File Systems I

DM510 Operating Systems

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

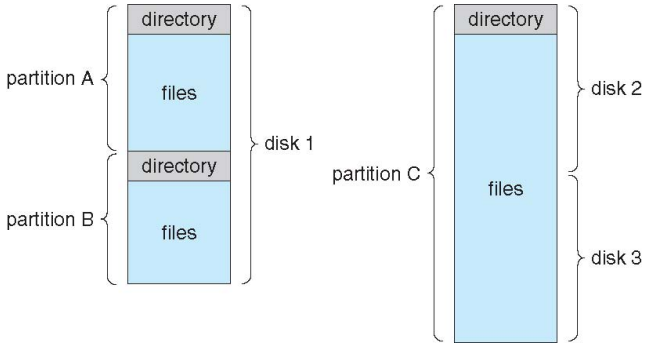
Today's lecture

- Chapter 13 of course book
- Part of Chapter 15
- Next lecture: Chapter 14 (Implementation of file systems)
- Next programming project: implement a file system

Overview

Role of a file system

- With a **file system**, the operating systems provide data structures and routines to store and retrieve data residing (usually) on a partition of a HDD or SDD
- A file system stores file contents (data in various formats), attributes for files, and directory information (names, paths) that logically organizes 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
- Sophisticated programmes are used 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

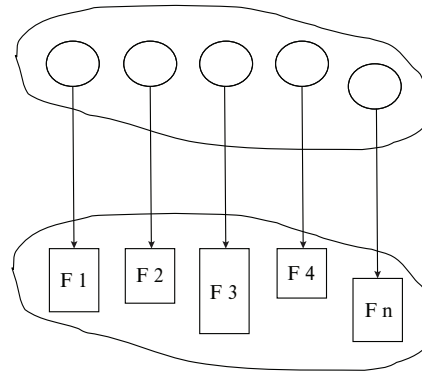
File attributes

Along with the file contents, attributes (meta-data) of files is stored

Examples of file meta data

- Name
- Identifier: unique number used internally
- Protection: ownership and permissions
- Timestamp
- Encoding
- Checksum
- ...

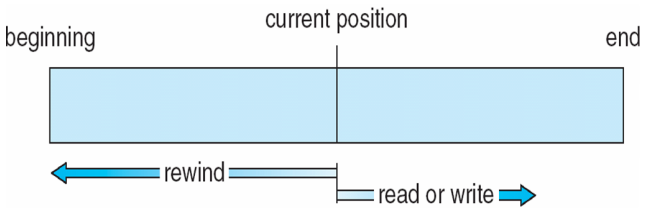
File attributes are usually stored in directory section of file system



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



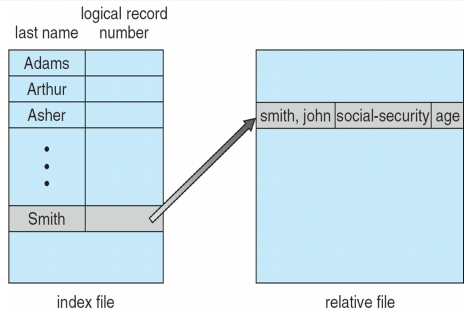
File access

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 indexes

- Contents of file may be accessed by position, for example, using memory mapping (`mmap` in Unix)
- 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 ...

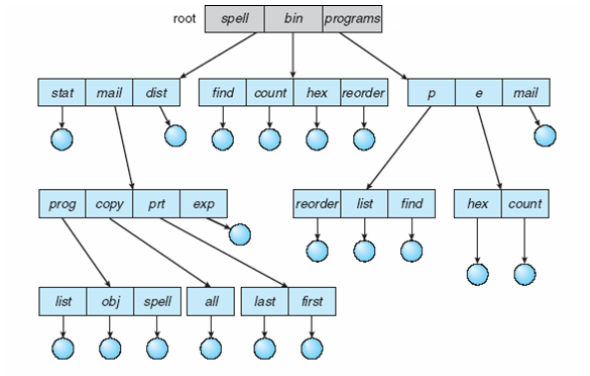
- logically organize files
- make files easy to find
- group them by roles, e.g. in Unix: executables `/bin`, configuration files `/etc`, user specific files `/home/<username>`...

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 as searching) is straight-forward

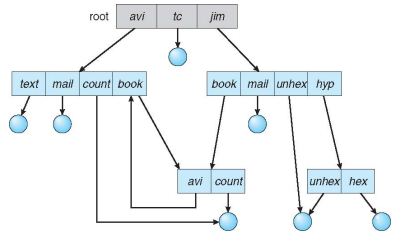
Circles: files
Rectangles: file or directory metadata

General graph structure

It can be useful to allow the same file or subdirectory to be in several directories

Implementation via links

- **Hard links:** several directory entries point to same content
- **Soft links:** directory entry redirects to other directory entry

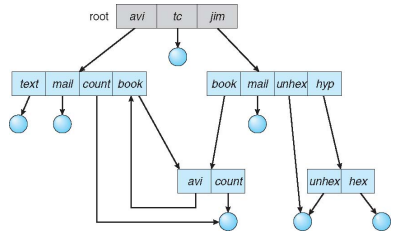


General graph structure

It can be useful to allow the same file or subdirectory to be in several directories

Implementation via links

- **Hard links:** several directory entries point to same content
- **Soft links:** directory entry redirects to other directory entry



This makes operations more complicated:

- Searching might search the same directory multiple times (or even get stuck in an infinite loop)
- It is unclear when we can delete file contents. This may require time-consuming garbage collection

General graph structure

It can be useful to allow the same file or subdirectory to be in several directories

Implementation via links

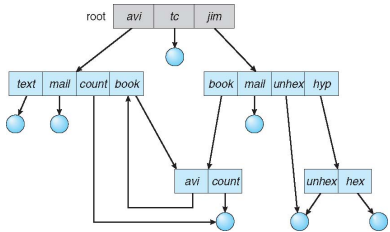
- **Hard links:** several directory entries point to same content
- **Soft links:** directory entry redirects to other directory entry

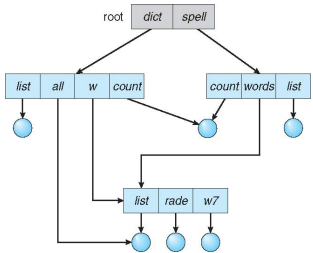
This makes operations more complicated:

- Searching might search the same directory multiple times (or even get stuck in an infinite loop)
- It is unclear when we can delete file contents. This may require time-consuming garbage collection

Without directed cycles many of the problems above can be solved, for example, reference counting would work. To ensure that the graph is a **directed acyclic graph (DAG)**, we can:

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



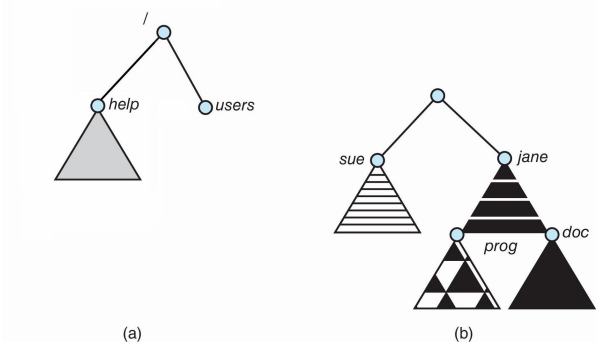


File System Mounting

Mounting

- Storage devices need to be **mounted** into root file system before they can be accessed

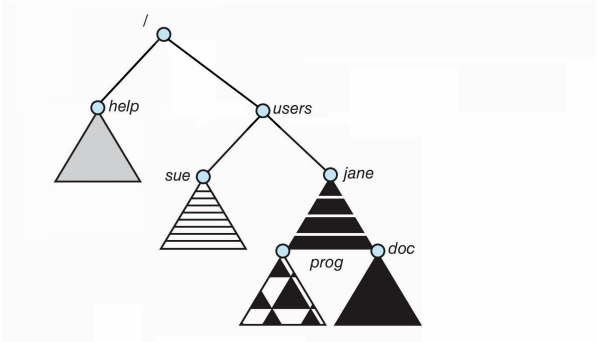
- For that, we mount device's file system into a **mount point**, in Unix a directory in the root file system
- If mount point was not empty, its content is not accessible until device is **unmounted**



Before mounting

Mounting

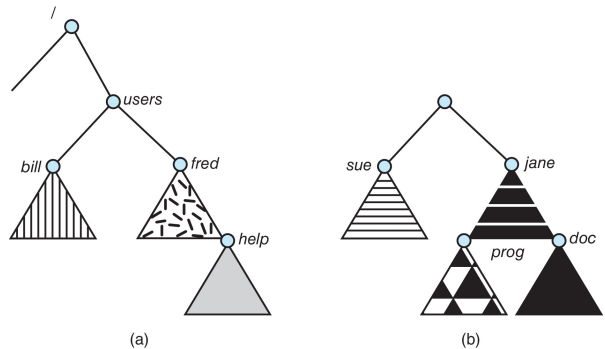
- Storage devices need to be **mounted** into root file system before they can be accessed
- For that, we mount device's file system into a **mount point**, in Unix a directory in the root file system
- If mount point was not empty, its content is not accessible until device is **unmounted**



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**

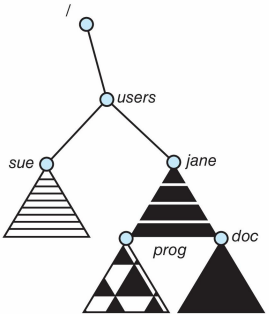


Before mounting

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 a number of new problems, which we do not detail here (see Chapter 15 for more information)
- With FUSE (Filesystem in Userspace) Linux allows file system implementation in user space that behave as other file systems. This will be used in the third programming project