TDIU11

Operating Systems

File-System Interface

[SGG7/8/9] Chapter 10

Christoph Kessler, IDA,
Linköpings universitet.

File-System Interface

- File Concept
- Access Methods
- Directory Structure
- File-System Mounting
- File Sharing
- Protection

How the file system looks to the user and to application programs

File Concept

- Primary memory is volatile
  - need secondary storage for long-term storage
- For now: A *disk* is a linear sequence of numbered blocks
  - With 2 operations: write block *b*, read block *b*
  - Low level of abstraction,
- Portability across different storage devices
- Solution: OS provides the ***file*** abstraction
  - Smallest allotment of secondary storage known to the user
  - Attributes (Name, id, size, …)
  - Typically, contiguous logical address space
  - Organized in a *directory* of files
  - API (operations on files and directories)

File Structure

- None - sequence of words, bytes
  - Most common – used by Unix, DOS, Windows, ...
  - Programs give meaning/structure to the byte sequence
  - Minimal requirement:
    The OS must understand its own executable format
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures

File Attributes

- **Name** – the only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can read, write, execute
- **Time, date, and user identification** – data for protection, security, and usage monitoring

Such information *about* files (i.e., **meta-data**) is kept in a **directory structure**, which is maintained on the disk.

Stored in a **File Control Block** (**FCB**) data structure for each file

File Operations (API)

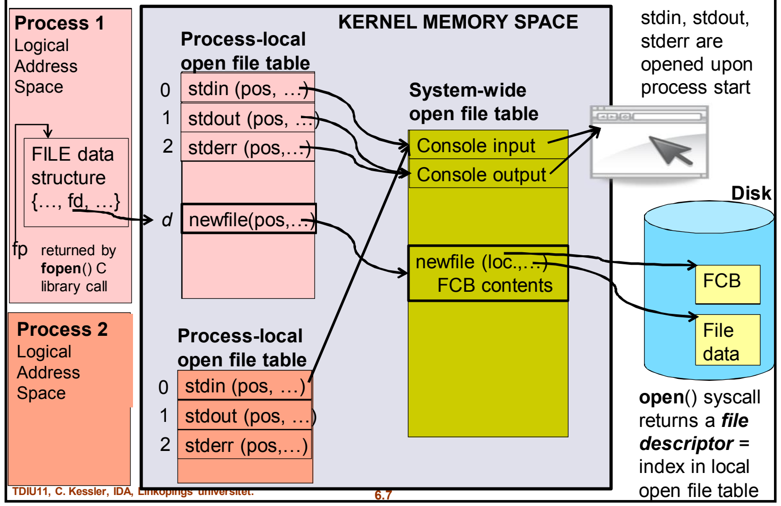
- **File** is an abstract data type, with operations
  - **Create**
  - **Write**
  - **Read**
  - **Reposition within file**
  - **Delete**
  - **...**
  - ***Open***(F_i) – search the directory structure on disk for entry F_i, and move the content of that entry to memory
  - ***Close*** (F_i) – move the content of entry F_i in memory to directory structure on disk

Unix / C example:

open (”filename”, ”mode”)

returns a ***file descriptor / handle***
= index into a per-process → table of open files (part of PCB)
(or an error code)

File descriptors and open file tables



stdin, stdout, stderr are opened upon process start

open() syscall returns a ***file descriptor*** = index in local open file table

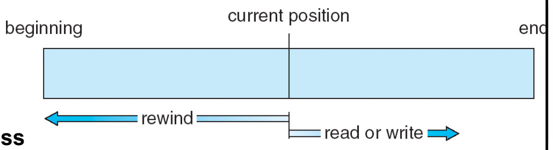
Open Files

- Data needed to manage open files:
  - **Disk location of the file** (and other metadata from FCB)
  - **File-open count**: count number of times a file is opened – to allow removal of data from open-file table when last process closes it
    - shared by all processes that opened the file
  - **File pointer (seekpos)**: pointer to next read/write location
    - one for every *open* system call (process)
- Collected in a system-wide table of open files and process-local open file tables (part of PCB)
  - process-local open file table entries point to system-wide open file table entries
  - Semantics of fork()?

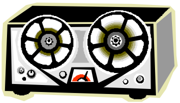
Open File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks (→ TDIU16)
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested (usually adopted by windows)
  - **Advisory** – processes can find status of locks and decide what to do (usually adopted by unix)

Access Methods



- **Sequential Access**



read next block
write next block
reset (rewind)

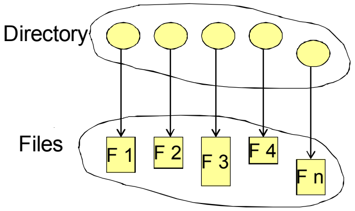
- **Direct Access**

read block *n*
write block *n*
position to *n*
    read next block
    write next block

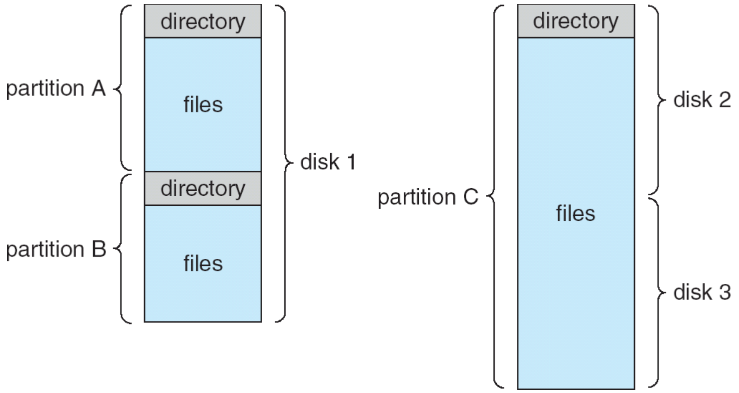
n = relative block number from beginning of file)

Directory Structure

- Files in a system must be organized in some way.
- **Directory**:
  A collection of *nodes* containing information about all files
- API:
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system
- Both the directory structure and the files reside on disk.



A Typical File-system Organization

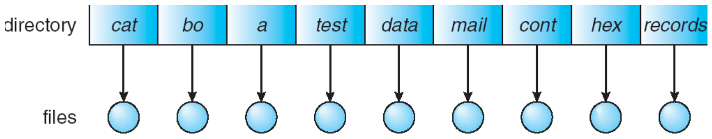


Organize the Directory (Logically) to Obtain …

- **Efficiency** – locating a file quickly
- **Naming** – convenient to users
  - Two users can use the same name for different files
  - The same file can have several different names
- **Grouping** – logical grouping of files by properties
  - e.g., all Java programs, all games, …

Single-Level Directory

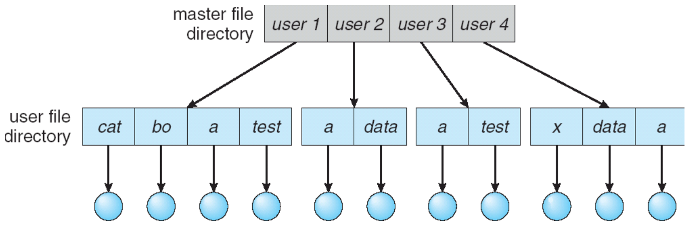
- A single directory for all users



Very simple
Naming problem
Grouping problem
Still used on simple devices, embedded systems, Pintos

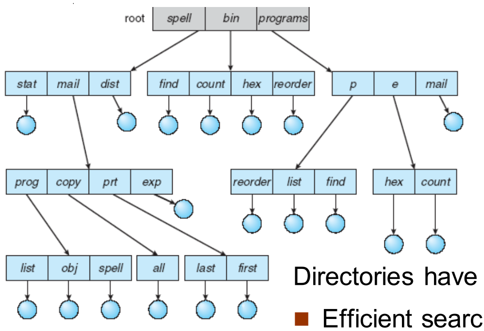
Two-Level Directory

- Separate directory for each user



- **Path name**: *username / filename*
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Directories have subdirectories

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - cd /spell/mail/prog; type list;
  - Absolute vs. relative file names

Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory

  touch <file-name>

- Delete a file in current working directory

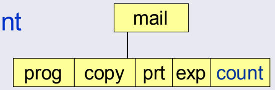
  rm <file-name>

- Creating a new subdirectory is done in current directory

  mkdir <dir-name>

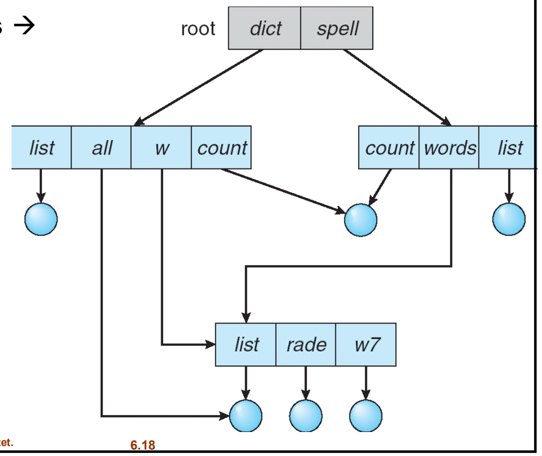
  Example: if in current directory /mail

  mkdir count



Acyclic-Graph Directories

- Have shared subdirectories and files
- Done with **links** →



Acyclic-Graph Directories (Cont.)

- Two different names (*aliasing*)
- If *dict* deletes *list* ⇒ dangling pointer

  Solutions:

  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution

- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

Hard links vs. Soft links (1)

- Example directory:

Name	Location
myfile	371
file2	524
…	
mylink_hard	371
…	
mylink_soft	./myfile
…	

Hard links vs. Soft links (2)

- **Hard links**
  - direct pointer (block address) to a directory or file
  - cannot span partition boundaries
  - need be updated when file moves to different place on disk
  - Unix: **ln** <filename> <linkname>

Name	Location
myfile	371
file2	524
…	
mylink_hard	**371**
…	
mylink_soft	./myfile
…	

Hard links vs. Soft links (3)

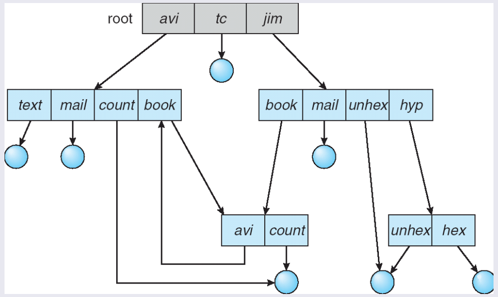
- **Soft links** (symbolic links, ”shortcut”, ”alias”)
  - files containing the actual (full) file name
  - still valid if file moves on disk
  - no longer valid if file name (or path) changes
  - Not as efficient as hard links
    - one extra block read
  - Unix: **ln –s** <filename> <linkname>

Name	Location
myfile	371
file2	524
…	
mylink_hard	371
…	
mylink_soft	**./myfile**
…	

Hard Links - Remarks

- If the entry in the directory contains size information, what happens if the file grows?
  - All directory entries pointing to this file must be updated… ☹
  - The Unix solution: The directory entries point to an ***inode*** (→) which contains file information
    - If the inode changes, see the change from all directories
- Hard links can cause (true) cycles in the file system
- Removal of hard links (including the original parent) can create disconnected subareas

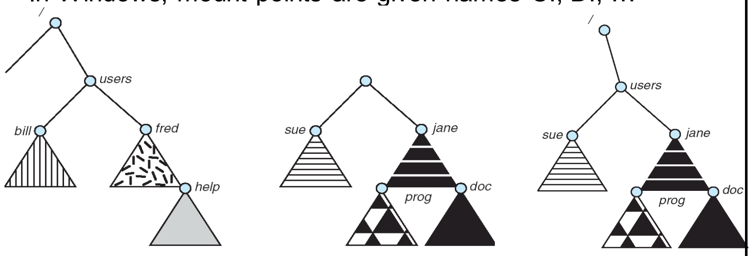
General Graph Directory



- How do we guarantee no cycles?
  - Every time a new hard link is added, use a cycle detection algorithm to determine whether it is OK (not for soft links)
  - Allow only links to files, not to directories
  - Garbage collection – mark all reachable files, delete the rest

File System Mounting

- A file system must be **mounted** before it can be accessed
- Mounting combines multiple file systems in one namespace
- An unmounted file system is mounted at a **mount point**
- In Windows, mount points are given names C:, D:, …



(a)
Existing
file system

(b)
Unmounted volume
residing on /device/dsk

Mount point:
Mounted /device/dsk
over /users

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
  - Network File System (NFS) is a common distributed file-sharing method
  - SMB (Windows shares) is another
- In order to have a protection scheme, the system should have
  - **User IDs** - identify users,
    allowing permissions and protections to be per-user
  - **Group IDs** - allow users to be in groups,
    permitting group access rights

Protection

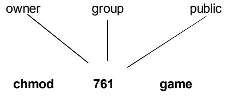
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- 3 modes of access: read, write, execute
- 3 classes of users:

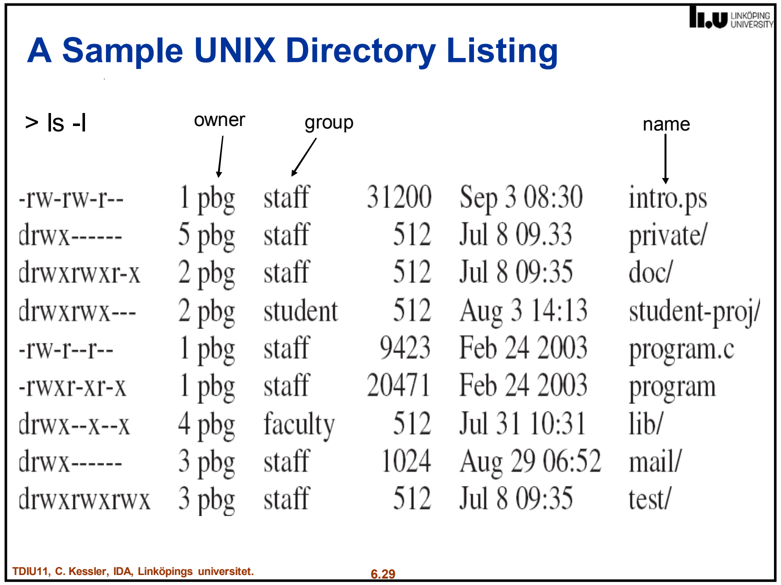
			RWX
a) **owner access**	7	⇒	1 1 1
			RWX
b) **group access**	6	⇒	1 1 0
			RWX
c) **public access**	1	⇒	0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file:

chgrp G game



A Sample UNIX Directory Listing

```
> ls -l                owner      group                               name

-rw-rw-r--      1 pbg   staff      31200   Sep 3 08:30    intro.ps
drwx------      5 pbg   staff        512   Jul 8 09.33    private/
drwxrwxr-x      2 pbg   staff        512   Jul 8 09:35    doc/
drwxrwx---      2 pbg   student      512   Aug 3 14:13    student-proj/
-rw-r--r--      1 pbg   staff       9423   Feb 24 2003    program.c
-rwxr-xr-x      1 pbg   staff      20471   Feb 24 2003    program
drwx--x--x      4 pbg   faculty      512   Jul 31 10:31   lib/
drwx------      3 pbg   staff       1024   Aug 29 06:52   mail/
drwxrwxrwx      3 pbg   staff        512   Jul 8 09:35    test/
```