - File contents (data)
    - File systems normally do not care what they are
  - File attributes (metadata)
    - File size
    - Owner, access control lists
    - Creation time, last access time, last modification time, …
  - File name
- **File access begins with…**
  - File name
    - open (“/etc/passwd”, O_RDONLY);

File System Basics (2)

- **File system: A mapping problem**
  - <filename, data, metadata> → <a set of blocks>



File System Basics (3)

- **Goals**
  - Performance + Reliability
- **Design issues**
  - What information should be kept in metadata?
  - How to locate metadata?
    - Mapping from pathname to metadata
  - How to locate data blocks?
  - How to manage metadata and data blocks?
    - Allocation, reclamation, free space management, etc.
  - How to recover the file system after a crash?
  - …

File Attributes

- **Attributes or metadata**

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File Operations

- **Unix operations**

```
int creat(const char *pathname, mode_t mode);
int open(const char *pathname, int flags, mode_t mode);
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);
int stat(const char *pathname, struct stat *buf);
int chmod(const char *pathname, mode_t mode);
int chown(const char *pathname, uid_t owner, gid_t grp);
int flock(int fd, int operation);
int fcntl(int fd, int cmd, long arg);
```

Directories

- **Directories**
  - For users, provide a structured way to organize files
  - For the file system, provide a convenient naming interface that allows the implementation to separate logical file organization from physical file placement on the disk
- **A hierarchical directory system**
  - Most file systems support multi-level directories
  - Most file systems support the notion of a current directory (or working directory)
    - Relative names specified with respect to current directory
    - Absolute names start from the root of directory tree

Directory Internals

- **A directory is …**
  - Typically just a file that happens to contain special metadata
    - Only need to manage one kind of secondary storage unit.
  - Directory = list of (file name, file attributes)
  - Attributes include such things as:
    - size, protection, creation time, access time,
    - location on disk, etc.
  - Usually unordered (effectively random)
    - Entries usually sorted by program that reads directory.

Pathname Translation

- **open("/a/b/c", …)**
  - Open directory "/" (well known, can always find)
  - Search the directory for "a", get location of "a"
  - Open directory "a", search for "b", get location of "b"
  - Open directory "b", search for "c", get location of "c"
  - Open file "c"
  - (Of course, permissions are checked at each step)

- **System spends a lot of time walking down directory paths**
  - This is why open is separate from read/write.
  - OS will cache prefix lookups to enhance performance.
    - `/a/b`, `/a/bb`, `/a/bbb`, etc. all share the "`/a`" prefix

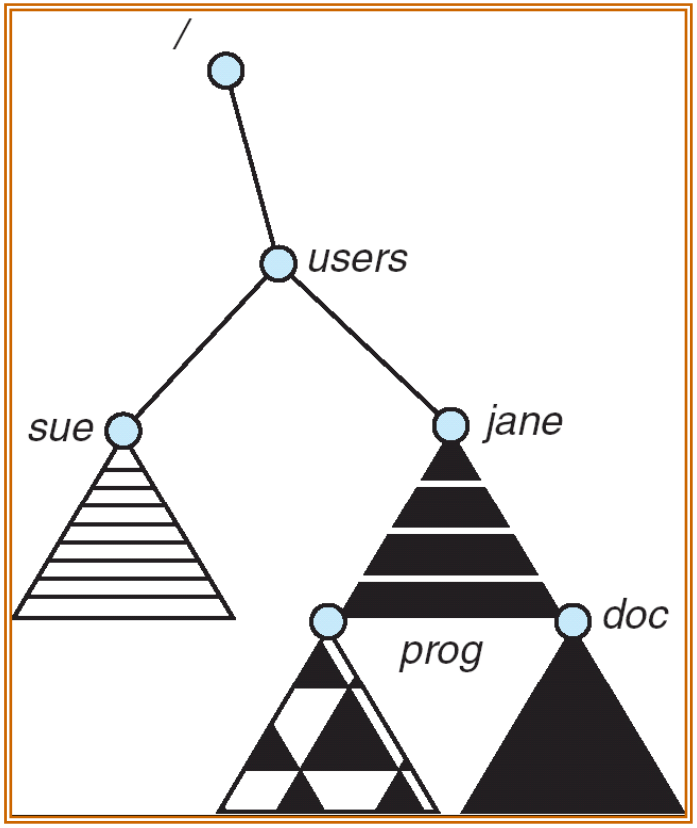
Directory Operations

- **Unix operations**
  - Directories implemented in files.
    - Use file operations to manipulate directories
  - C runtime libraries provides a higher-level abstraction for reading directories.
    - `DIR *opendir(const char *name);`
    - `struct dirent *readdir(DIR *dir);`
    - `void seekdir(DIR *dir, off_t offset);`
    - `int closedir(DIR *dir);`
  - Other directory-related system calls.
    - `int rename(const char *oldpath, const char *newpath);`
    - `int link(const char *oldpath, const char *newpath);`
    - `int unlink(const char *pathname);`

File System Mounting

- **Mounting**
  - A file system must be mounted before it can be available to processes on the system
    - Windows: to drive letters (e.g., C:₩, D:₩, …)
    - Unix: to an existing empty directory

      (= mount point)



File Sharing

- **File sharing**
  - File sharing has been around since timesharing.
  - File sharing is incredibly important for getting work done.
    - Basis for communication and synchronization.
    - On distributed systems, files may be shared across a network (e.g., NFS).
  - Three 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?
    - Concurrency control using locks
    - Protection

Consistency Semantics

- **UNIX semantics**
  - Writes to an open file are visible immediately to other users that have this file open at the same time.
  - One mode of sharing allows users to share the pointer of current location into the file.
    - via fork() or dup().
- **AFS session semantics**
  - Writes to an open file are not visible immediately.
  - Once a file is closed, the changes made to it are visible only in sessions starting later.
- **Immutable-shared-files semantics**
  - Once a file is declared as shared by its creator, it cannot be modified.

File Locking

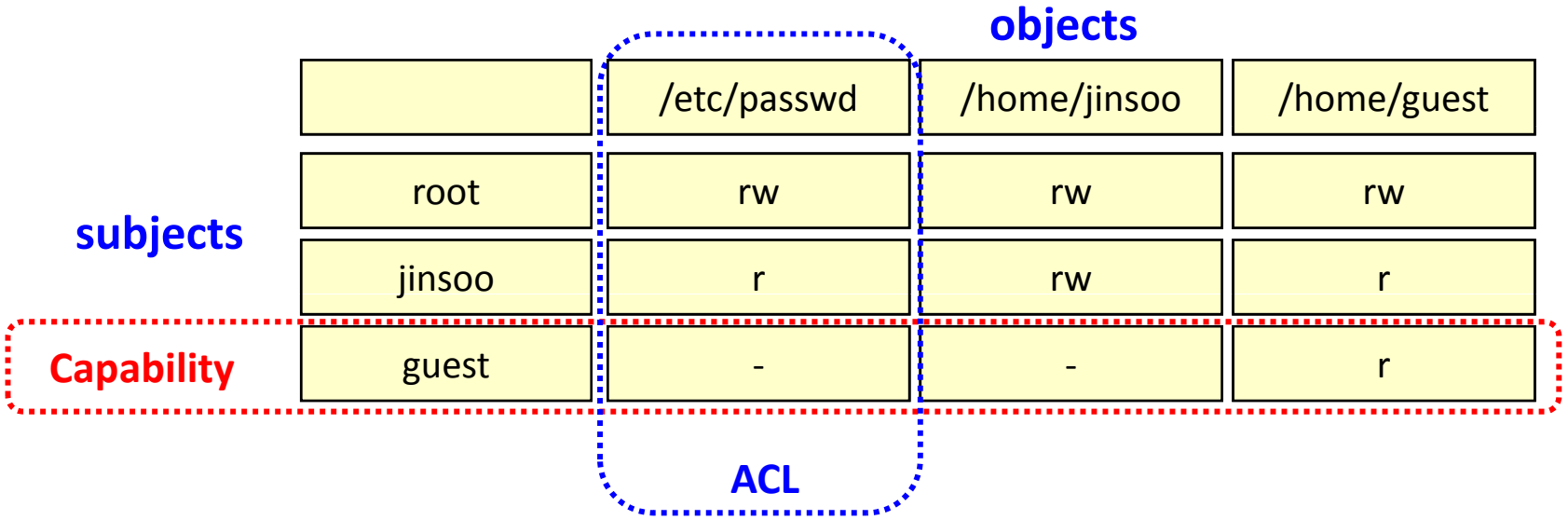
- **Advisory lock on a whole file**
  - int flock (int fd, int operation)
    - LOCK_SH(shared), LOCK_EX(exclusive), LOCK_UN(unlock)
- **POSIX record lock**
  - Discretionary lock: can lock portions of a file.
  - If a process dies, its locks are automatically removed.
  - int fcntl (int fd, int cmd, struct flock *lock);
    - cmd: F_GETLK, F_SETLK, F_SETLKW
    - struct flock { type, whence, start, len, pid };
- **System V mandatory lock**
  - A file is marked as a candidate by setting the setgid bit and removing the group execute bit.
  - Must mount the file system to permit mandatory file locks.
  - Use the existing `flock()/fcntl()` to apply locks.
  - Every `read()` and `write()` is checked for locking.

Protection (1)

- **Protection systems**
  - File systems must implement some kind of protection system.
    - to control who can access a file (user)
    - to control how they can access it (read/write/exec, etc.)
  - More generally:
    - generalize files to objects (the "what")
    - generalize users to subjects (the "who")
    - generalize read/write to actions (the "how")
  - A protection system dictates whether a given action performed by a given subject on a given object should be allowed.
    - You can read or write your files, but others can not.
    - You can read /etc/motd, but you cannot write to it.

Protection (2)

- **Representing protection**
  - Access control lists (ACLs)
    - For each object, keep list of subjects and their allowed actions.
  - Capabilities
    - For each subject, keep list of objects and their allowed actions.



Protection (3)

- **ACLs vs. Capabilities**
  - Two approaches differ only in how the table is represented.
  - Capabilities are easy to transfer.
    - They are like keys; can hand them off.
    - They make sharing easy.
  - In practice, ACLs are easier to manage.
    - Object-centric, easy to grant and revoke.
    - To revoke capabilities, need to keep track of all subjects that have the capability – hard to do, given that subjects can hand off capabilities.
  - ACLs grow large when object is heavily shared.
    - Can simplify by using "groups".
    - Additional benefit: change group membership affects all objects that have this group in its ACL.