File system

E. Campo M. Knoblauch Ó. López J. Clemente

Departamento de Automática
Universidad de Alcalá

/gso>

Index

1. Introduction
    - Definitions
    - Functionality of a file system
2. File system interface
    - Files and directories
    - Remote file systems
3. File system services
    - Usage of files by processes
    - File-related services
    - Directory-related services
    - Memory mapped files
4. File system organization
    - Structure and internal organization
    - Creation of file systems on disc
    - Mounting file systems
    - File systems reliability
5. Case studies
    - UNIX System V
    - FAT
    - Other file systems

File system

- Some processes need to use information that won't be fully loaded in their address spaces, and/or needs to be stored persistently

File system

Data structure, within a logical unit, that allows the operating system to store information in an organized manner that is independent of the processes that use it

- The file system takes care of abstracting physical properties of the different devices
- The information, being independent of the processes, is maintained after their completion. In addition, it can be used by several processes

File

File

Persistent data set (sequence) with an associated identifier seen as an entity by the user ⇒ minimal logic storage unit

- Files usually reside in permanent —non-volatile— storage devices: tapes, magnetic or optic discs, flash memories, etc.
- Files are accessed through system calls
- The can have a specific format, or none. It is the creator who defines the organization and the meaning of the bits that compose the file
- The goal is to achieve the abstraction of the storage device hardware

Functionality of a file system

- The file subsystem is the set of OS modules in charge of the interaction between the user and the stored information
- The interface offered to the user provides:
  - Naming services
    - Location
    - Extension
  - File services
    - Security, protection and encryption
    - Sharing
    - Access
    - Support for different file types
  - Directory services
    - Organization of the information

File attributes

- A file can be characterized by:
  - Name
  - Unique identifier
  - File type
  - Location
  - Size
  - Protection
  - Dates
  - Owner identification
  - Control information

File types

- The different file types supported must be clearly distinguishable
- Files can be grouped in two big classes: text files and binary files
- Techniques:
  - Include the type as part of the file name (extension): MS-DOS
  - Include some special value —magic number— inside the file: UNIX
- In a broader sense —not only plain sequences of bytes—, Linux, for example, supports: regular files, directories, links, pipes, etc.

File naming

- Main Feature: association of a name to a file upon its creation
- Allows access to the file unequivocally
- Names consist of alphanumeric and special characters
- The type of file names varies by OS:
    - Variable length: MS-DOS (8), UNIX (4.096)
    - They may have extensions to indicate the type of file:
        - Possibly different format and meaning per OS
        - Regardless of the extension, the OS should recognize at least the executables (magic number)

Methods of access to file contents

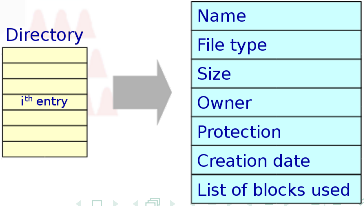
- Considering a file as a data sequence of fixed length, different in each file, how to access a specific datum?
- Operating systems can provide one or several access methods
- Basic access modes:
  - Sequential access
  - Indexed access: MVS
  - Direct access: UNIX

File protection

- It may be necessary to control access to information
- Modern operating systems provide mechanisms to protect and control access to files:
    - Schemes of privileges for users or groups for certain file operations:
        - Read, write, execute...
    - Access control lists (ACLs)
    - File system encryption

Directories

- System of organization of files that allows users to access and locate files easily
- They present a simple logical view to the user, so very different from the actual storage
- A directory is an object or data structure composed of the elements which groups together, called entries
- Each entry contains information about the file it references (MS-DOS) or a pointer to a data structure that contains this information (UNIX)



Name
File type
Size
Owner
Protection
Creation date
List of blocks used

Directory schemes (1/2)

- Single-Level Directory:
    - There is only one directory (CP/M)
    - There is no classification. Every file name must be different from the rest
- Two-Level Directory:
    - There is a master directory —root directory— containing other directories, one per user
    - For every user, operations on files are restricted to his/her directory
    - There may exist a special directory, accessible for every user, containing common executables:
        - PATH concept

Directory schemes (2/2)

- Tree-Structured Directories:
    - Naming complexity at user level ⇒ more general hierarchical structure
    - Allows users to sort their files into subdirectories. Examples: MS-DOS, UNIX, Windows
    - Representation of directories and subdirectories starting at the root directory
    - Every file in the system has a unique path name:
        - Concept of initial or connexion directory
        - Concept of current working directory
        - Concepts of absolute path and relative path

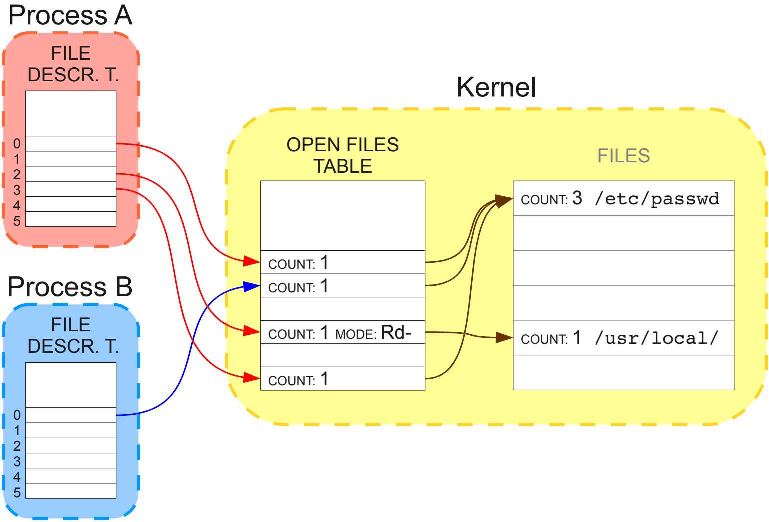
Remote file systems

- There are different file systems that provide access to files located on servers, for different users connected in remote locations
- The operating system must be able to manage network protocols and integrate remote units as part of the local file system
- Some of these systems are:
  - NFS
  - SMB - CIFS / SAMBA
  - CODA
  - AFS
- Not to be confused with file transference systems such as:
  - FTP
  - SCP

Accessing files from processes

- There are different access control tables to manage the communication of processes with files
  - File descriptor table
    - There is one per process. The user can access only an operation identifier ⇒ descriptor
  - Open files table
    - There is only one in the system. It contains as many entries as file operations have been permitted
    - Each entry is linked to a file using structures that depend on the type of file system

File access control tables



File system services handling files

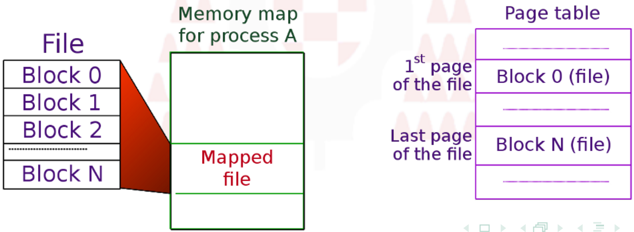
- Request to operate on a file (open)
- Cease operating (close)
- Read file contents
- Write contents in a file
- Change the operation pointer
- Consult file attributes
- Manipulate file attributes
- Etc.

File system services handling directories

- Create a new entry of any type of file and/or directory
- Eliminate a directory entry
- Request to operate on a directory (open, always for reading)
- Cease operating on a directory (close)
- Read the contents of a directory entry
- Modify the current directory
- Modify the root directory
- Obtain the path of the current working directory
- Etc.

Memory mapped files (1/2)

- Alternative way to access files through the use of virtual memory
- A program can request the OS to match (map) the contents of a file —or part of it— in its address space
  - The OS is responsible for maintaining the correspondence with the blocks of the mapped file
- When accessing a memory address associated with the mapped file, the process is accessing the file



Memory mapped files (2/2)

- Memory mappings, compared with ordinary read/write operations:
  - Accessing file data requires less system calls
  - Intermediate copies of the file system are avoided
  - File access is easier to program
- The mapping request can specify the desired protection:
  - Protection definition (read-only, read/write...)
  - Private or shared (on some systems). Example: dynamic libraries
- Main services: map and unmap

Allocation methods (1/2)

- Goal: make an efficient usage of the available disc space
- Allocation methods (broadly):
  - Place all the information together, in a row: provokes waste of space due to fragmentation
  - Use non-contiguous blocks: the block size must be defined; free blocks must be controlled
- Different methods to determine the blocks that are assigned to each file:
  - Contiguous allocation:
    - Only possible if the maximum size of the file is known upon its creation
    - The address of the first block is required
    - Advantage: no data fragmentation ⇒ high [time-]performance. The whole file can be read in a row with minimal seek times
    - Drawback: prohibitive internal and external fragmentation

Allocation methods (2/2)

- Management of blocks assigned to each file (continued):
  - Linked allocation:
    - The blocks of every file are organized forming a linked list
    - Problem: in order to reach block n, the previous $n-1$ blocks must be traversed
    - MS-DOS uses a duplicated File Allocation Table (FAT). Caching the information enhances performance
  - Indexed allocation:
    - An index block contains an array of pointers to the blocks. Any position can be reached by just looking up in this block
    - It allows direct access, but wastes space
    - The initial size of the index block limits the size of the file
    - Solution: use several index blocks, organized as a list or tree. UNIX uses a variant of this method

Free-space management

- The file system must keep track of the available blocks in order to assign them to new files or changing files
  - Bitmap (preferable, if it fits in memory):
    - A unit with *n* blocks uses a map of *n* bits
    - 1 means *free block*, and 0 means *occupied block*
  - Linked list of free blocks:
    - The OS maintains a pointer to the first free block, which contains a pointer to the next one...
  - Linked list with counts of contiguous free blocks:
    - Similar, but each block stores the position of the next block and a count of consecutive free blocks
  - Indexed list:
    - The first free block is used as an index of the other free blocks —usually in combination with counts of free blocks

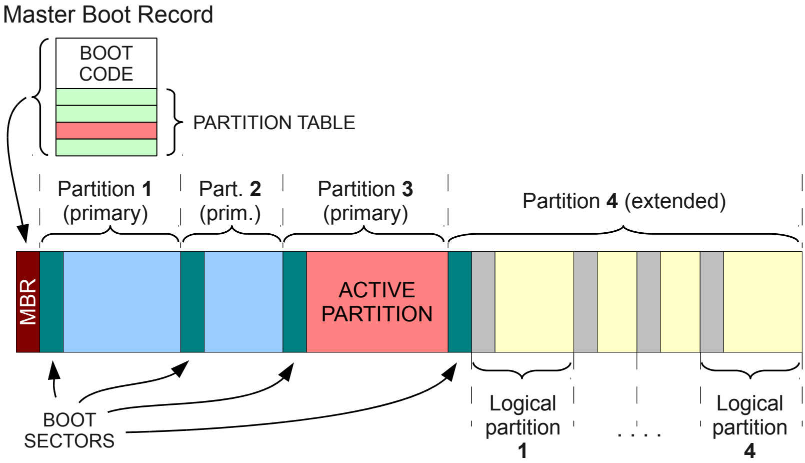
Partitioning (1/2)

- A file system must reside in one single disc unit
- Several types of file system —or several file systems of the same type— can coexist in a physical disc, if it is partitioned
- Partitioning a disc divides the physical disc in several logic discs
- In order to delimit the logical discs, some information must be stored in a reserved zone ⇒ Master Boot Record
- The MBR occupies the first disc sector, containing the boot program, a partition table with four entries, and the magic number `0xAA55`

Partitioning (2/2)

- The boot program zone can contain a boot manager, or simply a link to the bootable partition
- Every partition table entry specifies the type of partition, whether it is active or not, and the start and end positions in the disc
- In order to support more than four partitions per disc, a special type of partition exists, called extended partition
- An extended partition allows defining an unlimited number of partitions within it
- The magic number indicates whether the MBR is correct or not
- The rest of disc sectors can belong to a partition —containing a file system, or not— or be available for assignment to a new partition

Scheme of disc partitioning



Formatting

- Partitioning and formatting should not be confused. They are different and independent processes
- The partitioning establishes which part or parts of the disc will be available for a file system
- The formatting prepares a partition with the chosen file system
- The formatting consists in creating and initializing the data structures used by the file system to manage the occupied space and free blocks
- Therefore, the formatting consumes some space of the partition, and it is different for each file system type

Mounting file systems

- Design decision: provide just one single tree of directories, or several trees
  - On Windows systems: one tree per logic device
  - UNIX-like systems: just one single tree
  - The system must provide services for associating/de-associating file systems in the tree of unique names
  - UNIX commands or system calls: mount and umount

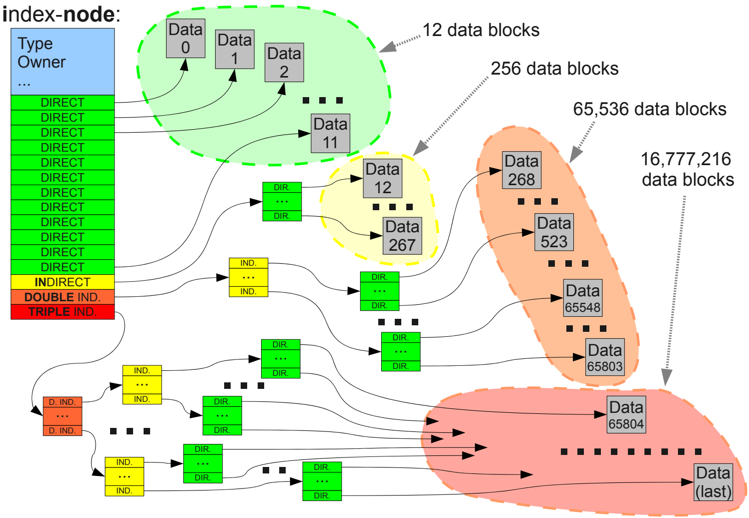
File systems reliability

- Some solutions are required just in case of possible physical media errors
- Data loss prevention mechanisms:
  - Backup copies
  - Version control
- Error recovery mechanisms:
  - In case of non-catastrophic errors —sectors with invalid data—:
    - Mark bad sectors by HW or SW
  - Against catastrophic errors —possible loss of information—:
    - Redundant discs (RAID 0, 1, 10, 5, 6, 60, ...)
  - Against file system errors —power cuts—:
    - Transactional file systems (journaling): JFS, ReiserFS, XFS, ...

Description of a UNIX-like file system

- Structure of single tree hierarchy
- Supports dynamically-growing files
- Allows protecting file data
- Maintains the independence from devices
- Everything is a file of some type:
  - Regular files, directories, block mode devices, character mode devices, pipes, sockets, symbolic links
- A table of index nodes allows accessing the files
  - Each index node contains 15 pointers to block (12 direct pointers, 1 simple indirect, 1 double and 1 triple)

Structure of a file's blocks

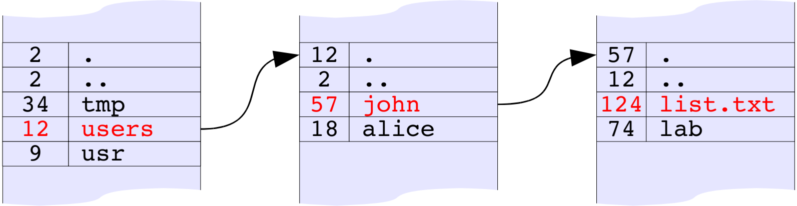


Structure of some file types

- Regular files
  - They simply contain the data
- Directories
  - They contain, at least, pairs inode-name
  - Upon creation, they already contain two entries
- Symbolic links
  - They contain a text string that refers the linked file by pathname
- Special files: block mode device, character mode device, pipe, socket
- Hard links are NOT a file type

From pathname to index node

- The nodes that correspond to directories are traversed according to the information found in them and the path to be resolved
- Depending on whether the path is absolute or relative, the starting point is the root directory or the current working directory
- Example:
    - `/users/john/list.txt` ⇒ index node 124



2	.
2	..
34	tmp
12	users
9	usr

12	.
2	..
57	john
18	alice

57	.
12	..
124	list.txt
74	lab

FAT file system (1/2)

- It contains, for every file, a linked list of the blocks that hold the data
- It has a File Allocation Table (FAT) that contains one entry per data block of the partition. Every entry can contain:
  - A free block identifier
  - A number indicating the next entry/block of the file
  - An end-of-file identifier, indicating that this is the last block of the file
- Due to the importance of the FAT, two copies of it are stored
- The two tables and the directory are stored in well known, fixed locations, in order to ease the system startup

FAT file system (2/2)

- Updating the FAT is vital, and takes a lot of time —the heads must move from the position of the file to the position of the FAT
- If the FAT is not updated periodically, some data might get to be inaccessible —unreachable, with nothing referencing them
- Drawbacks:
  - With very large partitions, a lot of space is wasted, since the FAT is very large too
  - Data fragmentation: blocks are assigned with no order
- Nowadays, FAT is not useful, because it was designed for systems with small discs and small main memories
- FAT32 is a successor of FAT, more robust and flexible, with some enhancements like, for instance, larger file systems and longer file names

Other file systems

- NTFS (New Technology File System)
  - Transactional file system
  - Supports file encryption
  - Provides access control for files and directories
- HFS+ (Hierarchical File System)
  - Used on Mac OS as an enhancement of the initial HFS
  - Maintains the file system information in a catalogue file
  - Used in hard discs or CDs, DVDs and IPod players. Recognised by Linux
- ZFS (Zettabyte File System)
  - Created by Sun Microsystems for Solaris
  - Transactional. Supports very large file systems and files (16 EB)
  - Allows making fast snapshots of the file system for backup copies

Bibliographic references

- [Sánchez, 2005] S. Sánchez Prieto.
  Sistemas Operativos.
  Servicio de Publicaciones de la UA, 2005.

- [Tanenbaum, 2009] A. Tanenbaum.
  Sistemas Operativos Modernos.
  Ed. Pearson Education, 2009.

- [Stallings, 1999] W. Stallings.
  Organización y arquitectura de Computadores.
  Ed. Prentice Hall, 1999.

- [Silberschatz, 2006] A. Silberschatz, P. B. Galván y G. Gagne
  Fundamentos de Sistemas Operativos.
  McGraw Hill. 2006