Sistemas de Operação / Fundamentos de Sistemas Operativos
File systems

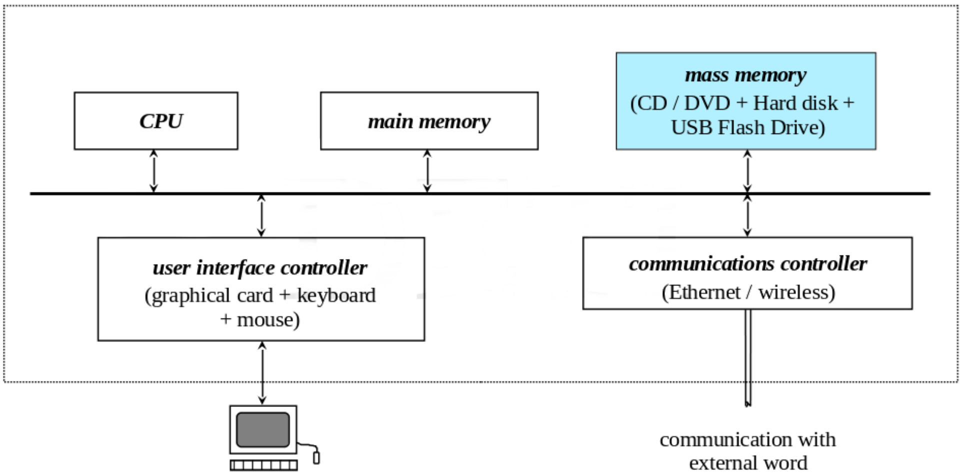
Artur Pereira <artur@ua.pt>

DETI / Universidade de Aveiro

Outline

Overview
The mass storage

- Simple view of a computational system, highlighting the mass storage component



Overview
Importance of mass storage (secondary memory)

- Storage of the operating system
  - When a computing system is turned on, there is only one program in main memory (in a small ROM-like region), the boot loader, whose main function is to read from a specific region of mass storage a larger program that loads into memory main, and runs, the program that implements the user interaction environment
- Warehouse of applications
  - For a computer system to perform useful work, a permanent place where to store the different applications must exist
- Warehouse of user files
  - Furthermore, almost all programs, during their execution, produce, consult and/or change variable amounts of information that more or less permanently must be stored

Overview
Properties of mass storage

- non-volatility – information exists beyond the processes that produce and/or use it, even after the computer is turned off
- large storage capacity – the information manipulated by the computer processes can far exceed that which is directly stored in their own address spaces
- accessibility – access to stored information should be done in the simplest and fastest way possible
- integrity – the stored information must be protected against accidental or malicious corruption
- sharing of information – the information must be accessible concurrently to the multiple processes that make use of it

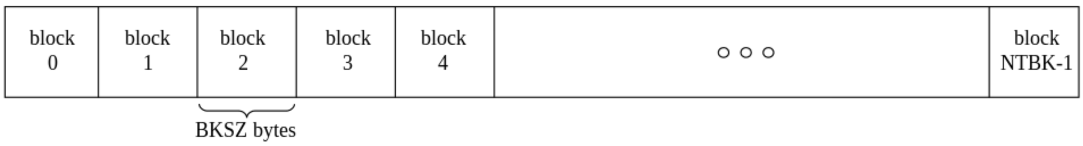
- File system is the part of the operating system responsible to manage access to mass storage contents

Mass storage
Types of mass storage devices

Type	Technology	Capacity (Gbytes)	Type of use	Transfer rate (Mbytes/s)
CD-ROM	mechanical / optical	0.7	read	0.5
DVD	mechanical / optical	4–8	read	0.7
HDD	mechanical / magnetical	250–4000	read / write	480
USB FLASH	semiconductor	2–256	read / write	60(r) / 30(w)
SSD	semiconductor	64–512	read / write	500

Mass storage
Operational abstraction of mass storage

- Mass storage is seen in operational terms as a very simple model
  - each device is represented by an array of NTBK storage blocks, each one consisting of BKSZ bytes (typically BKSZ ranges between 256 and 32K)
  - access to each block for reading or writing can be done in a random manner
- This is called Logical Block Addressing – LBA
  - Blocks are located by an integer index (0, 1, ...)
  - The ATA Standard included 22-bit LBA, 28-bit LBA, and 48-bit LBA

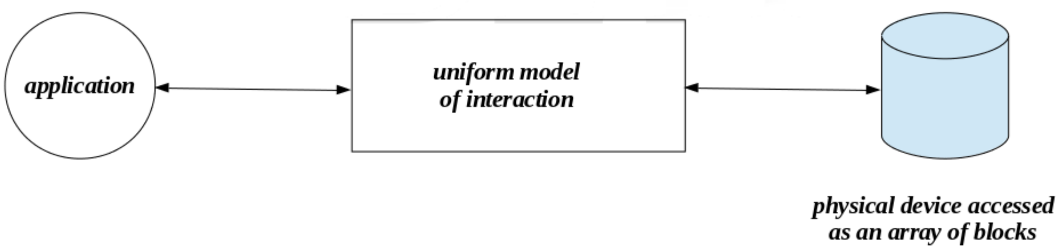


- Note that:
  - a block is the only unit of interaction
  - thus, a single byte **can not** be accessed directly

- What to do to change a byte of a block?

Mass storage
Application abstraction of mass storage

- Some considerations:
  - Despite creating a uniform model, LBA is not an appropriate way for an application to access mass storage data
  - Direct manipulation of the information contained in the physical device can not be left entirely to the responsibility of the application programmer
  - Access must be guided by quality criteria, in termos of efficacy, efficiency, integrity and sharing
- Thus, a uniform model of interaction is required



- Solution: the **file concept**

File concept
What is a file?

- file is the logical unit of storage in mass storage
  - meaning that reading and writing information is always done within the strict scope of a file
  - But have in mind that the physical unit of interaction is the block

- Basic elements of a file:
  - name/path – the user (generic) way of referring to the information
  - identity card – meta-data (ownership, size, permissions, times, ...)
  - contents – the information itself, organized as a sequence of bits, bytes, lines or registers, whose precise format is defined by the creator of the file and which has to be known by whoever accesses it

- From the point of view of the application programmer, a file is understood as an abstract data type, characterized by a set of attributes and a set of operations

File concept
Types of files

- From the operating system point of view, there are different types of files:
  - ordinary/regular file – file whose contents is of the user responsability
    - from the operating system point of view it is just a sequence of bytes
  - directory – file used to track, organize and locate other files and directories
  - shortcut (symbolic link) – file that contains a reference to another file (of any type) in the form of an absolute or relative path
  - character device – file representing a device handled in bytes
  - block device – file representing a device handled in blocks
  - socket – file used for inter-process and inter-machine communication
  - (named) pipe – file used for inter-process communication

- Note that text files, image files, video files, application files, etc., are all regular files

File concept
Attributes of files

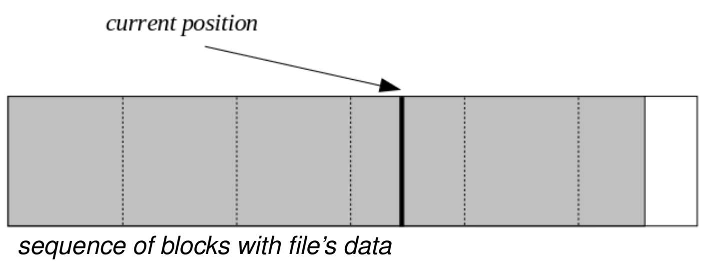
- Common attibutes of a file
  - type – one of the referred above
  - name/path – the way users usually refer to the file
  - internal identification – the way the file is known internally
  - size(s) – size in bytes of information; space occupied on disk
  - ownership – who the file belongs to
  - permissions – who can access the file and how
  - access and modification times – when the file was last accessed or modified
  - location of data in disk – ordered set of blocks/clusters of the disk where the file contents is stored

---

- Remember that a disk is a set of numbered blocks (the LBA model)

File concept
Operations on files (1)

- Common operations on regular files
  - creation, deletion
  - opening, closing – direct access is not allowed
  - reading, writing, resizing
  - positioning – in order to allow random access



File concept

Operations on files (2)

- Common operations on directories
  - creation, deletion (if empty)
  - opening (only for reading), closing
  - reading (directory entries)
    - A directory can be seen as a set/sequence of (directory) entries, each one representing a file (of any valid type)

- Common operations on shortcuts (symbolic links)
  - creation, deletion
  - reading (the value of the symbolic link)

- Common operations on files of any type
  - get attributes (access and modification times, ownership, permissions)
  - change attributes (access and modification times, permissions)
  - change ownership (only root or admin)

Directories

Concept

- Common disks may contain thousands or millions of files
  - It would be impractical to have all that files at the same access level
- Directory is a mean to allow the access to disk contents in a hierarchical way
- From a user point of view, a directory can be seen as a container containing files and other directories
- From an implementation point of view, a directory can be seen as a table of directory entries
- Every directory entry is a key-value pair that directly or indirectly associates the name of a file to its attributes

	name	**attributes**
ent[0]		
ent[1]		
ent[2]		
...		
ent[n]		

Directories
Name and path

- The existence of a file hierarchy makes the name insufficient to reference a file in a disk
  - Different files in the hierarchy can exist having the same name
  - How to access a file giving just its name? Not easy, if possible
- The notion of path must be introduced
  - A path is a sequence of names where all but the last must be directory names or shortcuts pointing to directories
  - In Unix, character / is used as separator
  - In Windows, character \ is used as separator
  - Names `.` and `..` have special meanings
- A path can be absolute or relative
  - An absolute path references the location of a file from a root point
  - A relative path references the location of a file from an intermediate point
- In Unix, the root point is a single, global root file hierarchy
  - Different storage devices are *mounted* somewhere in this hierarchy
- In Windows, there is a root point per storage device (A:, B:, ...)

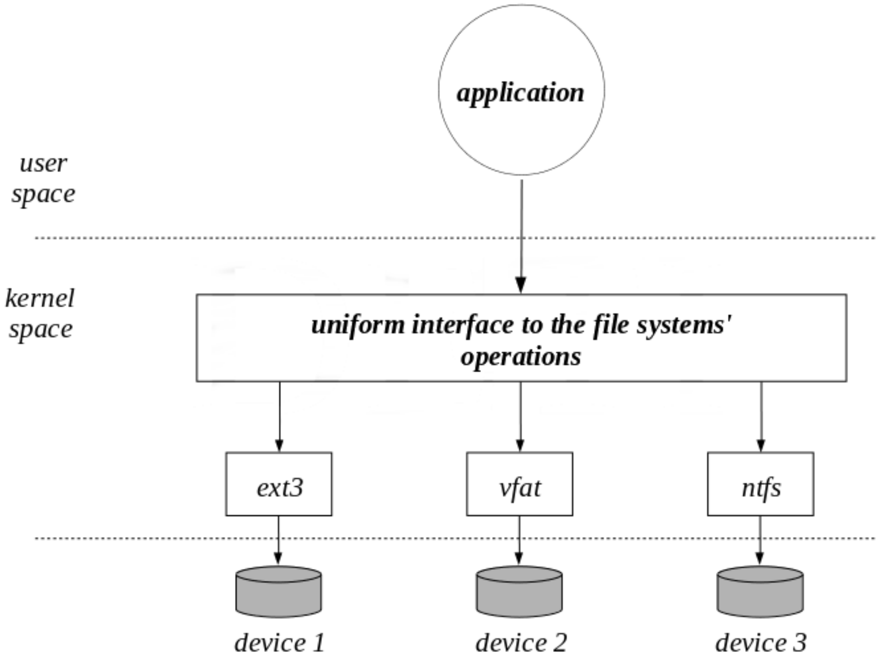
File system
Role of operating system

- A role of the operating system is to implement the file concept, providing a set of operations (system calls) which establishes a simple and secure communication interface for accessing the mass storage contents

- The file system is the part of the operating system dedicated to this task
- Different implementations of the file data type lead to different types of file systems
  - Ex: ext3, FAT, NTFS, APFS, ISO 9660, ...

- Nowadays, a single operating system implements different types of file systems, associated with different physical devices, or even with the same
  - This feature facilitates interoperability, establishing a common means of information sharing among heterogeneous computational systems

File system
Virtual file system

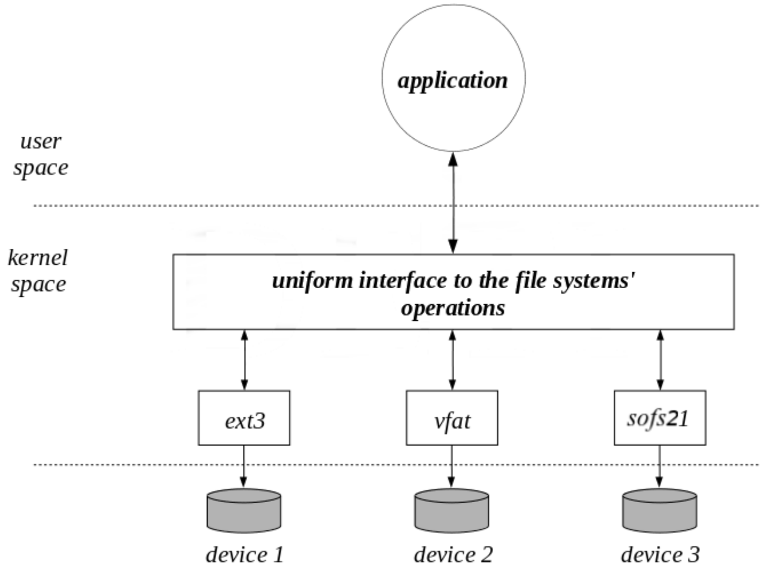


File system
Typical file operations on Unix

- As referred to before, the operations are based on system calls
- system calls on regular files
  - creat, open, close, link, unlink, read, write, truncate, lseek, ...
- system calls on directories
  - creat, open, close, mkdir, rmdir, getdents, ...
- system calls on symbolic links
  - readlink, symlink, ...
- system calls common to any type of file
  - mknod, chmod, chown, stat, utimes, ...

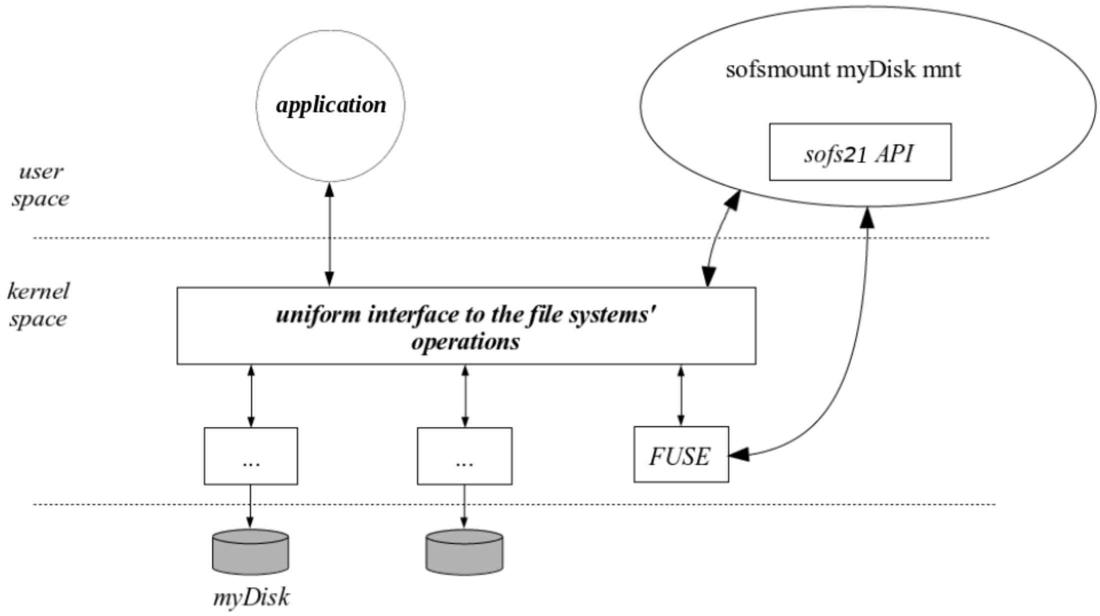
- On a terminal execute `man 2 ≪syscall≫` to see a description

File system
File system as a kernel module



- Safety issue: running in kernel space
  - Malicious or erroneous code can damage the system

File system
File system as a FUSE module

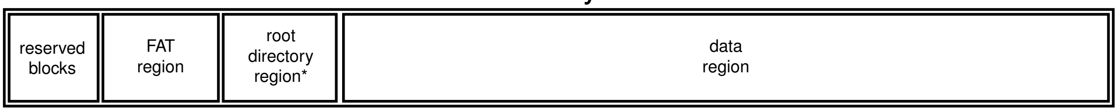


- Safe: running in user space
  - Malicious or erroneous code only affects the user

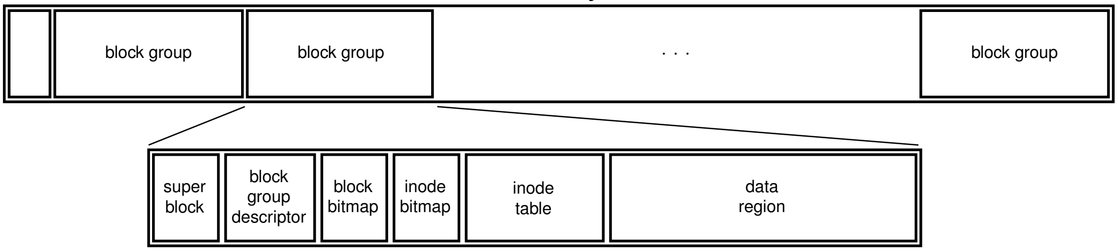
File system
How to implement it?

- An implementation issue is how to organize the device storage space, seen as an array of blocks, as to obtain the desired abstract view
- Different file systems are related to different internal architectures

FAT* file systems



ext2 file system



Data blocks
Some points

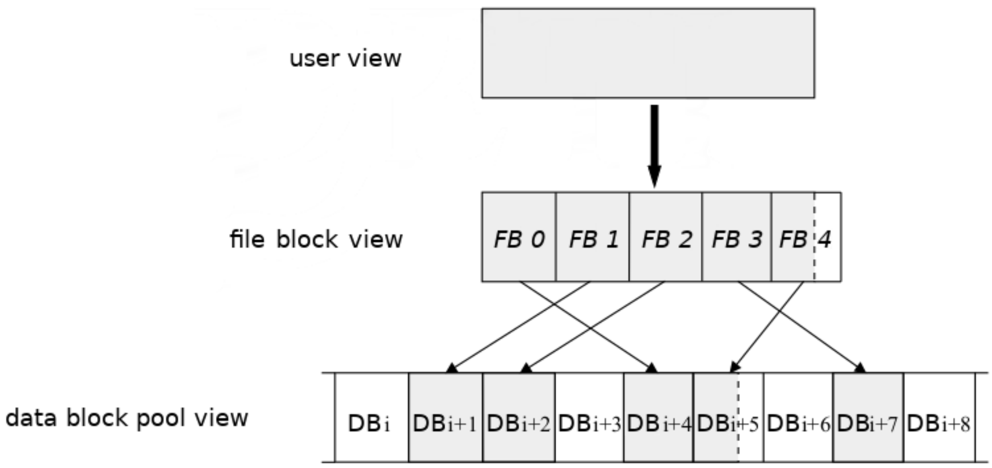
- The block (cluster in Windows) is the unit of allocation for file contents
  - A block can be a single disk sector (the disk storage unit) or a contiguous sequence of sectors, usually in powers of 2
- Blocks are not shareable among files
  - in general, an in-use block belongs to a single file
- The number of blocks required by a file to store its information is given by

$$N_b = \text{roundup}\left(\frac{\text{size}}{\texttt{BlockSize}}\right)$$

  - N_b can be very big – if block size is 1024 bytes, a 2 GByte file needs 2 MBlocks
  - N_b can be very small – a 0-bytes file needs no blocks for data
- It is impractical that all the blocks used by a file are contiguous in disk
- Also, the access to the file data is in general not sequential, but random instead
- So a flexible data structure, both in size and location, is required

Data blocks
File content views

- The programmer view: a file is as a continuum of bytes
- The file block view: a file is as a sequence of blocks
- The data block view: in general, a file is scattered along the data block region



Data blocks
Sequence of blocks of a file: the FAT file system approach

- How is the sequence of (references to) data blocks stored in a FAT file system?
- The first reference is directly stored in the directory entry
- Then, the file allocation table (FAT) allow to identify the remaining block references
  - The FAT is an array of references, stored in a fixed part of the disk
  - Each entry can be 12- 16- or 32-bits long

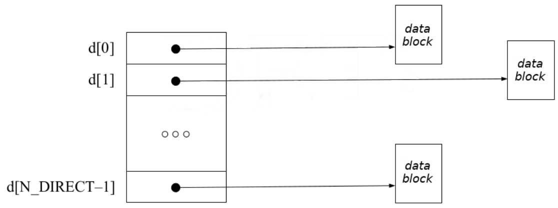
																. . .	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		N-1

- Assuming a FAT16 file system and that the sequence holding the data of a file is $2, 5, 6, 7, 10, 3$, the contents of the FAT, in what is related to that file, is

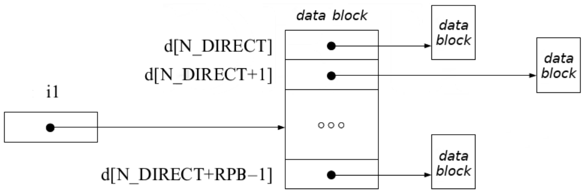
	FFFF	5	FFFF		6	7	10			3						. . .	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		N-1

Data blocks
Sequence of blocks of a file: the sofs21 file system approach (1)

- How is the sequence of (references to) data blocks stored in an sofs21 file system?
- The first references are directly stored in the file's inode
  - An inode is a record containing the metadata of a file

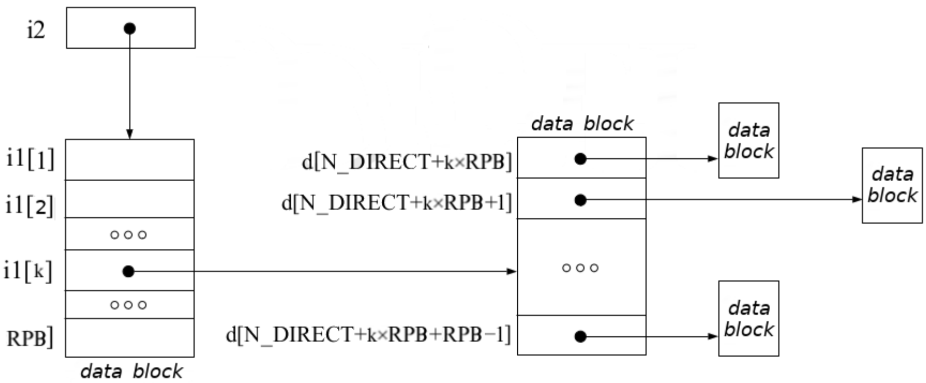


- Then, inode field `i1` points to a data block with references



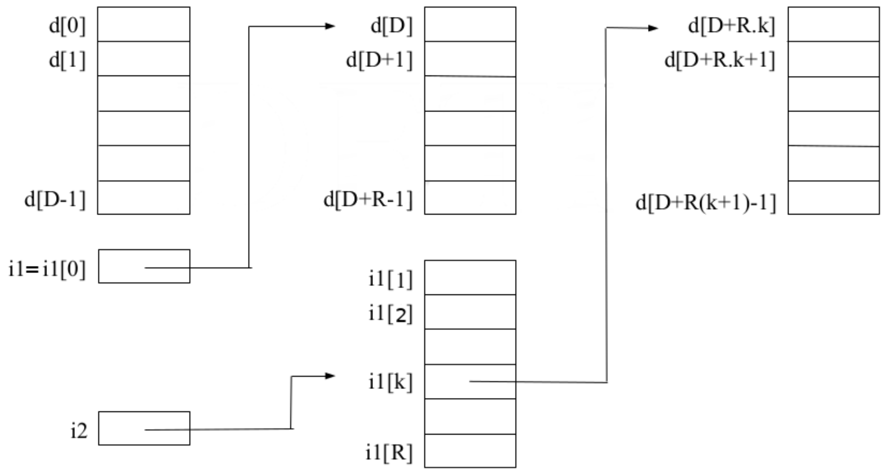
Data blocks
Sequence of blocks of a file: the sofs21 file system approach (2)

- Finally, inode field `i2` points to a data block that extends `i1`



Data blocks
Sequence of blocks of a file: the sofs21 file system approach (3)

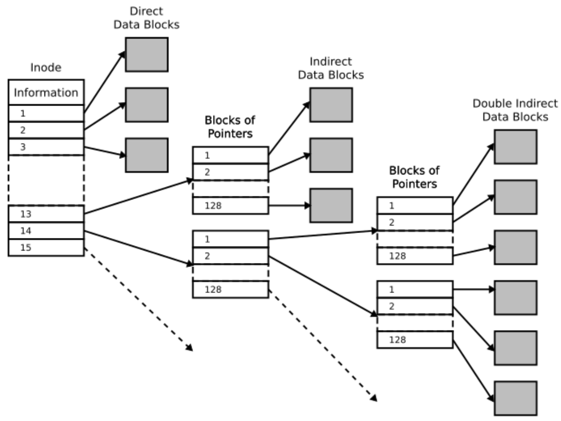
- Puting all together



- A file can contain “holes”, corresponding to null references covered by the size and representing streams of zeros

Data blocks
Sequence of blocks of a file: the ext2 file system approach

- How is the sequence of (references to) data blocks stored in an ext2 file system?
- The approach is the same as before with an additional triple indirect pointer
  - There are 12 direct pointer, 1 indirect, 1 doubly indirect and 1 trebly indirect



source: https://en.wikipedia.org/wiki/Ext2

- A file can contain “holes”, corresponding to null references covered by the size and representing streams of zeros

Data blocks
Maximum size and access cost

- What is the maximum size of a file in a FAT12 file system?
  - Considering a cluster of 1, 4 and 16 sectores, being a sector 512 bytes long
- What is the maximum size of a file in a FAT32 file system?
  - Considering a cluster of 1, 4 and 16 sectores, being a sector 512 bytes long
- What is the maximum size of a file in an ext2 file system?
  - Considering a block of 2, 4 and 16 sectores, being a sector 512 bytes long
- What is the maximum cost to localize a file cluster in a FAT32 file system?
  - Considering a 1 GiB file and a 8 KiB cluster
- What is the maximum cost to localize a file block in an ext2 file system?
  - Considering a 1 GiB file and a 8 KiB block

Data blocks
List of free data blocks

- One important issue relating to the data blocks of a disk is knowing which are free at a given moment
  - If a file grows requiring a new data block, what free data block should be allocated?
- In the FAT file system:
  - A FAT entry with the value `0` represents a free cluster
  - Allocating a new data block requires search the FAT looking for an entry with that value
- In the ext2 file system:
  - There is a section in a block group containing a bitmap of free/allocated data blocks within the group
  - Allocating a new data block requires searching the bitmap looking for a bit at 0

Inodes
What is an inode?

- In Unix, the inode (identification node) plays a central role in the implementation of the file data type
  - An inode is typically identified by an integer number

- It corresponds to the identity card of a file and contains:
  - file type
  - owner information
  - file access permissions
  - access times
  - file size (in bytes and blocks)
  - sequence of disk bocks with the file contents
- The name/path is not in the inode – it is in the directory entry

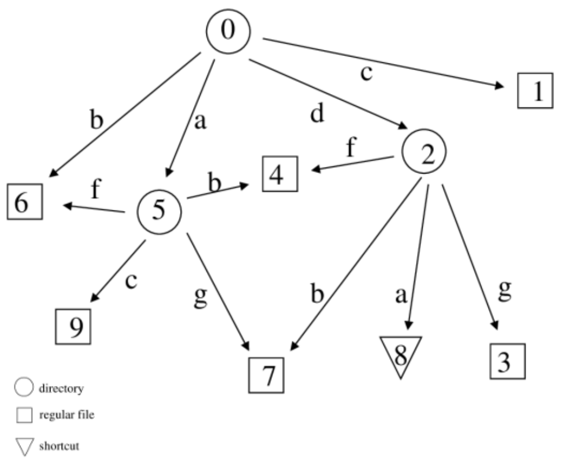
- In an ext2 file system, in every block group, there is a region reserved for inodes, the inode table
- There is also an inode bitmap representing the free/allocated inodes

---

- disk inodes vs. in-core inodes

Inodes
Hierarchy of files in Unix/Linux file systems

- Every file uses one and only one inode
- Same inode may have different pathnames
- Hierarchy of files may not be a tree



- The contents of a disk can be seen as a graph, where
  - Nodes are the files (directories, regular files, shortcuts, . . .), each one having an associated inode
  - Arrows define the hierarchy
- Paths `/a/b` and `/d/f` refer to the same regular file (inode 4)
- Paths `/a/g` and `/d/b` refer to the same regular file (inode 7)
- Paths `/b` and `/a/f` refer to the same regular file (inode 6)

Directory implementation
in the FAT file system

- A directory is a set of directory entries , these being key-value pairs that directly associate file names to file attributes

	name	attributes
ent[0]		
ent[1]		
ent[2]		
. . .		
ent[n]		

- All entries have the same size – 32-bytes long
- Normal name field is composed of 8 plus 3 characters, referred to as the name and the extension
  - Long file name are supported using a trick
- Does not exist an ownership field
- One of the fields indicate the first cluster

Directory implementation
in the ext2 file system

- A directory is a set of directory entries , these being key-value pairs that directly associate file names to inode numbers

	name	inode number
ent[0]		
ent[1]		
ent[2]		
. . .		
ent[n]		

- Size of the entries depends on the name length
  - Does not exist the notion of extension
- Different directory entries can point to the same inode
  - This is referred to as hard links
- The file attributes are in the inode
  - Do exist an ownership field

File system
The FAT file system

- The data region is divided into portions of contiguous blocks called clusters
- Each file may occupy one or more clusters depending on its size
  - Those clusters may be no contiguous, so a file is represented by a chain of clusters
  - A cluster is only used by a single file
- The chain of clusters is implemented using the file allocation table (FAT), stored in the FAT region
  - The FAT region contains two copies of the FAT, for recovery purposes
- A FAT corresponds to an array of indices, with an entry per cluster
  - There can exist more entries in the FAT than clusters, since the FAT must occupy an integer number of blocks (sectors)
  - The number of bits used for each entry defines the FAT model (12, 16, 32)
  - The entry values are stored in little-endian format

File system
Diving into a FAT32 file system

- A file named ”xxx“ was created at the root of a FAT32 filesystem, whose contents is ”SO/FSO“.
  - In what sector is name ”xxx“ stored?
  - In what cluster/sectors is its contents stored?
- Questions to answer:
  - First sector of the root
  - First sector of the data region
  - Number of sectors of a cluster
  - ...

File system

Diving into a FAT32 file system: the 1st sector

- Sector size in bytes (offset: 11; size: 2): 512
- Cluster size in sectors (offset: 13; size: 1): 8
- Reserved area size in sectors? (offset: 14; size: 2): 32
- Number of FATs (offset: 16; size: 1): 2
- Volume size in sectors (offset: 32; size: 4): $N_T = 7801336$
- FAT size in sectors (offset: 36; size: 4): $N_F = \frac{N_T - 28}{2 + 2^{10}} = 7603.6 \Rightarrow 7604$
  - FAT[0] and FAT[1] are reserved
  - 4 upper bits of any entry are reserved and must be preserved
- First cluster of root in data region (offset: 44; size: 4): 2
- First sector of root in volume (offset:): $32 + 2 * 7604 = 15240$