DM510: File System Interface

Lars Rohwedder

Disclaimer

These slides contain (modified) content and media from the official Operating System Concepts slides: https://www.os-book.com/OS10/slide-dir/index.html

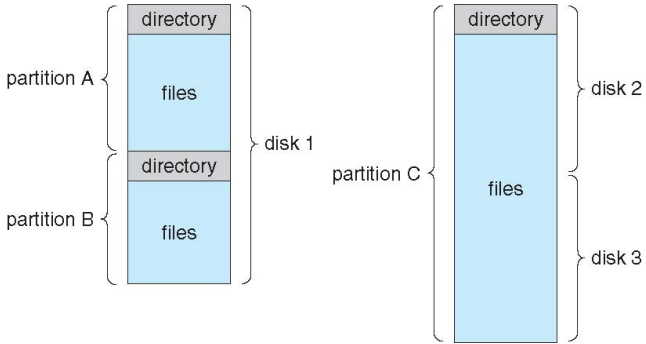
Today's lecture

- Chapter 13 of course book
- Part of Chapter 15
- Next lecture: Chapter 13 (Implementation of file systems)
- Next programming project: implement a file system

Overview

Role of a file system

- In the form of a **file system**, operating systems provide formats and routines to store and retrieve data (usually) on non-volatile storage
- A file system stores file contents (data in various formats), additional attributes for files, and directory information to logically organize the files



Files

File types

- Files have various roles: pictures, texts, executables...
- Internal file format varies greatly based on either text (ASCII, UTF-8, etc.) or raw binary data, often indicated by file ending
- Often sophisticated programmes exist to open or edit specific file formats

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

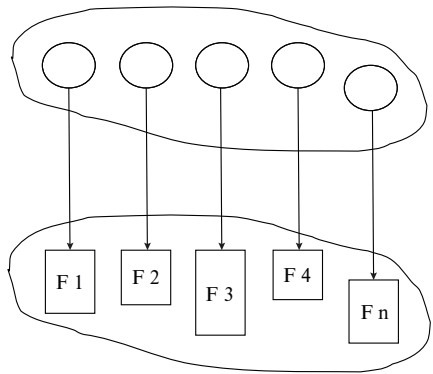
File attributes

Along with the file contents, various meta-data on files is stored

Examples

- **File name**
- **Identifier:** unique number used internally
- **Protection:** ownership and permissions
- **Timestamp**

File attributes are usually stored in directory section of file system



File protection

- Each file belongs to a specific user (“owner”) and a specific group (“owning group”). Even on systems used by only one person, there is usually at least one regular user and one superuser (root, admin, etc.)
- In Unix different permissions to read/write/execute a file are given to owner, users of file’s group, and other users. Sometimes indicated by three tripels:

  rwxrwxrwx

  Forbidden actions,are replaced by ’-’. For example, only owner can execute:

  rwxrw-rw-

  We can also write decimal number based on the three bits. Previous example:

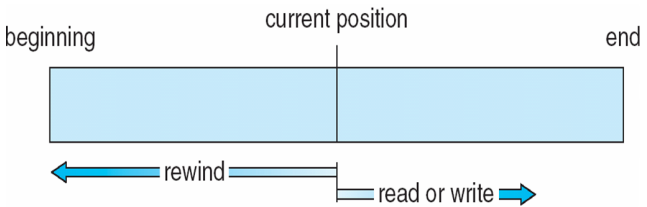
  766

  because $7 = 1\,1\,1 = \mathrm{rwx}$ and $6 = 1\,1\,1 = \mathrm{rw}-$
- Permissions and ownership can be changed by `chmod`, `chown`, and `chgrp`

File access

Sequential access

- We keep current position pointer with every open file
- Read and write increases current position after operation
- Rewind/seek operation to change current position



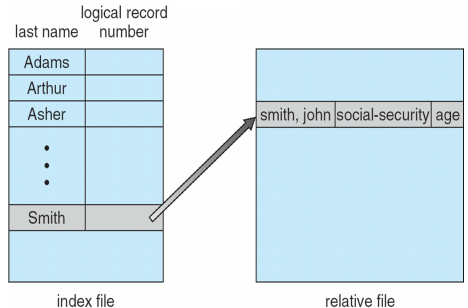
File access

Sequential access

- We keep current position pointer with every open file
- Read and write increases current position after operation
- Rewind/seek operation to change current position

Random access and other

- Contents of file may be accessed by position, like array
- In Unix: `mmap`, see previous demo
- For fast access: an index file (can be part of the same file) contains data structure that provides fast mapping of a key to position in file



Directories

Purpose of directories

Directories ...

- logically organize files
- make them easy to find
- group them by roles, e.g. in Unix: executables `/bin`, configuration files `/etc`, user specific files `/home/<username>`...

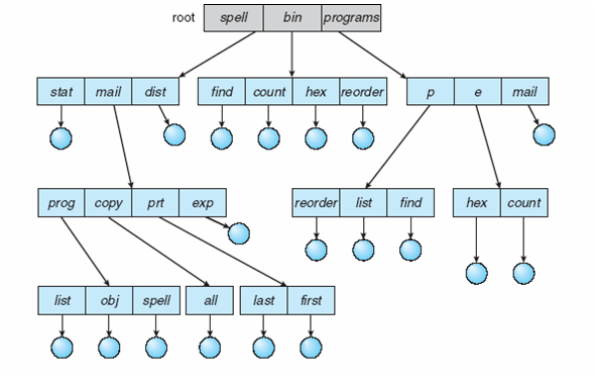
Should allow operations of

- searching for files, creating/deleting/renaming files and directories, listing directory content, traversing file system

Tree directory structure

- Natural recursive structure of files and directories: directories can contain other directories or files

- There is one special `root` directory `/`
- Each file or directory is uniquely identified by path from root, e.g.: `/spell/mail/exp`
- Implementation of operations (such e.g. searching) straight-forward

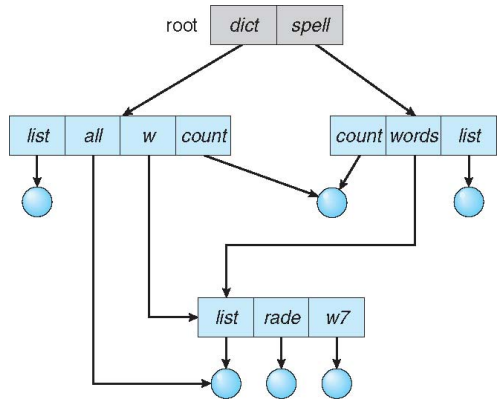


Circles: files

Rectangles: file or directory metadata

Acyclic graph structure

- Sometimes usefull to allow same file/subdirectory to be in several directories
- Operations more complicated: for efficiency, same directory should not be searched several times



Implementation via links

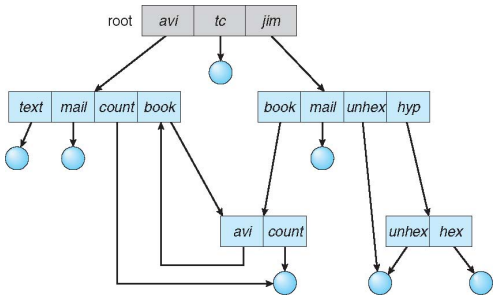
- **Hard links:** several directory entries point to same content
- Keep track of reference count (number of hard links to content) to know when it can be removed
- **Soft links:** directory entry redirects to other directory entry

General graph structure

- One could even allow cycles...
- Reference count not enough, may need time consuming garbage collection

Might be simpler to disallow cycles, but how?

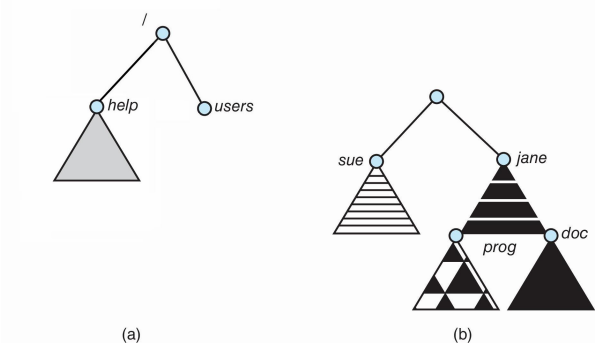
- Cycle detection (also time consuming)
- Disallow hard links for directories



File System Mounting

Mounting

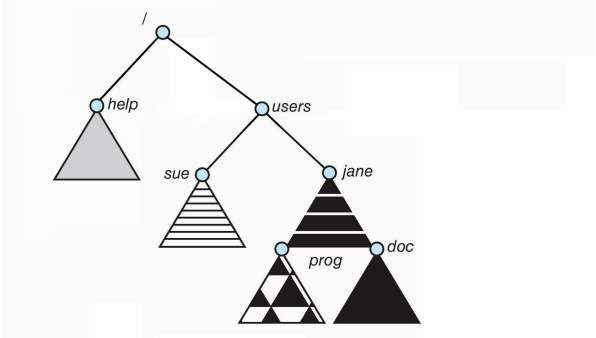
- Storage devices need to be **mounted** into root file system before they can be accessed
- For that, we mount device's file system into a **mount point**, in Unix a directory in the root file system
- If mount point was not empty, its content is not accessible until device is **unmounted**



Before mounting

Mounting

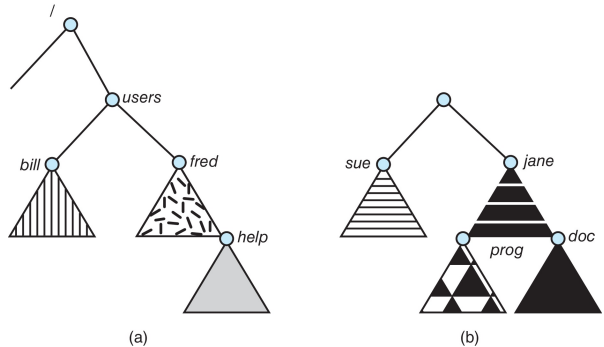
- Storage devices need to be **mounted** into root file system before they can be accessed
- For that, we mount device's file system into a **mount point**, in Unix a directory in the root file system
- If mount point was not empty, its content is not accessible until device is **unmounted**



After mounting

Mounting

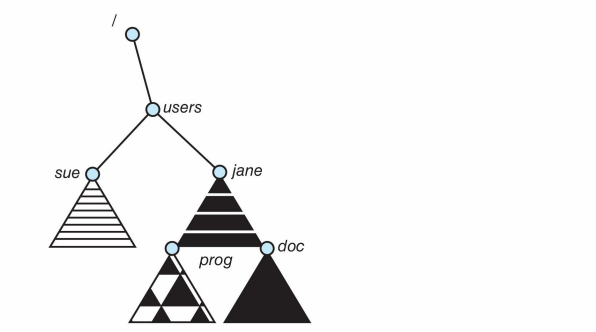
- Storage devices need to be **mounted** into root file system before they can be accessed
- For that, we mount device's file system into a **mount point**, in Unix a directory in the root file system
- If mount point was not empty, its content is not accessible until device is **unmounted**



Before mounting

Mounting

- Storage devices need to be **mounted** into root file system before they can be accessed
- For that, we mount device's file system into a **mount point**, in Unix a directory in the root file system
- If mount point was not empty, its content is not accessible until device is **unmounted**



After mounting

Virtual file systems and remote file systems

- Operating system often use abstraction of **virtual file system**, which allows different internal implementations, as long as they provide the necessary interface
- Some file systems may even be remotely connected through network (TCP or UDP)
- Mounting and using network storage creates other consistency problems, which we do not detail here (see Chapter 15 for more information).