4.3. THE VIRTUAL FILE SYSTEM

Jordi Garcia
2013-2014

The VFS

Contents

1. Concepts about the file system
   - The user view
2. The disk structure
3. Files in disk – The ext2 FS
4. The Virtual File System

The File System

Concepts (1)

- **Disk**: Device that stores information (files)
- Many files x many users: **OS management**
  - Files organization
    - Directories
    - Links
  - Files protection: `rwx rwx rwx`
    - Data file: read / write / execute
    - Directory:
      - `r` allows reading directory contents
      - `w` allows writing directory contents (add or remove files)
      - `x` allows changing directory

The File System

Concepts (2)

- **File system**: Logical view of a disc
  - The information is organized: files, directories
- Features
  - Inverted tree structure
    - Or graph (acyclic / cyclic), when links
  - Root directory: /
  - Current directory: .
  - Parent directory: . .
    - Exception: Parent of root directory
- All processes have a working directory: `cwd`
  - Relative path (assuming `cwd`)
  - Absolute path (`cwd` not assumed)

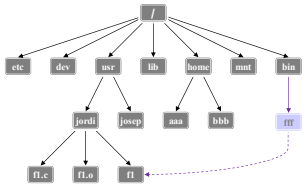
The File System

Concepts (3)

- **File**: Any object with a name in the File System
- File types
  - Regular file
    - Contains data
  - Directory
    - Contains files (regular, directory, ...)
  - Link
    - Soft / symbolic (points to the name)
    - Hard (points to the content)
  - Device
    - Type, major, minor
  - Named pipe
    - Type p
  - ...

The File System

Example



- With . and . . files in each directory
- If there are links, the tree becomes a graph

The File System

Additional system calls

- Usual I/O system calls, and

Service	System call
Create / remove link	`link / unlink / symlink`
Change file permissions	`chmod`
Change file proprietary / group	`chown / chgrp`
Get inode information	`stat / lstat / fstat`

- And other system calls to manage
  - Permissions
  - Relationship
  - Features
  - …

The File System

OS responsibilities

- … and guarantee **correctness, robustness, protection** and **efficiency**

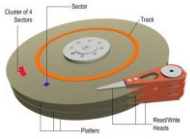
The VFS

Contents

1. Concepts about the file system
2. The disk structure
   - Disk structure
   - Disk contents
3. Files in disk – The ext2 FS
4. The Virtual File System

The disk structure

The disk



- Consists of: Platter/head/track/**sector**
  - All sectors have the same size
- It can also be seen as a list of sectors

SECTOR-1	SECTOR-2	SECTOR-3	SECTOR-4	...	SECTOR-N

The disk structure

Disk contents

- **Metadata**
  - Boot sector (optional, only if bootable disk)
  - Super block (one, or several through the disk)
    - Disk structure (metadata / data), size, ...
    - List of free sectors
  - Info about files (the data) storage
- **Data**: organized in blocks
  - 1 block (OS view) = N sectors (disk view)

BOOT sector	Super block	Info about files storage	Data block	Data block	Data block	Data block	Data block	...	Super block	Info about files storage	Data block	Data block	Data block	...

- The actual format depends on specific FS
  - ext2, ext3, NTFS, ...

The disk structure

Space allocation/free

- Disk allocation strategies
  - Continuous
    - For write-once disks
  - Discontinuous (at block level)
    - Linked list: FAT
    - Index table: **inode**
      - The **ext2** file system
- Free space management
  - Depends on allocation strategy
    - Bitmap of free blocks

The VFS

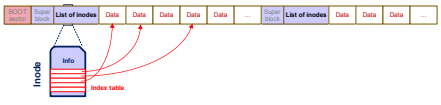
Contents

1. Concepts about the file system
2. The disk structure
3. Files in disk – The ext2 FS
   - Regular file
   - Directory
   - Links
   - Other types
4. The Virtual File System

Files in disk – ext2

Regular file (1)

- Contains: data (any type)
- Where: In data blocks, discontinuously
- File attributes
  - Packed into one data structure: **inode**
    - Metadata with **all information** about the file
    - … including the **index table**
    - … except the **file name** (!)



Files in disk – ext2

Regular file (2)

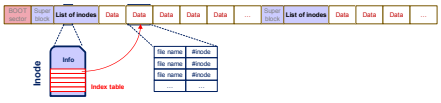
- The **inode** data structure
  - 1 inode per file

- File type: **data** *(dir, link, device, pipe, …)*
- Size (determines #blocks and offset in last block)
- Permissions
- Proprietary
- Dates (creation, modification, access)
- Number of references (later)
- …
- Index table (13 pointers)
  - 10 direct pointers
  - 1 single indirect pointer
  - 1 double indirect pointer
  - 1 triple indirect pointer

Files in disk – ext2

Directory file (1)

- Contains: List of files/inodes
- Where: In data blocks
  - Usually 1 block but, if more, discontinuously
- Data block contents
  - **file name** - #inode



  - Always, at least, `.` and `..` directories

Files in disk – ext2

Directory file (2)

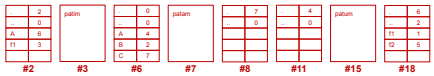
- The inode data structure
  - File type: dir
  - *All other fields are similar than regular file*
- Considerations in Linux
  - Root directory inode is always #0
  - `.` directory points to the same directory
  - `..` directory points to the parent
    - Except for root, that points to itself

Files in disk – ext2

Example (1)

- In Linux, the root inode is always **#0**
- Build the following file system:
  - Inodes

  - Data blocks



Links

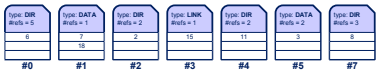
- Soft Link (or symbolic link)
  - **New file object** that points to a **path name**
  - The inode data structure contains
    - File type: **link**
    - All other fields are similar than regular files
  - The data block contains the path name
- Hard link
  - **New directory entry** (name) that points to an existing **inode**
  - There is not any specific inode for it
    - **Number of references ++**
  - There are not data blocks for it
  - Note that `.` and `..` are hard links

Other types

- Device
  - File object that represents a device in the FS
  - The inode data structure contains
    - File type: **char / block**
    - Major, minor, ...
    - No data blocks are used
- Named pipe
  - File object that represents a pipe in the FS
  - The inode data structure contains
    - File type: **pipe**
    - No data blocks are used
- And more: Sockets, ...

Example (2)

- In Linux, the root inode is always **#0**
- Build the following file system:
  - Inodes



  - Data blocks

The VFS

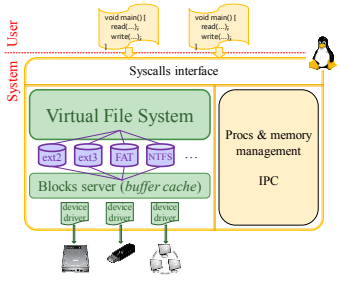
Contents

1. Concepts about the file system
2. The disk structure
3. Files in disk – The ext2 FS
4. The Virtual File System
   - Overview
   - Internal VFS data structures
   - System calls and the VFS

Virtual File System

Overview

- Abstraction layer on top of a specific FS



Virtual File System

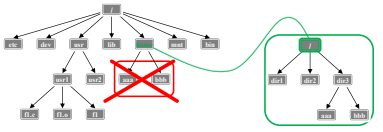
So ...

- **FS**: Information in disk, non-volatile, to store and organize files according to a predetermined format (ie: ext2, FAT, NTFS, ...)
  - Different FS offer different options, attributes, security levels, ... but it is independent of the user interface and VFS implementation
- **VFS**: Information in memory, part of the OS, to access and manage the files at run-time (from a process)
  - Contains a copy of the information in disk (metadata) to speed up the access times

Virtual File System

Mounting FS

- Different devices and/or partitions can be accessed in a system
  - Devices: hard disks, pen drives, network units
  - Partitions: Set of consecutive sectors in disk (metadata and data blocks), managed as independent logic entity
  - Each device and/or partition has its own directory structure and FS implementation
- In order to access them, they have to be **mounted** in the main directory tree
  - `mount -t ext2 /dev/hda1 /home`



Virtual File System

Internal VFS data structures

- **Channels Table** → done!
  - Defines the files that can be accessed by a process
- **Open Files Table** → done!
  - Represents the opened files in the system
- **Inodes Table** → done!
  - Contains the used files, a "copy" of the inode object
    - And other run-time information, such as **#refs**
    - *Actually, **inodes** in memory (IT) are called **vnodes**, and are a superset to represent the inodes info + other FS related info*
- **Buffer cache** → new!
  - Contains a copy of data blocks (for any device)
  - Used to speed access times and facilitate sharing
  - *Also, dentry cache with a copy of directory entries*

Virtual File System

`open` and the VFS (1)

- The `open` syscall receives a file name (path) as a parameter and, therefore, it has to access the VFS
  - Sequence of inodes and directory data blocks
  - Until the last inode is reached (and added to the IT, if first time)
    - No data blocks are accessed!
- 1 OFT entry is added
- 1 CT entry is added
- In case of caches, most of these disk access may be avoided or, later, reused

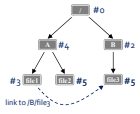
Virtual File System

`open` and the VFS (2)

- If the opened file is new (create), a new line in the directory data block will be added
  - Create new indode (and write to the disk)
  - Modify directory data block
  - Modify directory inode (data, size)
  - And modify super block (list of free inodes)

Virtual File System

Example (3)



- Compute number of disk accesses to execute:
  - `open("/A/file1", O_RDONLY);`
  - `open("/A/file2", O_RDONLY);`
  - Assuming:
    - 1 inode = 1 block
    - 1 directory = 1 block
    - No caches of any type are used

Virtual File System

`read` and the VFS

- The `read` syscall reads data blocks from disk, as long as they are not in the buffer cache
  - 1 or more, depending on size and offset
  - Also, depends on whether blocks are pointed by direct or indirect indexes in the inode

Virtual File System

`write` and the VFS

- The `write` syscall writes data blocks to the buffer cache (if present)
  - Later (periodically), data blocks from the buffer cache are written back to the disk
- Assuming N blocks have to be written (depending on size and offset), special attention has to be paid to the first and last
  - If initial offset is not block start, then this data block has to be read before writing
  - If final offset is not block end, then this data block has to be read before writing
  - All other data blocks are written directly
- Also, depends on whether blocks are pointed by direct or indirect indexes in the inode

Virtual File System

File deletion in the VFS

- Regular file
  - Decrements #refs and if zero removes inode + data blocks
  - Removes directory entry for this file
- Directory
  - Some VFS check if directory is empty; otherwise remove recursively all files included in it
  - Decrements #refs and if zero removes inode + data block
  - Removes directory entry for this file
- Soft link
  - Decrements #refs and if zero removes inode + data blocks
    - Does not remove the file pointed by this link
  - Removes directory entry for this file
- Other special files, just remove the inode

Virtual File System

Last considerations

- The presence of links in a FS turns the directory tree into a directory graph
  - If the graph is cyclic, the OS has to control infinite loops during file access, backup, …
- Different OS strategies to manage this
  - Cycles with soft-links are easy to manage. For instance, the backup process does not follow any soft-link
  - Cycles with hard-links are really difficult to manage, so
    - Deny hard links if they cause a cycle
    - Implement some type of mechanism to detect cycles