# Chapter 4.1

# The File System

## The User Perspective

Print Version of Lectures Notes of *Operating Systems*

Technical University of Cluj-Napoca (UTCN)
Computer Science Department

Adrian Coleșa

March 18th, 2020

4.1.1

## Purpose and Contents

### The purpose of today's lecture

- General Overview of The File System Module
- File Concept
- Directory Concept

4.1.2

### Bibliography

- Andrew Tanenbaum, *Modern Operating Systems*, 2nd Edition, 2001, Chapter 6, pg. 380 – 398.

4.1.3

## Contents

4.1.4

## 1 File System (FS) Overview

### Context

- users need to **store** and **retrieve**
  - **persistent data**
  - **large amount of data**
- users need to **share data**
- ⇒ OS should
  - **provide services** for such needs
  - **manage** storage devices (area) and stored data

4.1.5

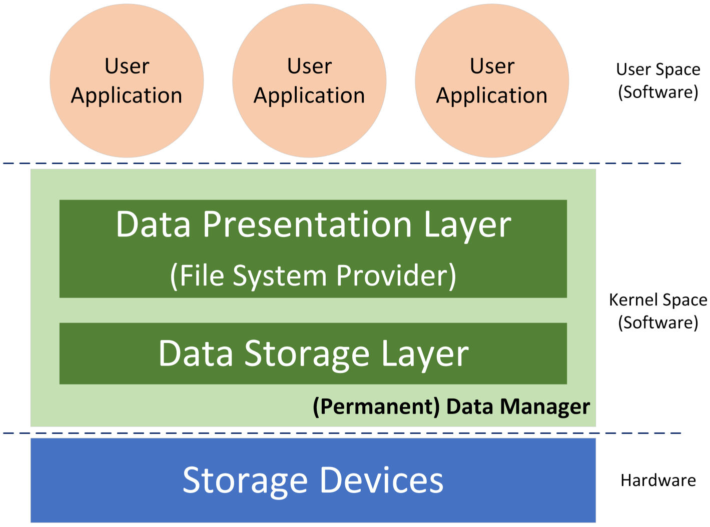


Figure 1: Data Manager's Layers. What we call "File System" is the way the "Data Manager" makes visible the data storage space to users.

### Definition

- **an OS's component**
  - ⇒ part of OS
- the **interface** between the user and the physical storage devices
  - *provides* the users access to the storage area
  - *manages* the information on the storage area
- ⇒ the (permanent) **data manager**

4.1.6

### Role

- **provides** the users a simplified, uniform and **abstract view of the storage area**
  - hides the complexity of physical storage devices
  - abstract concepts: *file* and *directory*
  - ⇒ *its name: File System*
- **manages** data on the storage area
  - interacts directly with storage devices
  - ⇒ contains a hardware dependent layer (code modules)

4.1.7

### File System Architecture. General View

4.1.8

### File System Architecture. Detailed View

4.1.9

# 2 Fundamental Concepts

## 2.1 File

### Definition. User Perspective

- the basic **storage unit** of information
  - anything the user wants to store must be placed in a file
- provides a (convenient) way to
  - **store** the information on storage devices
  - **retrieve** the information back later
- **a container**, i.e. **a box**
  - a collection of related information defined by its creator

4.1.10

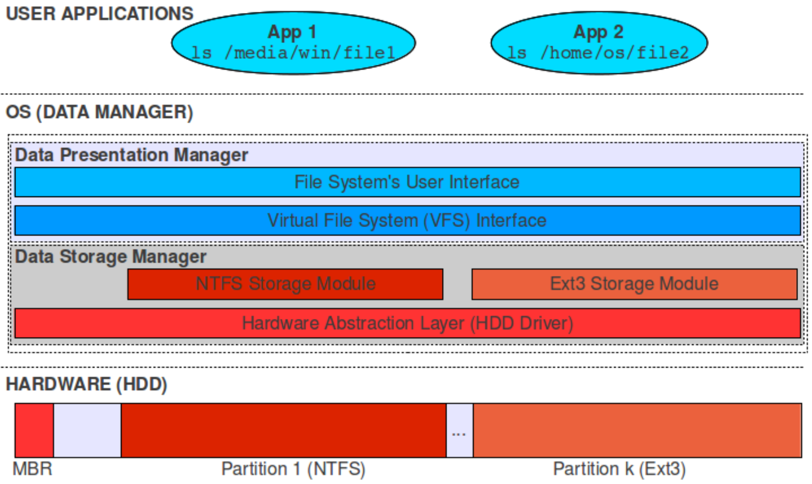
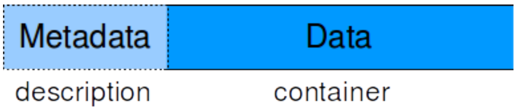


Figure 2: Data Manager's Components



# The file is the basic abstract concept for storing user data!

4.1.11

## Definition. OS (Internal) Perspective

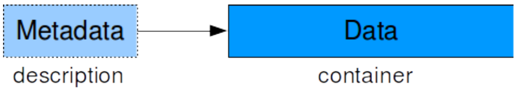
- an abstraction mechanism: a *container*
  - models the way data is provided to users
  - does not necessarily correspond to the way data is stored (on physical devices)
- OS must
  - **store** the container's contents on storage area
  - **retrieve** back later the container's contents
- the file maps the user view of data to the physical data
  - identifies on physical devices and links together physical data (i.e. chunks of bytes) belonging to a file

4.1.12

## File's Components

- **data**: user's useful information placed in the container
- **metadata**: description associated to stored data
  - needed, maintained and sometimes imposed by OS
  - some fields also used by user
- the two components **can be stored in different parts** on a storage area
  - there is a link maintained from metadata to data
  - ⇒ FS always **starts from a file's metadata** in order to access that file

4.1.13



### File Metadata: Name

- the user's way to **identify** (refer to) a file
- **must be unique** (there are though some exceptions!)
- each file must have (at least) a name
- consists of a **string of characters**
  - upper vs. lower letters, char sets
  - its length is generally limited
- file name's **extension**
  - normally **not imposed by OS**
  - *rarely*: may be recognized and used by the OS
  - *usually*: **a user-space convention** to provide hints about file's contents
  - required by some application (e.g. *gcc* requires .c)
  - examples: `book.`*`docx`*, `letter.`*`odt`*, `setup.`*`exe`*, `arch.tar.`*`gz`*, `progr.`*`c`*, `image.`*`jpg`*

4.1.14

# From the OS perspective every filename is just a string of (unrestricted) characters!

4.1.15

### File Metadata: Type

- Regular files
  - Contain user data (text or binary)
- Directories
  - System files used to organize file space
- Links
  - System files used to redirect the access to other files
- Special files
  - Model I/O devices
  - *Character*: terminals, mouse, etc.
  - *Block*: disks
- Pipes
  - Inter-process communication (IPC) mechanisms

4.1.16

### File Metadata: Structure

- **file's structure = the way the file's contents is organized and interpreted**
- possible strategies
  1. **no structure**: just a *sequence of bytes*
     - used by most OSes for normal files
  2. **specialized structures**: used normally for system files
     - sequence of (fixed-length) records
     - tree of records (e.g. B-Trees)

4.1.17

## File Metadata: Type vs. Structure – Pros and Cons

- different **file types** ⇔ different **file structures**
- that is **used for system files** (directories, links, pipes etc.)
  - managed by OS transparently from the user
- **But . . .** should the OS support **types and structures for regular files**?
  - **if, yes**
    - +: **efficient** / particular file's contents manipulation
    - +: convenient for novice users (not much / complex code)
    - -: additional OS code
    - -: restrictions, rigidity
    - -: file not portable
  - **if, no**
    - +: **flexible** ⇒ convenient for advanced users
    - +: no additional OS code
    - -: no OS support, additional application code
    - ⇒ **each application must manage / interpret itself the contents of files it works with**

4.1.18

## File Metadata: Type vs. Structure – Real Life OSes

- **no structure for regular files**
  - *text* vs. *binary* files: just a user convention
  - basically, **all files are binary**, i.e. each file's contents must be handled specifically by applications using it
  - ⇒ *text file* is just a particular file type
- ⇒ **flexibility** (*separate mechanism by policy*)
- **yet**, *every OS recognizes its own executable files*

4.1.19

# From the OS perspective every file is just a sequence of bytes!

4.1.20

## Question

**Yet, from the user perspective . . .** how would you classify the following files in terms of **text** vs **binary**, based on their filename extension?

1. `program.c`
2. `archive.zip`
3. `paper.pdf`
4. `index.html`
5. `persons.xml`
6. `app.java`
7. `cv.docx`

4.1.21

## File Metadata: Attributes

- system attributes
  - type
  - length
  - owner
  - access permissions (rights): basically *read*, *write* and *execute*
  - time stamps (e.g. creation time, last access time, last modification)

  - addresses of allocated blocks for that file
    - named during that course *Block Addresses Table* (BAT)
    - the link between metadata and data
  - . . .
- user attributes (if supported)
  - anything the user wants
  - examples: *source Web address* (for an html file), *place* (for an image file), *author* (for a document) etc. 4.1.22

### File Access Method

- **sequential** access
  - **impose an order** on the way the bytes are accessed
  - could be imposed by the storage device (e.g. tapes)
  - could be imposed by the nature of the file's contents
    - e.g. **specific for directories**
- **random** access
  - accesses the bytes or records **out of order**
  - specific to storage devices like hard disks
  - normally, **specific to regular user files** 4.1.23

### Operations On Files

- Access file data
  - *open*
  - **write**, i.e. *store* data
  - **read**, *get back* stored data
  - position (seek), i.e. *randomly* accessing stored data
  - *close*
- Manipulate files/Access file metadata
  - create
  - get/set attributes
  - rename
  - truncate
  - delete 4.1.24

## 2.2 Directory

### Definition. User Perspective

- the way to **organize / classify files**
  - when too many, difficult to find them based just on their name
- a **collection / class** of related elements (files)
  - a file could be placed in a directory (or more?)
  - ⇒ directory's name is a sort of additional (user) metadata associated to the file
- reduces the size of the filename space
  - files in different directories could have the same name
- imposes a structure / **hierarchy** on the file space
  - helps the user locating files easier in huge file spaces
  - **finding files visually** in the hierarchy is based on **navigation**
    - refine filters gradually, based on subdirectory names
  - does not help (too much) the **user applications locating a particular file**
    - if not knowing in advanced a file's complete path
    - ⇒ **file must be searched** in the entire FS tree 4.1.25

# The directory is the abstract concept for organizing user files! 4.1.26

### Definition. OS (Internal) Perspective

- an abstraction mechanism of grouping files
- a **container**: data (bytes) that must be stored
  - a **special file managed by OS** transparently from user
  - i.e. directory's bytes are interpreted by the OS and user-applications provided directory's elements
  - usually organized as specialized searching data structures, e.g. B-trees
- it also **consists of data and metadata**
  - very similar to a regular file
  - ⇒ just a FS-element from the OS perspective

4.1.27

### Directory Hierarchy

- types
  - single-level directory systems
  - two-level directory systems
  - **hierarchical** directory systems
    - the most general form: **tree** or **graph**
- **file paths**
  - the way of **identify a file in a hierarchy**
  - a list of consecutive nodes ending with the file name

4.1.28

### File Paths

- **absolute paths**
  - starting node is the **root directory**
  - examples
    - `c:\Program Files\Application\run.exe` (Windows)
    - `/home/students/adam/program.s` (Linux)
  - usage: access files at known fixed places
- **relative paths**
  - not staring from the root directory
    - starting node is the **current directory**
  - each application has its own current directory associated to it
  - examples
    - `Application\run.exe` (Windows)
    - `students/adam/program.s` (Linux)
  - special notations (directories)
    - current directory: '`.`'
    - parent directory: "`..`"
  - usage: access files in a subtree with a known structure, but located to an unknown location in the overall FS tree

4.1.29

# The directory concept imposes a hierarchy on the file space, requiring a path in order to identify a file!

4.1.30

### Question

Supposing the current working directory of an application is "`/home/os`", which will be the corresponding absolute paths for the following relative paths?

1. `file_1`
2. `./file_2`
3. `project/file_3`
4. `../file_4`
5. `../../file_5`
6. `../../../../file_6`
7. `../../etc/file_7`
8. `./../../etc/apache2/././../../home/os/./file_8`

4.1.31

### Operations On Directories

- create
- delete
- rename
- opendir
- readdir
  - **sequential access**
  - positioning not allowed anywhere
- rewind
- "write" (**not directly allowed**)
  - add elements (create directories and files)
  - delete elements (delete directories and files)
- close

4.1.32

## 3 Conclusions

### What We Talked About

- **file system (FS)** as an OS component
  - role
  - general structure
- **file** concept
  - used for **storing user data**
  - components: **data** (container component) and **meta-data** (description component)
  - meta-data fields: name, types, attributes etc.
  - **structure**: sequence of bytes, specialized collection of elements
- **directory** concept
  - used for **organizing the file space**
  - a system file: its bytes are interpreted by the OS and provided as a collection of elements
  - imposes a **hierarchy**
  - FS elements (e.g. files) specified by a **path**
  - absolute vs relative path

4.1.33

## Lessons Learned

- the basic storage unit for the user (applications) is the file
  - ⇒ storing just a single byte of data needs placing it in a file
- OS usually not involved in the interpretation of a file's contents
  - ⇒ file structure (format) is the user-application business
- text vs binary files is just a user convention
  - from the OS perspective all files consist just in a sequence of bytes
- FS organization, i.e. classifying files, is a real need
  - helps the user to navigate visually in the FS space in order to find her files
  - does not help (too much) the user applications finding files

4.1.34