Chapitre 3 - Le système de fichiers

2.0

Système d'exploitation GNU/Linux

1 Avril 2022

Table des matières

Objectifs

A l'issue de ce chapitre, l'apprenant doit être capable de :

1. distinguer les **types de mémoires de masses** et leurs **caractéristiques ;**
2. distinguer les concepts de **formatage physique** de disque, de **partitionnement** et de **formatage logique**;
3. distinguer les principes de fonctionnement du **système de fichier Extensible File Système -ExtFS** ;
4. distinguer les principes de fonctionnement du **système de fichier virtuel** (**Virtual File System - VFS**) **de Linux** ;
5. effectuer les **opérations** mise à disposition par le **système de fichiers virtuel**.

Introduction

Ce chapitre est relatif à la gestion des **mémoires de masse** (disque dur, clé usb, cd, dvd,...) par le système d'exploitation. Les mémoires de masse sont utilisées pour **stocker de manière permanente des informations sous forme de fichiers**. Tous ces fichiers ne sont en réalité qu'une **suite de bit** que le système d'exploitation confie à la mémoire de masse (à une partition en réalité) à travers le pilote du matériel.

La **gestion des fichiers** sur les **mémoires de masse** est une des **fonctions du système d'exploitation**. Elle est confiée à un module appelé **système de gestion de fichiers** ou plus simplement **système de fichiers** qui est indiquée sur la partition (on parlera de formatage logique). Les systèmes d'exploitation GNU/Linux utilisent un système de fichier appelé **extended file system - EXT**. Un système GNU/Linux ne peut être installé que sur une partition formatée en EXT.

Toutefois, **GNU/Linux** prend en charge beaucoup d'autres systèmes de fichiers (développés par d'autres systèmes d'exploitation), Il peut exploiter une partition qui a été formatée avec un autre système de fichier comme FAT , NTFS, Minix, ...

A la différence des systèmes d'exploitation qui donnent un accès aux différents systèmes de fichiers par des identifiants de périphériques (numéro ou nom de lecteur, cas de windows), les systèmes UNIX offrent un accès unifié et transparent aux différents systèmes de fichiers par une **arborescence à entrée unique** à l'aide d'un **système de fichier virtuel**.

Nous aborderons ainsi dans ce chapitre les points suivants :

1. les mémoires de masse
   - définitions, caractéristiques, formatage physique, partitionnement ;
2. Les systèmes de fichiers
   - formatage logique, exemples de systèmes de fichiers ;
3. Le système de fichiers EXT
   - concepts de base, structure du SF, utilitaires
4. le système de fichier virtuel de Linux - VFS
   - fichiers et arborescence, fonctionnalités , architecture et structure.

I.Les mémoires de masse

1. Définition et caractéristiques

Définition

Une **mémoire de masse** désigne un **support de stockage** de l'information qui est :

1. **non volatile** : les informations sont conservées même en l'absence d'alimentation électrique
2. accessible en **lecture et écriture**.
3. et de "**grande capacité**" (ce critère est relatif)

Caractéristiques des mémoires de masse

Citons quelques **caractéristiques importantes** des mémoires de masse :

- Capacité de stockage
- Temps d'accès aux informations
- Débit de transfert des informations
- Durée de vie des informations stockées.
- Coûts

2. Types de mémoires de masse

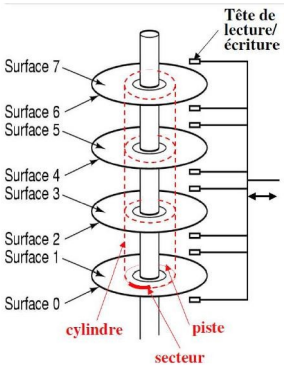
Nous nous limiterons à citer les **trois technologies** les plus présentes dans le marché:

1. le **disque dur** traditionnel à **support magnétique** toujours utilisé dans les ordinateurs, caméscopes, consoles de jeux...
2. les supports basés sur la **mémoire flash** (clé usb, carte mémoire, disque dur SSD).
3. les **disques optiques** (CD, DVD, Blu-ray)

2.1. Disque Dur

Structure du disque Dur

Le disque dur est composé de **plusieurs plateaux** empilés les uns sur les autres et constitués de **pistes alignées** de manière à former des **cylindres**. La **lecture et l'écriture** se fait grâce à des **têtes de lecture** situées de part et d'autre de chacun des plateaux.



Disque dur

Contrôleur de disque

Le **disque dur** est **relié à la carte-mère** par l'intermédiaire d'un **contrôleur de disque** dur faisant l'**interface** entre le processeur et le disque dur :

1. **IDE** (Integrated Drive Electronics) qui permettait de connecter deux disques dur avec un transfert d'environ 10 à 133 Mo/s a aujourd'hui presque disparu
2. **SATA (Serial Advanced Technology Attachment)** qui permet un transfert de 35Mo/s à ses débuts et en sa version 3 jusqu'à 6 GO/s environ est la technologie la plus utilisée actuellement

Connecteur IDE

Connecteur SATA

2.2. La mémoire flash

La **mémoire flash** est une **mémoire de masse** basée sur la technologie EEProm (Electrically Erasabe, Programmable, Read-Only Memory). L'élément de base de la **mémoire flash** est un **transistor** (MOSFET : Metal Oxyde Semiconductor Field Effect Transistor) lui permettant de **stocker les bits de données**. Chaque transistor possède une « **grille** » qui permet d'**enregistrer** ou de modifier les données en **piégeant les électrons** dans cette grille via l'application de différentes tensions aux points d'entrée. Une mémoire flash standard peut supporter aujourd'hui jusqu'à **100 000 écritures et effacements** au cours de sa durée de vie.

Elle est **résistante aux chocs** car ne possédant aucun élément mécanique. Elle est **beaucoup plus rapide que le disque dur classique**.

Elle est encore **cher**, le site de Avast estime le prix d'un disque dur classique HDD 500 GO entre 23 et 50 € et un disque dur SSD 500GO entre 55 et 140€.

Types de mémoire flash

Nous rencontrons **deux types de mémoire flash** :

1. la **flash NOR** principalement utilisé pour le stockage des **systèmes d'exploitation** dans les **téléphones portables**, les **décodeurs TV** et les **appareils photo**;
1. la **flash NAND** adapté au stockage de masse de données, c'est lui qui est utilisé pour les **cartes mémoires SD** (Secure Digital) et **MS** (Memory Stick) ainsi que pour les **disques SSD** et les **clés USB**.

Matériel basé sur la mémoire flash

Parmi les supports de stockage basés sur la mémoire flash, nous pouvons citer :

- les clé USB (Universal Serial Bus) ;
- les cartes mémoires ;
- les disques durs SSD (solid-state drive).

2.3. Disques optiques

Un **disque optique** est un support de **stockage de masse amovible** constitué de **polycarbonate**. L'information est lue et enregistrée par l'intermédiaire d'un **rayon laser**.

C'est un support qui a énormément évolué. Si les premiers CD avaient un contenu fixé à la fabrication, du son (**CD Audio**), les générations suivantes pouvaient conserver des données et sont devenus réinscriptibles (**CD-RW, DVD-RW**). La dernière génération est le disque **blu-ray** qui peut contenir plus de vingt six (26) heures de Video en simple définition.

3. Formatage physique de disque

Certains matériels de stockage comme le disque subissent généralement ce qu'on appelle un **formatage de bas niveau** ou **formatage physique** en **usine**. Ce formatage permet d'initialiser/réinitialiser le disque dur par une vérification du disque (zones défectueuses) et l'**organisation de sa surface** en **éléments basiques**.

Formatage physique d'un disque dur

Le **formatage physique** d'un **disque dur traditionnel** à support magnétique consiste en l'**inscription de secteurs** sur les pistes. Historiquement, les **secteurs** étaient de **512 octets**. Les constructeurs sont maintenant passés au **format avancé (advanced format)** avec des **secteurs** de **4 ko** combinaison de 8 anciens secteurs de 512o. Les sites de certains constructeurs comme seagate offrent une bonne documentation sur cette question [1].

Les **systèmes informatiques** ont beaucoup d'aspects qui reposent sur la taille de secteur de **512O**. La **transition** s'est fait à travers une **émulation** de secteurs de 512 par les secteurs de 4KO. Cela est indiqué par le sigle **512e** : une taille de **secteur physique de 4 KO** et de **secteur logique de 512 octets**. On trouve aujourd'hui dans le marché des disques **4K natifs** sans émulation indiqués par un logo, ils sont bien pris en compte par les systèmes d'exploitations. Le noyau Linux les prend en compte depuis la version 2.6.31.

Logo advanced format

Formatage physique des mémoires flash (clé USB, carte mémoire, disque dur SSD)

Le formatage physique des mémoire flash consiste en la vérification du matériel (**cellules défectueuses**) et à la "**mise à zéro**" de toutes les cellules.

Attention : Formatage en usine

Le formatage physique est effectué en usine sur les matériels de stockage. Certains outil permettent de l'effectuer sur votre propre matériel en cas de dysfonctionnement.

4. Partitionnement de disque

Le **partitionnement** d'un disque est le **fractionnement** du disque en **plusieurs parties** destinée chacune à accueillir un **système de fichiers**. Le partitionnement d'un disque est obligatoire, il doit contenir au moins une partition.

Il se fait **après le formatage physique** et **avant le formatage logique**. Cela sert par exemple à installer des **systèmes d'exploitation différents** n'utilisant pas le même système de fichiers ou à créer des zones sur le disque dont les **données ne seront pas mélangées**. Un disque peut ainsi contenir une ou plusieurs partitions.

1 - https://www.seagate.com/fr/fr/tech-insights/advanced-format-4k-sector-hard-drives-master-ti/

Table de partitionnement

Les partitions d'un disque sont inscrites dans la **table de partitionnement** du disque qui selon la technologie est soit sur le **Master Boot Record - MBR** ou sur le **GPT (GUID Partition Table)**.

BIOS et Master Boot Record - MBR

En réalité, le premier programme exécuté par l'ordinateur au démarrage est le **BIOS (Basic Input Output System)** qui est situé sur la **mémoire morte** de la carte mère. Il permet d'**initialiser le matériel** (détection du clavier, de l'écran et du disque dur,...)et c'est lui qui **lance le système d'exploitation** à partir du premier secteur de 512 octet du disque dur appelé **secteur d'amorçage** ou **secteur 0** et qui contient le **Master Boot Record** (**MBR**).

Cette technologie présentait certaines **limites** , par exemple sur la **taille maximale des disques durs** de **2.2 TO** et sur la **limitation de partitionnement** du disque à **4 partitions physiques**.

UEFI et Guid Partition Table - GPT

L'**UEFI** (Unified Extensible Firmware Interface) est un nouveau standard qui **remplace le BIOS**. Il prend en charge les disques durs de **plus de 2.2 TO** et recommande l'utilisation d'une nouvelle table de partitionnement : le **Guid Partition Table** qui supporte jusqu'à **128 partitions**.

Commandes de partitionnement

Le **partitionnement** d'un disque peut être effectué par l'utilisateur par les commandes d'un système d'exploitation ou un logiciel comme *GParted*[2]. Sur GNU/Linux, la commande **fdisk** permet de manipuler la table de partition d'un **disque MBR**, la commande **gdisk** permet de manipuler une table de partitionnement **GPT**.

Partitions obligatoires et/ou recommandées

Certains systèmes d'exploitation imposent plusieurs partitions pour leur installation. GNU/Linux nécessite au moins deux (2) partitions pour son installation :

1. Une **partition principale** formatée en **EXT** et **montée** sur le répertoire **/**
2. une **partition swap** (**partition d'échange**) qui est une zone du disque dur utilisée comme mémoire virtuelle pour décharger la RAM. Il est conseillé de lui donner le double de la taille de la RAM. Les commandes **mkswap** et **swapon/swapoff** permettent respectivement de formater et d'activer/désactiver une partition de swap.

Il est conseillé selon l'usage qui sera fait d'un système GNU/Linux d'avoir plusieurs partitions : par exemple une pour les données utilisateurs, une pour les données de l'administrateur, une pour les fichiers de configurations, ...

2 - https://gparted.org

II.Généralités sur les systèmes de fichiers

Système de fichiers

Un **système de fichier** ou système de gestion de fichier désigne la façon de stocker et d'organiser les données sur un support de stockage (en réalité une partition). Avant de pouvoir **utiliser un support de stockage** sorti de l'usine, il est nécessaire de le **partitionner** et ensuite d'**inscrire un système de fichier** sur les partitions que vous souhaitez utiliser à travers le **formatage logique**.

Nous allons aborder dans cette section le partitionnement, le formatage logique et revenir sur quelques exemples de systèmes de fichiers.

1. Formatage Logique et systèmes de fichiers

Définitions

Le **formatage logique** est définit comme l'**inscription d'un système de fichiers** sur une **partition de disque**. Toute **partition**, avant d'être utilisée doit donc être **formatée** avec un **système de fichier**.

Un **système de fichier** ou **système de gestion de fichiers** désigne ainsi la façon de stocker et d'organiser les données sur un support de stockage (en réalité une partition). Il est **inscrit** sur la **partition** par le processus de **formatage**.

Systèmes d'exploitation et systèmes de fichiers

Le formatage logique d'une partition d'un système de stockage peut être effectué par l'utilisateur avec les commandes et outils du système d'exploitation ou avec un logiciel tiers.

Certains systèmes d'exploitation ne propose le **formatage** que dans les **systèmes de fichiers qu'ils ont conçu**. A la différence de ces systèmes d'exploitations, **GNU/Linux** permet de **formater** une partition aussi bien en **EXT** que dans d'**autres systèmes de fichiers** qui n'ont pas été conçu dans le monde UNIX comme **NTFS** et **FAT**.

Certains systèmes d'exploitation ne peuvent **exploiter** que des **partitions formatées** avec les **systèmes de fichiers qu'ils ont conçu**. **GNU/Linux** est un système qui permet d'exploiter des **partitions formatées** sur des **systèmes de fichiers autre que EXT**. Il prend en charge aujourd'hui plus d'une quinzaine de formats de systèmes de fichiers.

Commandes Linux

Plusieurs commandes permettent de **connaître le système de fichier** d'une partition :

- **lsblk -f** qui affiche les périphériques blocs.
- **df -hT** qui indique l'espace occupé par les systèmes de fichiers
- ...

Les commandes **mkfs.type** (**type** désigne le système de fichier comme ntfs, ext, vfat,) permettent de **formater** une **partition** sur GNU/Linux avec le système de fichier indiqué par **type**.

Exemple : Exemples de commandes de formatage

- **mkfs.ext4** crée un système de fichier ext4
- **mkfs.ntfs** formate une partition en ntfs
- **mkfs.vfat** formate en vfat

Remarque : Formatage et effacement des données

Dans la croyance populaire, il est retenu que le formatage est un effacement des données de la partition. En réalité, l'effacement des données est une conséquence du formatage.

Lorsque nous formatons une partition, le système n'efface pas forcément les données (fichiers) qui sont dessus, Il indique à la partition qu'il va être organisé/réorganisé selon le système de fichiers choisi et qu'il est donc considéré comme vierge.

caractéristiques d'un système de fichiers

Les attributs suivants sont à prendre en compte sur les systèmes de fichiers :

1. **taille maximale d'une partition** sur laquelle on peut inscrire le système de fichier ;
2. **taille maximale d'un fichier** dans le système de fichier ;
3. **la gestion des droits d'accès** aux fichiers(sécurité et confidentialité) ;
4. la **journalisation** qui offre plus de fiabilité.

Choix d'un système de fichier au formatage

Le **choix d'un système de fichier** au **formatage** est déterminé d'abord par le **futur usage de la partition** tout en prenant en compte les caractéristiques. Prenons quelques exemples :

- Une partition sur laquelle nous souhaitons installer un système Windows (Windows en utilise actuellement plusieurs) est à formater en NTFS.
- Une partition sur laquelle nous souhaitons installer un système GNU/Linux est à formater en EXT (4).
- Une partition de données seulement utilisée par Windows peut être formatée en NTFS ou FAT (moins sécurisé, moins fiable)
- une partition de données partagée entre Windows et GNU/Linux peut être formaté en NTFS ou en FAT.
- Un support amovible est souvent formaté en FAT
- ...

2. Quelques exemples de systèmes de fichiers

Exemple : EXT de GNU/Linux

Extended File System - Ext est le **système de fichiers natif de Linux**. En ses versions 1 et 2, il ne dispose pas de la journalisation. **ext3** ajoute la **journalisation** à Ext2. **ext4** est un ensemble de mises à niveau d'ext3, y compris des améliorations substantielles en matière de **performances** et de **fiabilité**, ainsi que d'importantes **augmentations des limites de volume**, et de **taille de fichier**.

Exemple : FAT et NTFS de Ms Windows

Ms Windows propose **deux grandes familles** de systèmes de fichiers :

1. **File Allocation Table - FAT** est un **système de fichier** développé par **Microsoft**, Il désigne **FAT16**, né en 1984, qui utilise des adresses d'**unités d'allocation sur le disque** codées sur **16 bits** (2^{16} possibilités) mais aussi son prédécesseur **FAT12**. Il se rencontre moins fréquemment aujourd'hui, il est surtout utilisé pour les disquettes 3½ (3 pouces 1/2) et les anciennes clés usb formatées sous Windows .
   Son successeur **FAT32**, né en 1996 sur Windows 95 OSR2, utilise des adresses sur **28 bits**. Il est utilisé par plusieurs constructeurs comme système de fichiers pour cartes mémoires (memory sticks) et clé USB , car, bien documenté, ce système de fichiers reste **le plus universellement utilisé et accessible**, il est compatible avec la plupart des systèmes d'exploitation. **VFAT** est une extension de FAT qui ajoute la possibilité d'utiliser des **noms de fichiers longs**.
   et il a un successeur **exFAT** qui augmente la **limite de taille de partition** et de **taille de fichier** en utilisant des adresses 64 bits (certains l'appelle **FAT64**) et dont **les spécifications ont été rendu publiques par Microsoft** . C'est le format de système de fichier utilisé aujourd'hui sur les clés USB, cartes mémoires, disque SSD,...
2. **New Technology File System - NTFS** , né avec Windows NT3.1, est un système de fichiers développé par Microsoft, il est considéré comme très peu documenté mais

améliore FAT sur la sécurité avec la **gestion des droits** et aussi la fiabilité avec la **journalisation**. L'**écriture depuis Linux** sur ce système de fichiers est stable à l'aide du pilote **ntfs-3g**. Ce pilote est inclus de base dans Ubuntu, et disponible en paquets dans les dépôts pour les versions antérieures.

Exemple : le format UDF

Universal Disk Format - UDF est un format **ouvert**, **normalisé** et **indépendant du SE** . Il est utilisé pour les **lecteurs optique (CD, DVD)** et peut être utilisé sur n'importe quel autre support de disque. Il est maintenu par l'*Optical Storage Technology Association*[3].

Complément : Comparatif de quelques systèmes de fichiers

Système de fichiers	taille max de fichier	taille max de partition	journalisation	droits d'accès
ext2FS (Extended File System)	2 TiO	4 TiO	Non	Oui
ext3FS	2 TiO	4 TiO	Oui	Oui
ext4FS	16 TiO	1 EiO	Oui	Oui
FAT	2 GiO	2GiO	Non	Non
FAT32	4 GiB	8 TiO	Non	Non
exFAT	128 PiO	128 PiO	Non	Non
NTFS	16 TiO	256 TiO	oui	oui
UDF	16 EiO	2TO	Non	Oui

EiB = Exbioctets (1024 pébioctets) :: PiB = Pébioctet (1024 tébioctet) :: TiB = Tébioctet (1024 gibioctets) :: GiB = Gibioctet (1024 mibioctets) :: MiB = Mibioctet (1024 kibioctets) :: KiB = Kibioctet (1024 octets).

3 - http://www.osta.org/

III.Extended File System, le système de fichier de Linux

Au **formatage**, le **système d'exploitation** (ou le logiciel de formatage) inscrit le **système de fichier choisi** sur la **partition de disque** ciblée. Le **système de fichier par défaut** du noyau **Linux** est **Extensible File System - ExtFS** qui est à sa version 4 aujourd'hui.

Dans ce cours nous traiterons du système EXT2 qui est considéré comme le système de fichiers le plus largement répandu dans la communauté Linux. Ext3 lui apporte la journalisation et ext4 des améliorations sur la performance ...

Dans leur *article de référence*[4], les concepteurs du **second système de fichiers étendu** Ext2 rappellent qu'il fut conçu et implémenté pour résoudre certains problèmes présents dans le premier système de fichiers étendu Ext.

C'est un **système de fichier puissant** qui utilise la **sémantique des fichiers Unix** et présente d'**excellentes performances**. Il est **robuste** au sens ou **les pertes de données sont réduites** lors d'usage intensifs. Ext2FS a dû prévoir de la place pour les **extensions**, permettant ainsi aux utilisateurs de bénéficier de **nouvelles fonctionnalités** sans avoir à reformater leur système de fichiers.

Nous reviendrons dans cette section sur l'**historique** du système de fichier EXT2, sur les **concepts de base** de EXT2, sur la **structure** du système de fichier et enfin sur les **utilitaires**.

1. Historique

Le système de fichier Minix

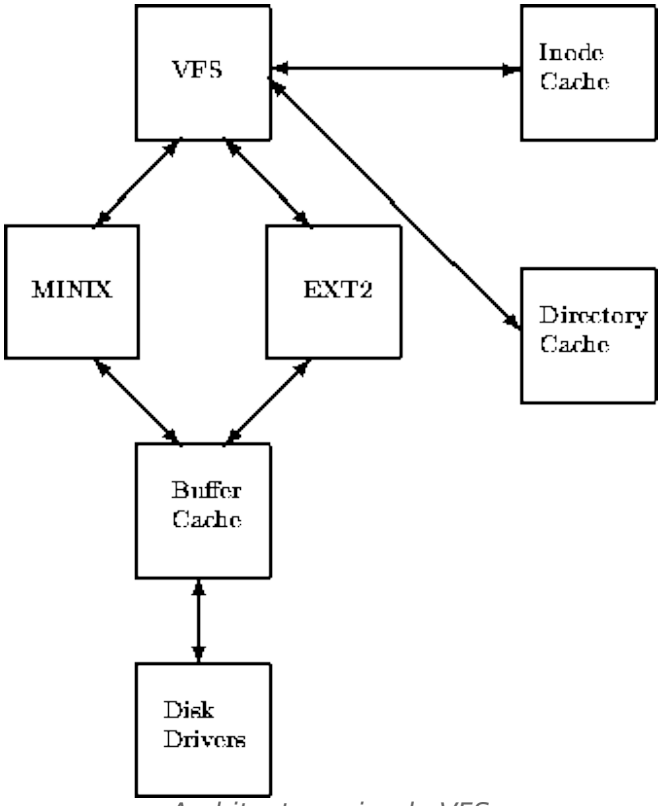
Au début du système d'exploitation Linux, Linus Torvald a décidé d'implémenter le support du **système de fichiers Minix dans Linux**. C'était un système de fichier **efficace** et relativement **exempt de bogues**.

Avec les **restrictions** dans la conception du **système de fichier Minix** telle que la **taille maximale de partition à 64 MO** et la taille des **noms de fichiers de 16/30 caractères**, la réflexion était né sur l'implémentation d'un nouveau système de fichiers dans Linux.

La naissance du système de fichier Virtuel

Afin de **faciliter l'ajout de nouveaux systèmes de fichiers** dans le noyau Linux, une **couche VFS (Virtual File System)** a d'abord été développée. La couche **VFS** a été initialement écrite par Chris Provenzano, puis réécrite par Linus Torvalds avant d'être intégrée au noyau Linux. Elle sera l'interface entre l'utilisateur et les différents systèmes de fichiers. Cela est illustré dans le schéma suivant.

4 - http://e2fsprogs.sourceforge.net/ext2intro.html



Architecture simple VFS

Le système de fichier Ext

Après l'intégration du VFS dans le noyau, un nouveau système de fichiers, appelé **système de fichiers étendu - Ext**, a été implémenté en **avril 1992** et ajouté à **Linux 0.96c**.

Ext dans sa première version ne prenait pas en charge l'**horodatage séparé d'accès, de modification d'inode et de modification de données**. Le système de fichiers utilisait des listes liées pour garder la trace des blocs libres et des inodes, ce qui produisait de mauvaises performances : au fur et à mesure de l'utilisation du système de fichiers, les listes devenaient non triées et le système de fichiers se fragmentait.

Le système de fichier Ext2

EXT2 est basé sur le code de Ext avec une **forte réorganisation du code** et de nombreuses **améliorations**. Il a été conçu dans une optique d'évolution et prend en compte l'intégration de futures améliorations. Si a ses débuts EXT2 présentait quelques bogues, ils ont été rapidement corrigés et il est devenu **très stable** et de facto le **système de fichiers standard de Linux**.

L'évolution dans les caractéristiques des trois systèmes de fichiers est présentée dans le tableau suivant :

	Minix FS	EXt FS	Ext2 FS
taille max de partition	64 MO	2 GO	4TO
taille max de fichier	64 MO	2 GO	2 GO
longueur de nom de fichier	16/30 c	255 c	255c
support 3	Non	Non	Oui

	Minix FS	EXt FS	Ext2 FS
horodatages			
Extensible	Non	Non	Oui
taille de bloc variable	Non	Non	Oui

2. Concepts de base

Le **système de fichiers Linux Ext2** met en œuvre un ensemble de concepts de base communs dérivés du système d'exploitation **Unix** :

1. les **fichiers** sont représentés par des **nœuds d'index** ou **inodes**, nous rappelons qu'un **index** est une **structure** chargée d'**ordonner** et de **trier** des objets afin de pouvoir les **retrouver** plus rapidement (recherche) ;
2. les **répertoires sont des fichiers** contenant une **liste d'entrées (inode, nom de fichier, taille)**
3. les **périphériques** sont représentés par des **fichiers spéciaux** pour les Entrées/Sorties.

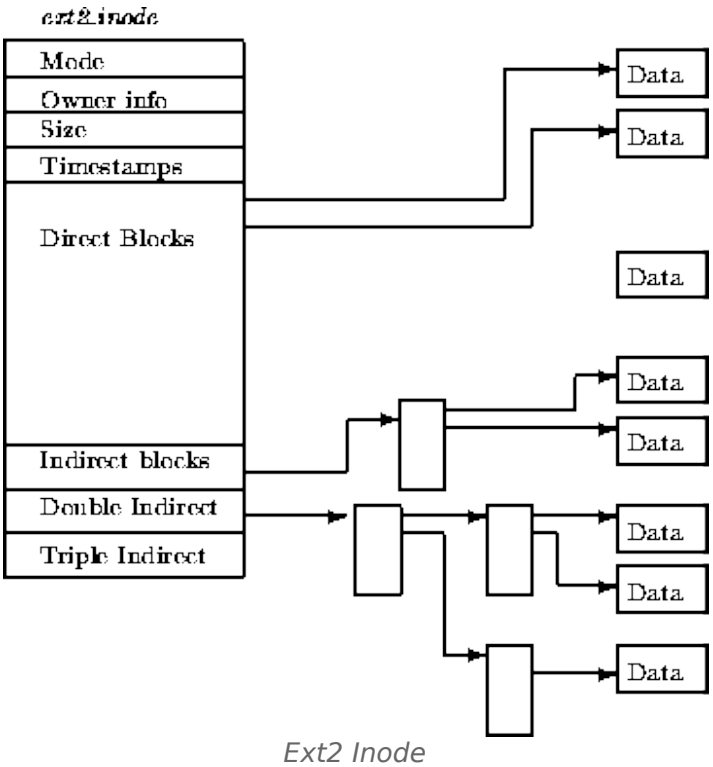
C'est ce qui est exprimé par : **"Dans un système Unix, tout est fichier".**

Nœuds d'index (Inodes)

Chaque **fichier** est représenté par une **structure**, appelée **nœud d'index** ou plus communément **inode**.

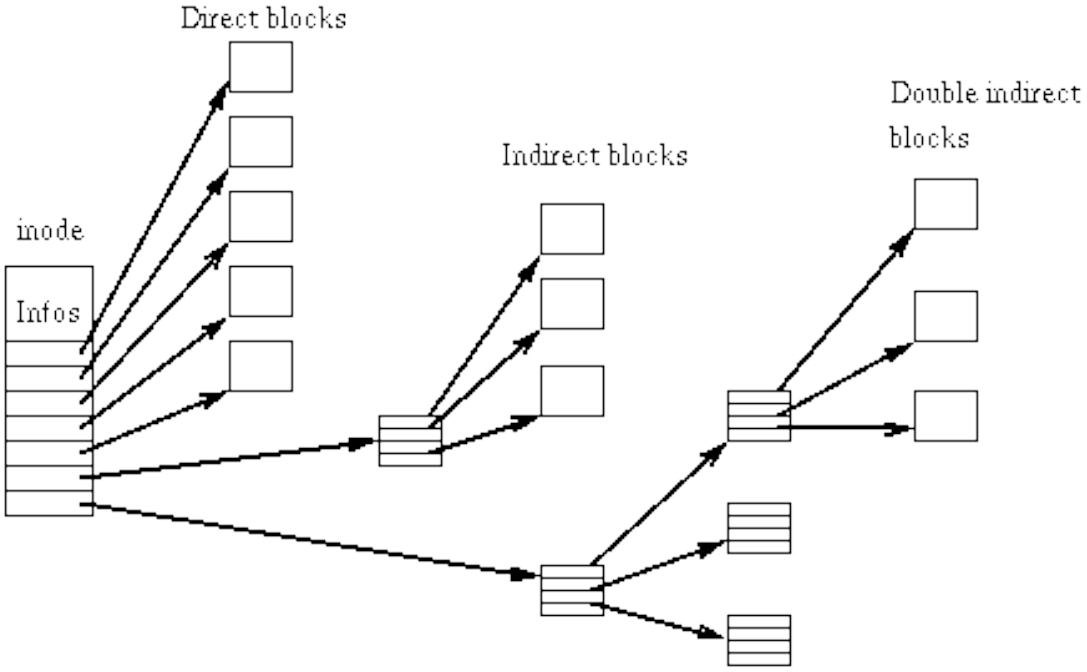
L'**inode** est une structure qui contient **64 champs** parmi lesquels le **numéro d'index** et la **description du fichier** :

1. **index d'inode**, cet index est un **numéro unique** sur le système de fichier (la partition) ;
2. **mode** qui indique le **type de fichier** (fichier normal, répertoire, fichier spécial,...);
3. **droits d'accès** (lecture, écriture, exécution) ;
4. **propriétaires** (utilisateur propriétaire - UID et groupe propriétaire - GID);
5. **horodatage** (accès- atime, modification d'inode - ctime et de fichier -mtime);
6. **taille en octets**;
7. **compteur du nombre de lien physique**
8. **pointeurs vers les blocs de données**.



Ext2 Inode

Les **adresses des blocs de données** allouées à un fichier sont stockées dans son inode. **12 champs** parmi 64 contiennent des **adresses de blocs de données. Si ces 12 blocs de données sont insuffisants** pour contenir le fichier, **3 adresses** sont réservées respectivement en **simple, double et triple indirection** sur **des blocs d'adresses** qui pointent sur **des blocs de données**. Cela est représenté par la figure suivante et permet de gérer des fichiers de grande taille.



blocs de données de l'Inode

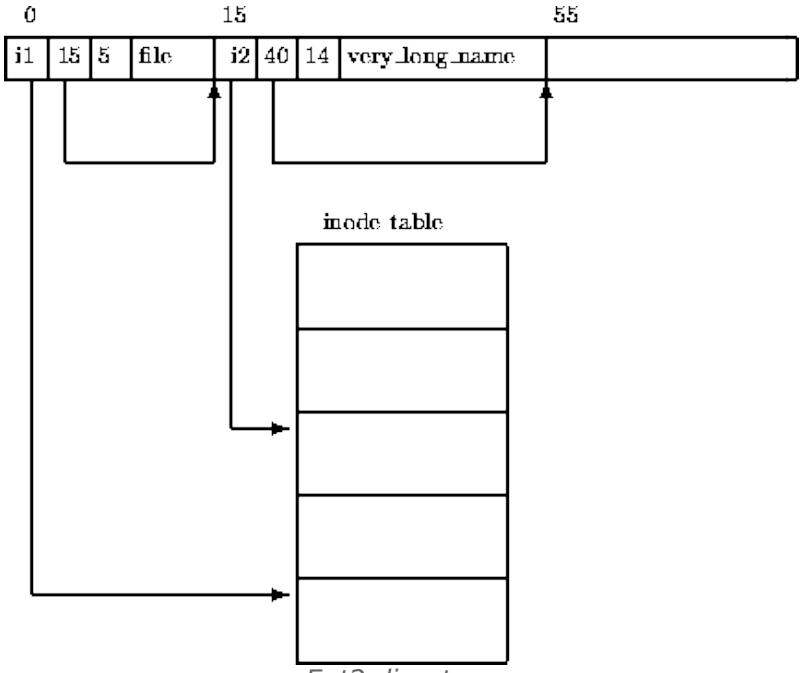
Répertoires

Les **répertoires** sont structurés en un **arbre hiérarchique**. Chaque **répertoire peut contenir** des **fichiers** et des **sous-répertoires**.

En fait, un **répertoire** est un **fichier** (ayant un inode) de **type directory (d)** contenant une **liste d'entrées**. **Chaque entrée** contient :

1. un **numéro d'inode** ;
2. un **nom de fichier ;**
3. la **longueur** du nom de fichier ;
4. et la **taille** de l'entrée .

Cela est représenté dans la figure suivante.



Ext2 directory

Pour **tout répertoire r**, les **deux premières entrées** sont toujours les entrées standards :

1. "**.**" qui a pour inode l'**inode du répertoire r**.
2. "**..**" qui a pour inode l'**inode du répertoire parent de r**.

Du point de vue d'un **développeur**, une arborescence de répertoires et fichiers linux est un arbre hiérarchique, un répertoire est un nœud de l'arbre et un fichier est une feuille, et chaque nœud contient des pointeurs vers ses enfants (nœuds ou feuilles).

Liens physiques/Liens durs

Les **systèmes de fichiers Unix** implémentent le concept de **lien physique ou lien dur**. Un **lien dur** désigne un **nom** pour une **inode**. A la création d'un fichier, le nom du fichier peut être vu comme un lien sur l'inode du fichier. **Plusieurs noms** peuvent être associés à un **inode**.

- L'**ajout d'un lien dur pour une inode i** consiste simplement à **créer une entrée de répertoire**, où **le numéro d'inode est l'inode i**, et à **incrémenter le nombre de liens dans l'inode** i.
- Lorsqu'un **lien sur l'inode i** est **supprimé**, c'est-à-dire lorsqu'on utilise la **commande rm** pour **supprimer un nom de fichier f** qui pointe sur **i** :
  - s'il existe plusieurs liens physiques sur i, le noyau **décrémente le compte de liens** pour **i** et s**upprime l'entrée du répertoire f.**
  - si **f** est le **seul lien physique** sur l'inode **i**, le noyau **désalloue l'inode**.

Ce type de lien est appelé **lien dur** ou **lien physique** et ne peut être utilisé que dans **un seul système de fichiers** : il est **impossible** de créer des **liens durs inter-systèmes de fichiers**. De plus, **les liens durs ne peuvent pointer que sur des fichiers** : un lien dur de

répertoire ne peut pas être créé pour empêcher l'apparition d'un cycle dans l'arbre des répertoires.

Liens symboliques

Les **liens symboliques** sont simplement des **fichiers** (avec leur propre inode) qui **contiennent un nom de fichier** (**chemin**). Lorsque le noyau rencontre un lien symbolique lors d'une conversion de nom de chemin en inode, il **remplace le nom du lien** par son contenu, c'est-à-dire **le nom du fichier cible**.

Les **liens symboliques** peuvent être vu comme l'équivalent des **raccourcis sur Windows**.

Fichiers speciaux

Dans les **systèmes Unix**, l'accès aux **périphériques** (clavier, souris, disques, ...) se fait par des requêtes sur des **fichiers spéciaux** appelés aussi **fichiers de périphérique (device files)**. Un **fichier spécial de périphérique n'utilise pas d'espace** sur le système de fichiers,il s'agit uniquement d'un **point d'accès au pilote** du périphérique.

Il existe **deux types de fichiers spéciaux** :

1. les **fichiers spéciaux de caractères** qui autorisent les opérations d'E/S en mode caractère (caractère par caractère), c'est le cas du clavier.
2. les **fichiers spéciaux de blocs** qui exigent que les données soient écrites en mode bloc (bloc par bloc) via les fonctions de cache tampon, c'est le cas des systèmes de stockages.

Un **fichier spécial** est référencé par un **numéro majeur**, qui identifie le **type de périphérique** (disque IDE, disque SATA, clavier,...), et un **numéro mineur**, qui identifie l'**unité** (premier disque sata, deuxième disque sata, ...).

3. Structure physique du système de fichier

Le système de fichiers EXT2, comme beaucoup de systèmes de fichiers, est construit sur le **principe que les données contenues dans les fichiers** sont conservées dans des **blocs de données**. La **taille des blocs est identique** sur un même système de fichiers et fixée au **formatage**. La **taille de chaque fichier** est arrondie à un **nombre entier de blocs**.

Par exemple, dans un **système de fichier ext2** formaté pour des **blocs de données de 1024 octets**, un fichier de 3500 octets (vous convertissez en ko si vous souhaitez) sera conservé dans 4 blocs de données (3500=1024*3+428).

Les **blocs de données** dans Ext2 sont regroupés dans des **groupes de blocs** qui constituent l'élément de base du système de fichier.

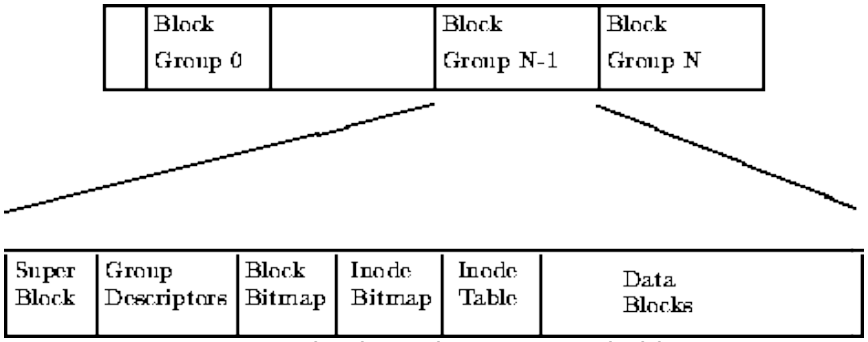
Groupes de blocs

Un **système de fichiers ext2** est composé de **groupes de blocs**. Les groupes de blocs ne sont pas attachés à l'agencement physique des **blocs/secteurs sur le disque** puisque les disques modernes tel que les disques SSD sont optimisés pour les accès séquentiels et pour cacher leur géométrie physique au système d'exploitation.

Chaque **groupe de blocs** contient :

1. une **copie redondante des informations de contrôles cruciales du système de fichiers**
   - les **super blocs**
   - et les **descripteurs du système de fichiers**
2. une partie du système de fichiers :
   - une **bitmap pour la gestion des blocs**,
   - une **bitmap pour les i-noeuds**,
   - une **partie de la table des i-noeuds**,
   - et des **blocs de données**.

Le **groupe de bloc 0** est appelé **secteur d'amorce** du système de fichier. La structure d'un groupe de blocs est représentée par ce schéma :



Structure physique des groupes de blocs

Cette structure offre entre autres avantages :

- une **forte fiabilité** : les **structures de contrôle** du système de fichier sont **répliquées** dans **chaque groupe de blocs**, il est facile de récupérer un système de fichiers dont le super-bloc a été corrompu.
- une **haute performance** : la **distance** entre la **table des inodes** et les **blocs de données** est **réduite**.

Ext2FS réserve certains blocs pour le super utilisateur (root). Normalement, 5% des blocs sont réservés. Cela permet à l'administrateur de rattraper facilement des situations où un processus utilisateur remplit le système de fichiers.

Super bloc

Le **super bloc** contient la description de la **taille** et de la **géométrie du système de fichiers**. Il contient entre autres informations :

1. un **nombre magique** qui permet de vérifier que le superbloc est sur un système EXT2.
2. Les **niveaux de révision majeure** et **mineure** qui permettent au code de montage de déterminer si ce système de fichiers supporte ou non des fonctionnalités qui ne sont disponibles que dans des révisions particulières du système de fichiers.
3. Le **décompte des montages du système** qui est incrémenté à chaque montage (en RW)
4. Un **seuil de nombre de montage** qui chaque fois qu'il est atteint permet au système d'afficher le message d'avertissement "maximal mount count reached, running e2fsck is recommended"(le nombre maximal de montages est atteint, il est recommandé d'exécuter e2fsck),
5. Le **numéro du groupe de bloc**;
6. La **taille d'un bloc** pour le système de fichier en octet. Elle est fixée au formatage ;
7. Le **nombre de blocs par groupe** . Elle est fixée au formatage ;
8. Le **nombre de blocs libres** dans le système de fichier;
9. Le **nombre d'inodes libres** dans le système fichier;
10. Le **premier inode** du système de fichier. Le premier inode d'un système de fichiers racine EXT2 serait l'entrée du répertoire '/'.

Ces informations permettent de maintenir le système de fichiers. Au montage, seul le superbloc du groupe de bloc 0 est lue mais chaque groupe de blocs contient une copie en double en cas de corruption du système de fichiers.

Descripteur du système de fichiers (Descripteur des groupes)

Le **descripteur de système de fichiers** ou encore **descripteur des groupes** est une structure qui fusionne les descriptions de chaque groupe de bloc, il est dupliqué dans chaque groupe de bloc.

Pour chaque groupe de blocs, la description contient :

1. le **numéro de bloc du bitmap d'allocation de blocs** pour le groupe de blocs ;
2. le **numéro de bloc du bitmap d'allocation d'inodes** pour le groupe de blocs ;

3. le **numéro de bloc du bloc de départ de la table des inodes** pour le groupe de blocs ;
4. **le nombre de blocs libres ;**
5. **le nombre d'inodes libres ;**
6. **le nombre de répertoires utilisés**.

Seule la première copie du descripteur (dans le groupe de blocs 0) est réellement utilisée par le système de fichiers EXT2. Les autres copies sont là, comme les copies du super-bloc, au cas où la copie principale serait corrompue.

blocs de données et autres éléments du groupe de bloc

Les autres éléments du groupe de bloc sont :

1. La **bitmap des blocs** du groupe qui est utilisée durant l'allocation des blocs ;
2. la **bitmap des inodes** du groupe utilisée durant l'allocation des inodes ;
3. la **table des inodes** du groupe ;
4. les **blocs de données** qui conservent les données des fichiers. La taille des blocs peut généralement être de 1024, 2048 ou 4096 octets.

Chemin d'accès de fichier

Un **nom de fichier** sur Ext a le même format que tous les noms de fichiers Unix. Il s'agit d'une **série de noms de répertoires séparés par des barres obliques** (``/'') et se terminant par le **nom du fichier** ou du répertoire : on parle de **chemin d'accès**.

Par exemple **/etc/X11/rgb.txt** est un **chemin d'accès** : "/" est le **répertoire racine**, **etc** est un **sous-répertoire de /**, **X11** est un **sous-répertoire de etc** et **rgb.txt** est un **fichier** qui est **dans le répertoire X11**.

Pour trouver l'inode représentant ce fichier dans un système de fichiers EXT2, le système doit analyser le nom de fichier, répertoire par répertoire, jusqu'au fichier :

1. Le système cherche l'inode du répertoire racine "/" (dans la table des inodes du super bloc 0)
2. il cherche l'entrée etc dans la liste des entrées du répertoire racine et récupère son inode
3. à partir de l'inode de etc, il récupère l'inode de X11 dans la liste des entrées de etc.
4. à partir de l'inode de X11, il récupère l'inode de rgb.txt dans la liste des entrées de X11.

Cet inode lui permettra d'accéder aux blocs de données du fichier.

Les fonctions

Le **système de fichiers** utilise plusieurs **fonctions**. Nous pouvons citer :

1. les fonction qui opérent sur les **inodes** (création, suppression,...)
2. les fonctions qui opèrent sur les **fichiers** (lecture, écriture)
3. les fonctions qui opèrent sur les **blocs**

Il utilise également un **cache tampon** pour les opérations d'Entrée/Sortie.

L'**allocation de blocs** entre **plusieurs fichiers** (n'oubliez pas que nous sommes sur un système multitâches) est gérée à partir du verrouillage du super-bloc et se fait sur le principe d'une **file** : **premier arrivé, premier servi**.

Il est intéressant ici de citer **quelques procédés d'optimisations des performances** sur le code du noyau de Ext2fs qui tendent à améliorer la vitesse des E/S lors de la lecture et de l'écriture des fichiers :

- Les **groupes de blocs** sont utilisés pour **regrouper des inodes et des données connexes** : le code du noyau essaie toujours d'**allouer des blocs de données pour un fichier dans le même groupe que son inode**.
- Lors de l'**écriture de données dans un fichier**, Ext2fs **pré-alloue jusqu'à 8 blocs adjacents** lors de l'allocation d'un nouveau bloc. Les taux de réussite de la pré-allocation sont d'environ **75%**, même sur des systèmes de fichiers très pleins. Cette pré-allocation permet d'obtenir de bonnes performances d'écriture en cas de forte charge. Elle permet également d'allouer des blocs contigus aux fichiers, ce qui **accélère les futures lectures séquentielles**.

- Ext2fs tire parti de la **gestion du cache tampon** : lorsqu'un bloc doit être lu, le **code du noyau demande l'E/S sur plusieurs blocs contigus**. De cette façon, il essaie de s'assurer que **le prochain bloc à lire sera déjà chargé** dans le cache tampon. Ext2fs étend ce procédé aux **lectures de répertoires**.

4. Les utilitaires ext2FS

Plusieurs utilitaires sont disponibles sur le système pour la gestion des systèmes de fichier ext :

1. **mke2fs** - Créer un système de fichiers ext2/ext3/ext4
2. **tune2fs** - Ajuster les paramètres des systèmes de fichiers ext2/ext3/ext4
3. **e2fsck** - Vérifier un système de fichiers Linux ext2/ext3/ext4

IV.Le système de fichiers virtuel - VFS

Dans les **systèmes UNIX**, les différents **systèmes de fichiers** ne sont **pas accessibles** par des **identifiants de périphériques** (numéro ou nom de lecteur) mais sont tous **combinés en une seule arborescence hiérarchique** qui représente le tout comme une entité unique. Cela se fait grâce au **système de fichier virtuel - VFS**.

Cette **arborescence** et le **contenu de ses principaux répertoires** sur GNU/Linux est définit par la norme **Filesystem Hierarchy Standard (FHS)**.

Les fichiers des différents **systèmes de fichiers** sont ainsi tous **accessible** par l'intermédiaire du **système de fichier virtuel VFS**. Il s'agit d'une **couche** de code qui met en œuvre les **actions génériques des systèmes de fichiers** et fait une **indirection** vers le **code spécifique de chaque système de fichier** pour traiter la demande.

L'une des **caractéristiques** les plus importantes des systèmes d'exploitation GNU/Linux est leur **prise en charge de très nombreux systèmes de fichiers**. C'est ce qui en fait un **système flexible**, capable de **coexister** avec de nombreux **autres systèmes d'exploitation**.

Dans cette section, nous verrons comment le noyau Linux prend en charge les différents systèmes de fichiers : Nous aborderons les **fichiers** et leur **arborescence** sur le VFS, les **fonctionnalités offertes par le VFS** à l'utilisateur et enfin **l'architecture et le principe de fonctionnement du VFS.**

1. Les fichiers et leur arborescence sur le VFS

Types de fichiers

Sur le VFS, l'utilisateur d'un système GNU/Linux peut manipuler sept types de fichiers différents :

1. **fichier régulier** : désigne les fichiers de données (ASCII, PDF, ...) ;
2. **répertoire** : est un conteneur de fichiers et de répertoires ;
3. **lien symbolique** : désigne un fichier qui pointe sur un autre fichier ;
4. **fichier spécial de caractère** : représente un périphérique en mode caractère ;
5. **fichier spécial de bloc** : représente un périphérique en mode bloc ;
6. fichier de **socket** : fichier dédié à la communication ;
7. fichier de **canal nommé** : fichier dédié à la communication entre processus.

Nous rappelons que sur Linux, tout est fichier.

la commande **file** permet de connaître le type d'un fichier. La commande **ls** également l'affichera en plus d'autres informations.

Arborescence et contenu des principaux répertoires

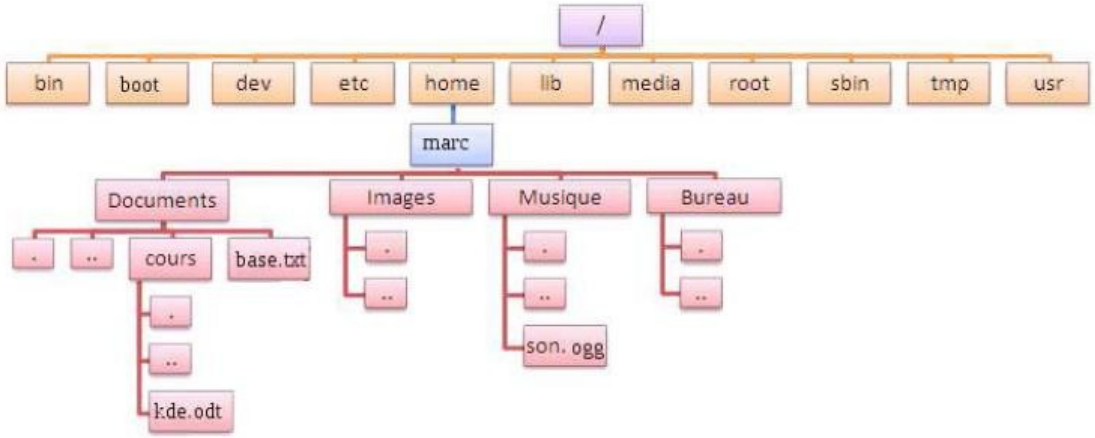
Pour l'**utilisateur** qui interagit avec le **système d'exploitation**, le **système de fichier virtuel** lui présente les fichiers et répertoires sous une **arborescence** :

- la **racine de l'arborescence** est un répertoire appelé **répertoire racine** et dénommé "/" (slash).
- Les **répertoires** contiennent soit des **fichiers**, soit **récursivement** d'autres **répertoires** (sous-répertoires). ce sont des nœuds.
- les **fichiers** sont regroupés dans des **répertoires**. ce sont les feuilles de l'arborescence.

Les principaux répertoires et leur contenu sont définis par la norme FHS :

- **bin** : **commandes de bases** nécessaires au démarrage et à l'utilisation du système.
- **boot** : fichiers statiques du **chargeur de démarrage** (grub, noyau,. . .).
- **dev** : fichiers de **périphériques**: (disques, écran,. . . .)
- **etc** - fichiers de **configuration**
- **home** - **Répertoires personnels** des utilisateurs

- **lib** - **bibliothèques** des exécutables et des modules du noyau.
- **mnt** - **point de montage** des partitions temporaires
- **media** - **point de montage** pour médias amovibles
- **opt** : **logiciels optionnels**
- **root** - **répertoire personnel** de l'utilisateur root.
- **sbin** - **executables de l'administrateur**
- **tmp** - **fichiers temporaires**
- **usr** - **hiérarchie secondaire**. Contient certains répertoires semblables à ceux présents à la racine mais qui ne sont pas nécessaires au fonctionnement minimal du système (X11).
- **var**: fichiers **variables** (base de données, journaux, mail, cache)



Arborescence des répertoires GNU/Linux

Chemin d'accès : chemin absolu/chemin relatif

Le **nom d'un fichier** (tout type de fichier) est manipulé en donnant le **chemin d'accès** qui mène au fichier. **Deux types de chemins d'accès** existent sur les systèmes UNIX :

1. le **chemin absolu** désigne la **succession de répertoire** à parcourir en **partant du répertoire racine "/"** pour **accéder au fichier**. Un **chemin absolu commence toujours par "/"** (la racine) suivi de la succession des répertoires qui permettent d'accéder au fichier, suivi du nom du fichier, séparés les uns les autres par / (ce n'est pas la racine, c'est un / de séparation de chemin).
2. un **chemin relatif** désigne la **succession de répertoire** à parcourir en partant **d'un répertoire R autre que le répertoire racine** pour **accéder au fichier**. On parlera alors de **chemin relatif du fichier par rapport au répertoire R**. Un chemin relatif ne commence jamais par "/" (la racine), il donne la succession des répertoires de R au fichier, suivi du fichier, séparés les uns les autres par / (ce n'est pas la racine, c'est un / de séparation de chemin).

Il faut préciser que chaque répertoire contient **deux répertoires particuliers** :

1. "**.**" qui désigne le **répertoire lui même** ;
2. "**..**" qui désigne le **répertoire parent** de ce répertoire.

Deux fichiers spéciaux sont également disponible sur le système Linux :

- **/dev/null :** Un ”trou noir”. Un fichier en écriture. Tout ce qui y est écrit disparaît à jamais. Toute tentative de lecture n'aboutira à rien. Peut être très utile en ligne de commande et dans certains script.
- **/dev/zero** : Pseudo périphérique spécial qui renvoie une infinité de caractères null (ASCII NUL, 0x00) lors d'une lecture. Souvent utilisé pour fournir un flux de données (écraser des informations ou initialiser un fichier).

Répertoire personnel - Home directory

Chaque utilisateur du système dispose d'un **répertoire personnel** (**Home Directory**) qui porte souvent son **nom** (login) et est **situé dans le répertoire home** (sauf root). L'utilisateur est **propriétaire** de ce répertoire et possède par défaut **tous les droits**. Il peut effectuer toutes les opérations souhaitées sur ce répertoire sachant que les autres utilisateurs ont des droits restreint sur ce répertoire. Le **répertoire personnel de l'administrateur** du système est le répertoire **/root**.

Cette structuration participe à la gestion de la **confidentialité** et de la **sécurité** des données de l'utilisateur.

Le **chemin d'accès du répertoire personnel** de l'utilisateur peut être représenté dans une commande par le caractère "**~**" tilde. Il est consigné dans la **variable d'environnement HOME**. L'**administrateur** du système peut **modifier le répertoire personnel** d'un utilisateur.

Répertoire de travail - working directory

Lorsqu'un **utilisateur se connecte** au système par un **terminal**, le système lui alloue un **répertoire de travail** ou **répertoire courant**. **A tout moment**, sur un **terminal** , un répertoire joue le rôle de **répertoire courant**.

Le **répertoire courant** ou répertoire de travail (**working directory**) est **celui auquel s'applique les commandes** lorsque celles-ci ne spécifient aucun paramètre de chemin d'accès alors qu'il est attendu. Les **chemins relatifs** qui figurent dans les commandes seront également des **chemins relatifs par rapport à ce répertoire de travail**.

Par exemple, la commande ls qui liste le contenu d'un répertoire lancée sans paramètres s'applique au répertoire de travail, elle listera le contenu du répertoire de travail. ls appliqué à un chemin relatif s'applique à ce chemin par rapport au répertoire de travail.

Par défaut, l'utilisateur a pour **répertoire de travail** son **répertoire personnel** mais il peut **changer de répertoire de travail** à tout moment et choisir un autre répertoire (pour lequel il a le droit de positionnement).

L'utilisateur peut **connaître** son **répertoire de travail** avec la commande **pwd** (present working directory). Il peut **changer de répertoire de travail** avec la commande **cd** (change working directory).

2. Fonctionnalités offertes par le système de fichier virtuel

Fonctions du VFS

Le **système de fichier virtuel** fournit un ensemble de **fonctions d'interface** à l'utilisateur du système d'exploitation. Cette **interface** est constituée d'un **ensemble d'opérations** associées à **deux types d'objets** : les **systèmes de fichiers**, les **inodes** et les **fichiers ouverts**. Nous listons ici quelques opérations que peut effectuer le système de fichier virtuel sur ces différents objets :

1. **Système de fichiers** : montage, démontage, vérification, ...
2. **Inodes** : création, déliage (unlink), ..., suppression
3. **fichiers ouverts** : lecture, écriture, ...

Ces opérations sont accessible à l'utilisateur à travers les commandes du système d'exploitation et les programmes informatiques.

2.1. Opérations associées au système de fichier

Montage d'un système de fichier

Tout **système de fichier** sur un **système d'exploitation GNU/Linux** est représenté par un **fichier de périphérique**. Par exemple le **premier disque dur SATA** peut être représenté par le ficher **/dev/sda**. S'il y'a **6 partitions** sur ce disque , ils peuvent être représentés successivement par **/dev/sda1,...,/dev/sda6**. Ces fichiers fournissent un point d'accès au pilote du périphérique et n'occupent pas d'espace.

Pour **exploiter une partition (un système de fichier)** dans le système d'exploitation, il est **nécessaire** de le **monter** sur le système de fichier virtuel, son **contenu** sera ainsi accessible à travers un **répertoire** de l'arborescence appelé **point de montage**.

Le **montage** consiste ainsi à rendre accessible le **contenu d'un système de fichier** (répertoire et fichiers) à partir d'un répertoire de l'**arborescence du système de fichier virtuel**. Le **répertoire** a partir duquel le système de fichier est rendu accessible est appelé **point de montage** du système de fichier. Il est recommandé de choisir un **répertoire vide**, sinon le contenu du répertoire est **caché** jusqu'au **démontage** du système de fichiers.

Le système de fichier racine "/" de la **partition principale** est **toujours monté**. Il est recommandé d'utiliser des sous-répertoires de /media pour le montage de périphériques amovibles et de /mnt pour le montage de partitions temporaires.

Options de montage

Un système de fichier pet être monté sous plusieurs options :

- montage en lecture : on peut lire les fichiers et répertoires
- montage en lecture/écriture : on peut écrire sur le système de fichier.
- montage avec des droits par défaut (umask,fmask,dmask)
- ...

Démontage

Lorsque nous finissons d'exploiter un système de fichier, nous pouvons le **démonter**, il ne sera plus accessible à travers le système de fichier virtuel. Un système de fichier en usage ne peut être démonté (fichier ouvert...).

Exemple : Exemple avec le montage

Considérons que nous utilisons une fois par mois la 2 ème partition du deuxième disque dur de notre ordinateur pour faire de la sauvegarde de données. Ce deuxième disque dur est désigné par le fichier de périphérique bloc /dev/sdb et sa deuxième partition par /dev/sdb2. Pour utiliser la partition, il est nécessaire de le monter sur un répertoire vide. Nous pouvons créer un répertoire "sauvegarde" dans le répertoire /mnt et monter la partition /dev/sdb2 dans ce répertoire. On dira que /mnt/sauvegarde est le point de montage de /dev/sdb2.

Montage automatique

Le système d'exploitation autorise de monter **automatiquement** des partitions du disque au **démarrage du système d'exploitation**, elles sont consignées dans le fichier **/etc/fstab**. Il autorise également de **monter automatiquement** des **périphériques** dès qu'ils sont **connectés au système**. Ils sont aussi consignés dans **/etc/fstab**.

Ce système de **montage automatique** est aujourd'hui automatiquement appliqué par **certaines distributions** comme ubuntu aux **périphériques USB** tel qu'une clé USB ou un disque externe.

Commandes et fichiers liés au montage

- On peut **lister les partitions** avec les commandes suivantes :
  - **fdisk -l** lister les périphériques et leurs partitions ;
  - **lsblk** afficher les périphériques blocs

- Le **montage** se fait avec la commande **mount** :
  - **mount /dev/sdb2 /mnt/sauvegarde** #monte /dev/sdb2 dans le répertoire /mnt/sauvegarde
    le système détecte très souvent le système de fichier mais il est possible de le lui indiquer avec l'option -t :
    mount -t ext4 /dev/sdb2 /mnt/sauvegarde
    Cette commande a beaucoup d'autres options(rw, umask, uid, ...), reportes vous à sa page de manuel
- **mount** affiche également les **montages en cours** (contenu de /etc/mstab)
- Le **démontage** se fait avec la commande **umount** suivi du nom du point de montage :
  - **umount /mnt/sauvegarde**

Citons ces deux fichiers de configuration très importants pour le montage :

1. Le fichier **/etc/mtab** contient la liste des **montages effectués**.
2. Le fichier **/etc/fstab** liste les **partitions montées automatiquement au démarrage** ou à la **connexion du périphérique**.

Vous pouvez vous reportez aux manpages ou à la *documentation ubuntu*[5]

Nous rappelons que les **périphériques de stockage** et **partitions** peuvent être manipulées à travers leur **Identifiant UNiversel Unique (Universal Unique IDentifier - UUID)** qui les identifie de manière unique sur un système ou à travers leur étiquette(label) .

2.2. Opérations liées aux inodes et fichiers ouverts/ Commandes sur le système de fichiers

Pour un **utilisateur** du système d'exploitation, les **opérations** liées aux **inodes du système de fichier virtuel** font référence aux **opérations liés aux fichiers**, l'utilisateur manipule en réalité les fichiers et donc à travers elle les inodes. Par exemple pour **créer une inode de VFS**, l'utilise **créer un fichier**, le VFS crée ainsi une **inode de VFS** et l'associe à une **inode du système de fichier physique**. C'est la même logique avec les **fichiers ouverts** .

Le système de fichier virtuel permet donc à l'utilisateur de faire des opérations sur les inodes (fichiers) et fichiers ouverts à travers les commandes du système (et aussi les programmes) sur les fichiers et les répertoires.

Affichage du contenu de répertoires

ls affiche le contenu de répertoires. Exemple :

$ **ls /bin /etc** # affiche les contenus de /bin et de /etc

Il affiche par défaut le contenu du répertoire courant (pwd) :

$ **ls**

Il possède des opérations intéressantes tel que -i pour affciher les inodes et -l pour un affichage long

$ **ls -il /etc** #affichage long avec les inodes du contenu de /etc

Affichage et changement de répertoire de travail (pwd)

La commande **pwd** affiche le nom du répertoire de travail actuel, il affiche le chemin absolu.

La commande **cd** permet de changer de répertoire de travail, il prend en paramètre le répertoire cible en chemin absolu ou en chemin relatif.

Exemples :

$ **pwd**

/home/lamine

$ **ls**

fic1 rep1 rep2

$ **cd rep1** # rep1 devient répertoire courant

5 - https://doc.ubuntu-fr.org/mount_fstab

$ **cd ../..** # /home devient répertoire courant(c'est le répertoire grand-parent de rep1)

Création et suppression de fichiers et de répertoires

Les commandes permettant de créer des répertoires et fichiers et d'en supprimer sont les suivantes :

1. **mkdir** : créer des répertoires ; l'option -p permet de créer en même temps les répertoires parents ;
   mkdir [OPTION] ... RÉPERTOIRE ...
2. **rmdir** : supprimer des répertoires vides ;
   rmdir [OPTION] ... RÉPERTOIRE ...
3. **touch** : modifie l'horodatage d'un fichier, Nous utiliserons la commande touch **pour** créer des fichier, cela n'est pas son rôle natif, elle permet de modifier l'horodatage d'un fichier mais si le fichier en paramètre n'existe pas, elle le crée.
   touch [OPTION] ... FICHIER ...
4. **rm** : effacer des répertoires et fichiers ;
   rm [OPTION] ... FICHIER ...

Exemples :

$ **pwd**

/home/moussa/tp

$ **mkdir rep5** # création du répertoire rep5 dans /home/moussa/tp

$ **touch fic1 rep5/fic21** # création du fichier fic1 dans tp et fic21 dans rep5 qui est dans tp.

$ **mkdir ../td** # création d'un répertoire td dans /home/moussa (parent de tp)

$ **rm rep5/fic21** # suppression de fic21

Copie, déplacement et renommage de fichiers et répertoires

Les commandes suivantes permettent respectivement de copier, de renommer et de déplacer. ils prennent deux paramètres : la source et la cible.

1. **cp** : copier des fichiers et des répertoires,
   cp [OPTION]... [-T] SOURCE CIBLE
2. **mv** : déplacer ou renommer des fichiers ;
   mv [OPTION]... [-T] SOURCE CIBLE

Exemple :

$ **pwd**

/home/moussa/tp

$ **cp fic1 rep5/** # copie fic1 dans le répertoire rep5

$ **mv fic1 rep1/** # déplace fic1 dans le répertoire rep1

$ **cp -r rep1/ ../td** # copie le répertoire rep1 (et son contenu) dans le répertoire frère de tp : td (/home/moussa/td)

$ **mv fic1 fic3** # renomme fic1 en fic3.

$ **mv fic1 ../fic5** #déplace fic1 dans le répertoire parent (/home/moussa) en le renommant fic5

Création de liens (physiques et symboliques)

La commande **ln** permet de créer des liens entre des fichiers. son option -s permet de créer un lien symbolique :

ln [OPTION] cible nom_du_lien

Exemples :

$ **pwd**

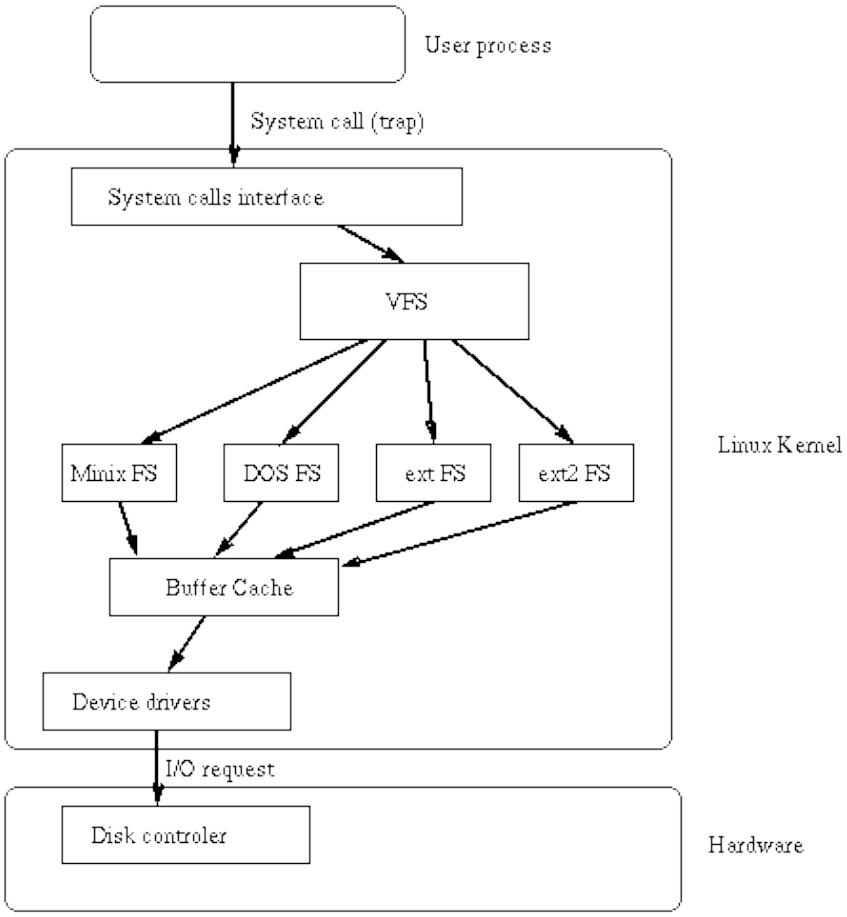
/home/moussa/tp

$ **ln fic1 lienfic1** # crée un lien physique lienfic1 dans tp et qui pointe sur le même inode que fic1

$ **ln -s ../fic5 slienfic5** # crée un lien symbolique slienfic5 dans tp et qui pointe sur le fichier ../fic5 (fic5 du parent)

3. Architecture et fonctionnement du VFS

Architecture du VFS



Le **VFS** est une **couche d'indirection** qui gère les **appels système orientés fichiers** et appelle les fonctions nécessaires dans le **code du système de fichiers physique** pour effectuer les **entrées/sorties**. Le mécanisme d'indirection est une technique très utilisée en informatique pour faire de l'abstraction et de l'intégration.

Lorsqu'un processus émet un **appel système orienté fichier**, le noyau procède comme suit :

1. il appelle une **fonction contenue dans le VFS** qui gère les **manipulations indépendantes de la structure** ;
2. cette fonction redirige l'appel vers une **fonction contenue dans le code du système de fichiers physique**, qui est responsable de la gestion des **opérations dépendantes de la structure**.
3. Le code du système de fichiers utilise les **fonctions de cache tampon** pour demander des **E/S sur les périphériques**.

Le VFS définit donc un **ensemble de fonctions** que **chaque système de fichiers doit mettre en œuvre.** Ce processus est supporté au niveau du VFS par **trois structures** appelées aussi **descripteurs** : le **super-bloc VFS**, l'**inode VFS** et le **fichier VFS**

Superbloc du VFS

Chaque système de fichiers monté est représenté par un superbloc du VFS ; entre autres informations, le superbloc VFS contient les éléments suivants :

1. **périphérique** :l'identifiant du périphérique de bloc dans lequel le système de fichiers est contenu

2. **pointeurs d'inode** : il y'a deux pointeurs :
   a. Le **pointeur d'inode monté** pointe sur le premier inode de ce système de fichiers.
   b. Le **pointeur d'inode couvert** pointe vers l'inode représentant le répertoire sur lequel ce système de fichiers est monté. Le superbloc VFS du système de fichiers racine n'a pas de pointeur couvert, il est toujours monté sur "/".
3. La **taille du bloc** en octets de ce système de fichiers, par exemple 1024 octets,
4. **Opérations sur le superbloc**
   Un pointeur vers un ensemble de routines qui manipule le superbloc pour ce système de fichiers. Entre autres choses, ces routines sont utilisées par le VFS pour lire et écrire les inodes et les superblocs.
5. **Type de système de fichiers**
   Un pointeur vers la structure de données file_system_type du système de fichiers monté,
6. **Système de fichiers spécifique**
   Un pointeur vers les informations requises par ce système de fichiers,

Les pointeurs de fonction (Opérations sur le superbloc) contenus dans ce descripteur permettent au VFS d'accéder aux routines internes du système de fichiers physique.

Inode VFS

Chaque fichier, répertoire, etc. dans le VFS est représenté par un seul et unique descripteur : l'**inode VFS**. Les informations de chaque **inode VFS** sont construites à partir **des informations du système de fichiers sous-jacent** par des routines spécifiques au système de fichiers. Les inodes VFS existent uniquement dans la mémoire du noyau et sont conservés dans le cache des inodes VFS tant qu'ils sont utiles au système. Entre autres informations, les inodes VFS contiennent les champs suivants :

1. **device** : identifiant du périphérique qui héberge cet inode ;
2. **inode number** : Il s'agit du numéro de l'inode et il est unique dans ce système de fichiers. La combinaison du périphérique et du numéro d'inode est unique au sein du système de fichiers virtuel;
3. **mode** : type de fichier ;
4. **user ids** : identifiant des propriétaires (utilisateur et groupe) ;
5. **times** : date de création, de modification et d'écriture ;
6. **block size** : taille du fichier en octets ;
7. **inode operations :** pointeur sur l'adresse de bloc de fonctions spécifiques au système de fichier (création d'inode, unlink,...) ;
8. **count** : le nombre de composants du système utilisant l'inode ;
9. **lock** : champs utilisé pour verrouiller l'inode VFS ;
10. **...**

fichier VFS

Le fichier VFS est une structure qui pointe sur l'adresse d'un bloc de fonctions qui permettent de manipuler les fichiers ouverts (écriture, lecture,...)

fonctionnement du VFS

Les **types de systèmes de fichiers** pris en charge par le noyau sont consignées dans une **table des systèmes de fichiers** qui pointe sur la **fonction à appeler pour le montage** du système de fichiers.

Cette fonction **lit le superbloc du système de fichier** (celui du groupe de bloc 0) au montage et renvoie un **descripteur : le superbloc VFS**. Une fois le système de fichiers monté, les **fonctions VFS** peuvent utiliser les **descripteurs** (superbloc VFS, inode VFS, fichier VFS) pour accéder aux **routines du système de fichiers physique**.

Complément : Recherche de fichiers

Deux outils permettent de faire de la recherche de fichier sur le système de fichier :

1. **locate** : recherche dans une base de données des noms de fichiers. Elle est plus rapide mais pas toujours à jour.
   $ **locate test.c**

chercher le fichier test.c

2. **find** :recherche des fichiers dans une hiérarchie de répertoire
   $ **find répertoire -name nom_du_fichier**
   Quelques exemples :
   $ **find / -name source.list** #recherche source.list dans toute l'arborescence
   $ **find ~ -name tp1.sh** # recherche tp1.sh dans votre home directory
   $ **find ~ -size +1000 -user ibou** # recherche dans votre home les fichiers appartenant à ibou de plus de 1000 ko

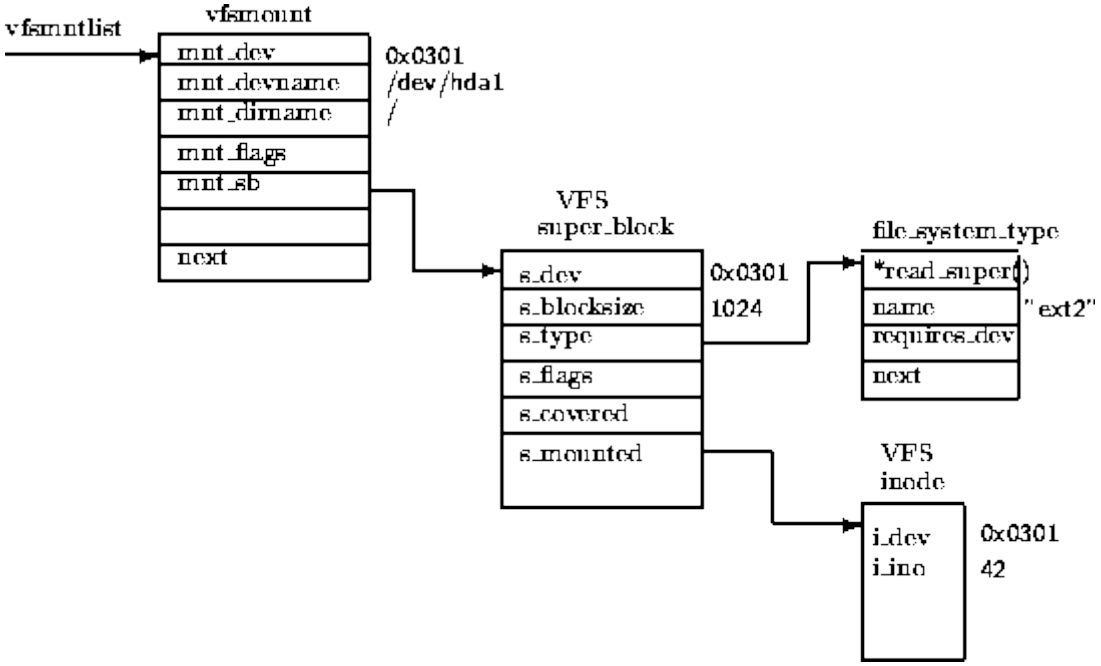
Reportez vous aux pages de manuel pour plus d'options. il est possible d'appliquer une action au résultat de find (commande (exec), suppression (delete)). Exemple :

$ **find ~ -user jean -delete**

recherche dans le répertoire personnel les fichiers appartenant à jean et les supprime.

Complément : Pour aller plus loin

Ce schéma présente quelques éléments techniques d'un système de fichiers monté, il est tiré de cette *ressource*[6] . Vous trouverez un peu de code sur ce *site*[7]



Vous pouvez également rendre disponible le code source du noyau sur votre ordinateur et aller le lire.

6 - http://www.science.unitn.it/~fiorella/guidelinux/tlk/tlk-html.html
7 - https://tldp.org/LDP/khg/HyperNews/get/fs/vfstour.html