esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

SYSTÈME

Linux : les systèmes de fichiers

By Rexy

Table des matières

Introduction

Le concept de fichier sous Linux présente des caractéristiques qui expliquent pour partie la puissance du système. Il est caractérisé par sa simplicité et son universalité. Par ce principe, Linux propose une vision uniforme des objets mis à la disposition des utilisateurs. Un fichier peut être un document ou le programme d'un utilisateur, le noyau lui-même, les utilitaires ou encore des fichiers représentant des périphériques. Quel que soit son type, un fichier est géré avec les mêmes primitives et selon les mêmes principes par le système.

Définition : un système de fichiers (File system = FS) est une entité regroupant un ensemble de fichiers sur un ***support logique***. Il contient non seulement les données des fichiers, mais également un certain nombre d'informations qui permettent au système d'assurer la gestion du SF. C'est une structure apte à supporter des fichiers.

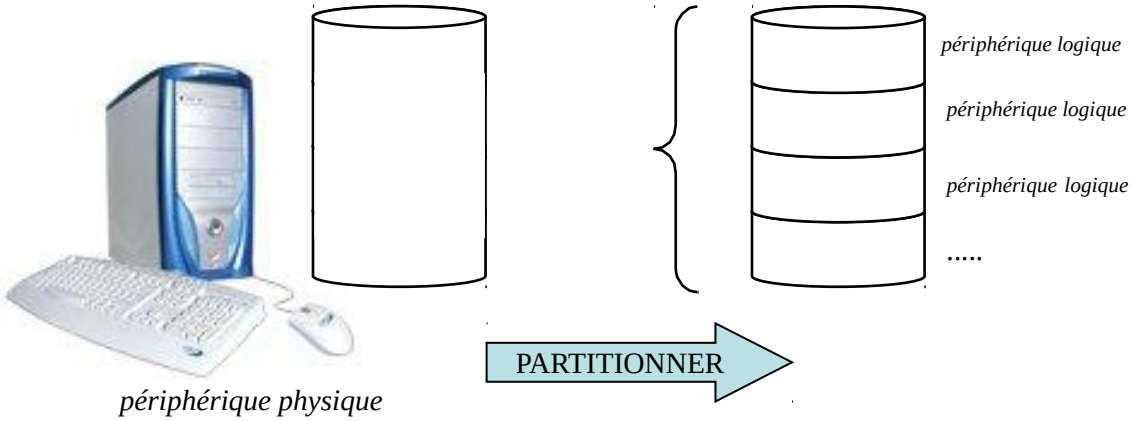
Organisation d'un support de données

L'espace physique d'un support de données peut être divisé en support logique pour plusieurs raisons :

- installer plusieurs systèmes d'exploitation sur le même support physique ;
- diviser l'espace physique d'un même support en plusieurs zones (système, données) pour des raisons importantes de sécurité ou d'optimisation des accès.

L'opération qui consiste à découper ce support s'intitule l**e partitionnement**. Ces supports logiques sont aussi appelés périphériques logiques. En pratique, il est coutume d'appeler le résultat de cette opération **une partition**.

Une partition = partie d'un support sans structure



Cette opération est réalisée avec des outils d'administration (commande fdisk), mais aussi avec des utilitaires spécifiques comme « Partition Magic » ou autre.

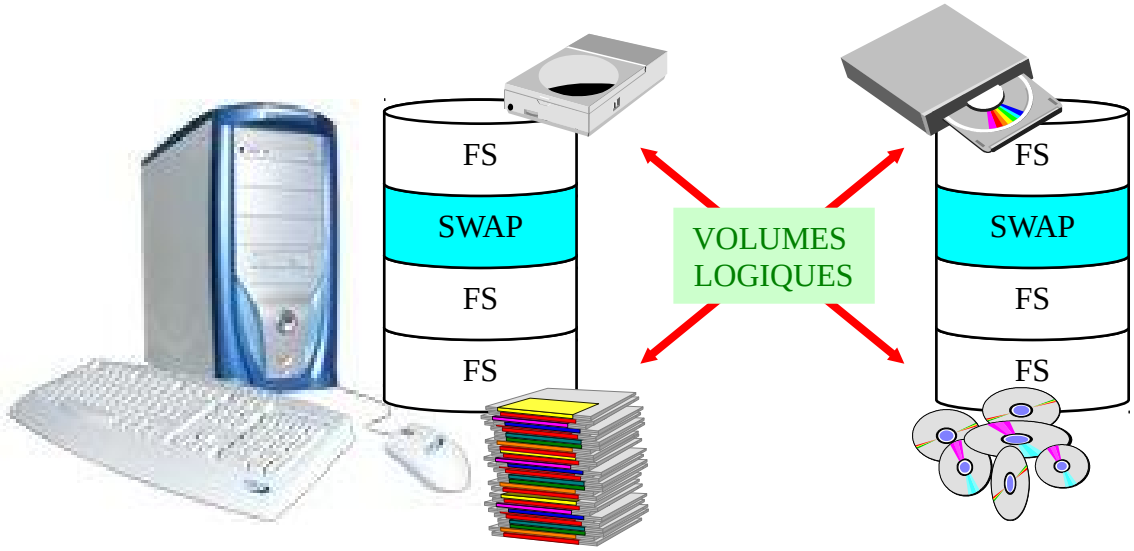
Synthaxe : fdisk /dev/sda

« sda » désigne le premier support (généralement connecté au bus SATA). « sdb » est le deuxième, etc.
Une fois le support partitionné, les partitions seront nommées : sda1, sda2, sdb1, etc.
Il est indispensable de mettre en place une structure de système de fichiers sur chaque partition. Cette opération est réalisable avec la commande **mkfs** (**m**a**k**e**f**iles**y**stem). Elle est parfois appelée, à tort, « formatage ».

Synthaxe : mkfs –t ext4 /dev/sda1

« sda1 » est le nom d'une partition, ext4 un type de système de fichiers.
Pour qu'un FS soit exploitable sur un système (lecture/écriture), il faut le faire reconnaitre par ce système (on dit qu'il doit être « monté »).

Volume logique = partition + FS



La zone de SWAP

La zone de SWAP est une zone de mémoire virtuelle sur disque dur ayant pour rôle d'étendre « artificiellement » la taille de la RAM (Random Access Memory). Cette zone sous Linux occupe généralement l'espace d'une partition spéciale, créée lors de l'installation du système d'exploitation. Elle ne contient pas de système de fichiers et est donc inaccessible directement à l'utilisateur.

Remarques :

- La zone de SWAP peut être un fichier. Exemple, sous Windows, cette zone est un fichier est situé dans l'arborescence du système (Windows\system\pagefile.sys). Un fichier de swap est moins performant qu'un swap disque, car son accès n'est pas direct (on passe par le FS)
- Les commandes « swapon », « swapoff », « swapon –s » réservée à l'administrateur permettant de gérer cette zone sous LINUX.

Structure d'un système de fichiers

Un système de fichiers est composé de blocs contenant les données des fichiers et de blocs contenant les données techniques.

Un bloc logique peut correspondre à un ou plusieurs segments (en général 2, 4 ou 8 blocs physiques). Sa taille est un multiple de 512 octets.

Les blocs d'un même fichier ne sont pas forcement contigus : c'est le système qui mémorise leurs emplacements. De cette manière, les blocs sont alloués dynamiquement au fur et à mesure de l'évolution de la taille du fichier.

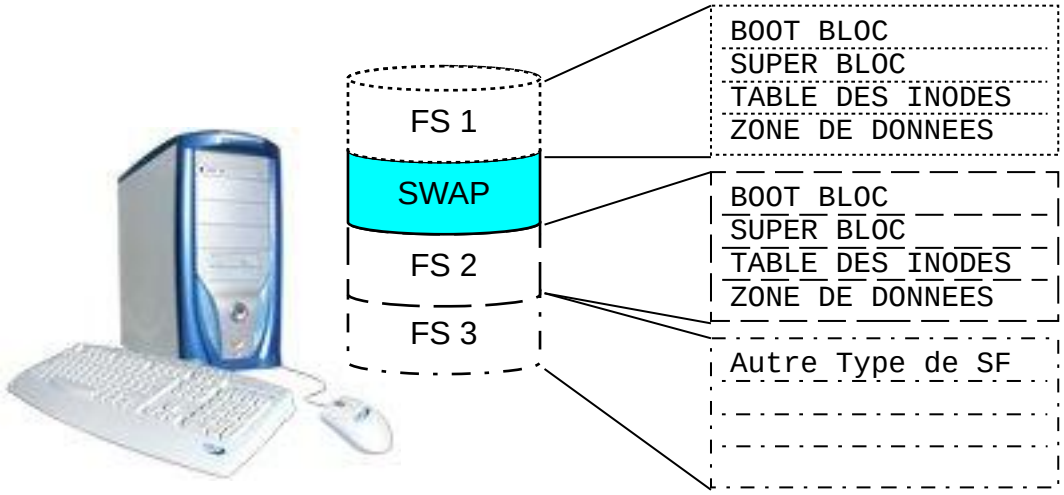
Remarques :

- La taille d'un système de fichiers est figée à sa création.
- La taille des blocs logiques est paramétrable.

La structure d'un système de fichiers est la suivante :

Bloc 0	BOOT BLOC	Contiens le programme de démarrage de

		la machine
Bloc 1	SUPER BLOC	Contiens des tables de contrôle
Bloc 2 à n	TABLE DES INODES	un IndexNode = un fichier)
Blocs n+1 à …	ZONE DE DONNÉES	Ensemble de blocs de données et de liens entre les blocs ; espace disque où sont stockés les différents fichiers.



Le bloc de démarrage

Ce bloc, "bloc 0" sur le disque, contient le(s) programme(s) assurant le démarrage (ou l'amorçage) et l'initialisation du système d'exploitation.
Tous les volumes logiques possèdent ce bloc de démarrage.
Seul celui du volume logique "système" contient les informations de démarrage (bootloader). Sur les autres volumes logique, ce bloc est vide.
Sa taille est un multiple de 512 octets.

Le super-bloc

Ce bloc décrit le système de fichiers. Il contient des tables, générées lors de la création du système de fichiers, qui permettent de contrôler celui-ci.
Voici quelques éléments de description : nom du volume logique, nom du système de fichiers, type du système de fichiers, taille du système de fichiers, nombre de blocs libres,etc.
Sa taille est un multiple de 512 octets.
Afficher le contenu du super-block avec la commande « dumpe2fs »

Synthaxe : dumpe2fs /dev/sman dump2fs

Il est possible également de modifier le contenu du super-block avec la commande tune2fs.

La table des inodes

Lors de sa création, un fichier est référencé de manière unique dans la table des inodes.
Les entrées de la table sont appelées inodes (index-node ou noeud d'index) et sont accessibles par l'index

en question que l'on appelle numéro d'inode.
Cette inode est en réalité une structure (langage C) contenant les informations nécessaires à la gestion du fichier référencé. En particulier, cette structure contient, directement ou non, les adresses des blocs de données constituant le fichier. **Le numéro d'inode est unique au sein d'un système de fichiers.**

C'est par lui que le fichier sera géré par le système (**et non par son nom qui peut être multiple et qui est référencé dans les fichiers de type répertoire**).

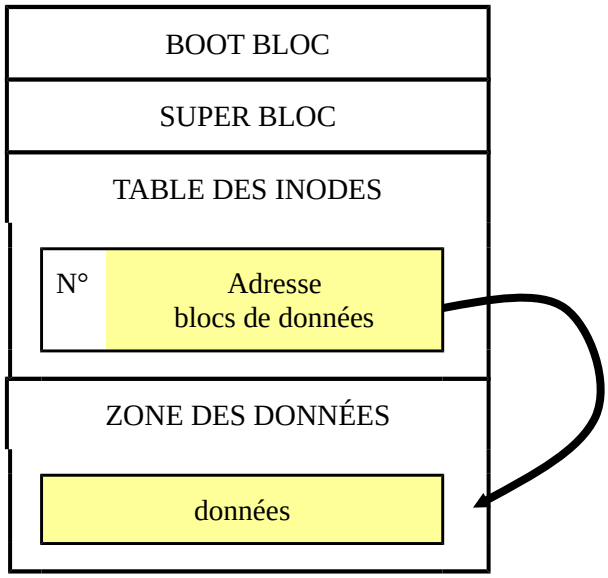
L'inode décrivant le fichier comporte :

- Numéro d'inode,
- Type de fichier et permissions d'accès,
- Numéro d'identification du propriétaire,
- Numéro d'identification du groupe propriétaire,
- Nombre de liens sur ce fichier,
- Taille du fichier en octets,
- Date du dernier accès au fichier,
- Date de la dernière modification du fichier,
- Date de la dernière modification de l'inode (= création),
- Tableau de plusieurs pointeurs (table de blocs) sur les blocs logiques contenant les morceaux du fichier.

Remarques : La taille de la table des inodes détermine le nombre maximum de fichiers que peut contenir un système de fichiers.

La zone de données

Il s'agit de l'espace disque résiduel dans lequel sont stockés les différents fichiers (blocs de données et liens de chaînage entre les blocs).



Cohérence du système de fichiers

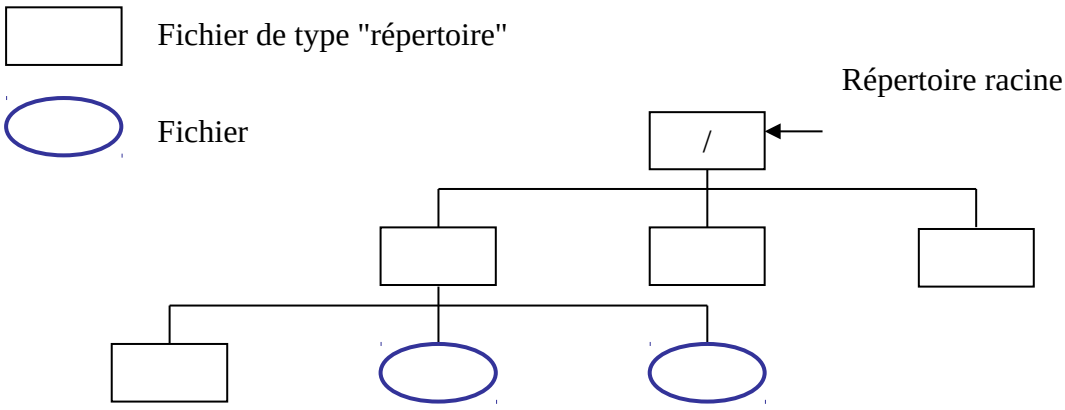
Les systèmes d'exploitation utilisent des zones tampons en mémoire centrale afin de minimiser le nombre d'accès disque.

Une image du super-bloc et de la table des inodes est chargée en mémoire (RAM) lors du chargement du système d'exploitation afin que toutes les opérations d'E/S se fassent à travers ces tables du système.
Lorsque l'utilisateur crée ou modifie des fichiers, c'est en premier lieu cette image mémoire qui est actualisée. Le système d'exploitation doit donc à un moment donné les actualiser (les synchroniser).
Cette mise à jour se réalise donc périodiquement et en dernier lieu au moment où l'utilisateur arrête le système d'exploitation.
Si cette dernière mise à jour ne peut être réalisée, le système de fichiers perd sa cohérence et cela peut provoquer des pertes de données.
C'est pour cette raison qu'il ne faut éviter d'arrêter brutalement un système d'exploitation.
La commande **fsck** permet de vérifier la cohérence du système de fichiers, et le cas échéant de le réparer.

Organisation du système d'exploitation

Organisation du Système de Fichiers

Un système de fichiers est vu par 1'utilisateur comme une structure hiérarchique et arborescente de répertoires (arbre inversé) qui permet d'accéder à n'importe quel fichier du disque.
Les nœuds de cette structure arborescente sont les fichiers de type « répertoire » et les feuilles, des fichiers. Les fichiers ainsi désignés pouvant être eux même des répertoires (ou fichiers répertoires).
L'arbre est construit à partir d'un répertoire racine unique.
Dans le cas particulier du système de fichiers contenant le système d'exploitation, ce répertoire racine est représenté par le caractère " / ".
Tous les autres répertoires sont reliés plus ou moins directement à ce répertoire racine. Ainsi, tous les fichiers appartiennent à la même arborescence.



Les Systèmes de Fichiers supplémentaires

Une station Linux peut posséder plusieurs disques durs, un lecteur de disquettes et un lecteur de cédéroms. Chacun de ces supports peut contenir un ou plusieurs systèmes de fichiers.

Un système de fichiers, créé sur un périphérique logique représentant une unité de stockage, est caractérisé, entre autres, par le nom du périphérique en question, ainsi que par sa taille en nombre de blocs et en nombre d'inodes.

Montage d'un système de fichiers

Un périphérique contient le système de fichiers du système d'exploitation proprement dit ou système de fichiers racine (*root file system).* Les références de ce système de fichiers sont toujours chargées en mémoire centrale au démarrage, et sont donc toujours accessibles.

Pour accéder à un autre système de fichiers, il faut effectuer ce qu'on appelle le *"montage"* (ou **attachement**) du système souhaité. Il s'agit de rattacher les références (Super bloc, Table des Inodes) de ce nouveau système de fichiers.

Le montage d'un système de fichiers consiste à connecter le *répertoire racine* de ce système à un répertoire du système de fichiers accessible. Cette opération est réalisée grâce à la commande ***mount***. L'action qui vise à déconnecter un système de fichiers pour le rendre inaccessible (ou démontage ou détachement) est réalisée par la commande ***umount***.

Remarques :

- L'ensemble des fichiers et répertoires du système de fichiers monté est à considérer, par l'utilisateur, comme faisant partie du système de fichiers racine : il n'y a plus, en fait, qu'une seule structure hiérarchique de fichiers.
- Le système d'exploitation Windows est différent, car chaque volume logique (correspondant à une unité de stockage) est affectée d'une lettre (a: , c:, ...) et garde sa propre structure hiérarchique.
- Les opérations de montage prédéfinies (au minimum le système de fichiers "racine" contenant le système d'exploitation) s'effectuent en général au démarrage de la machine. Ces opérations sont décrites dans des fichiers de configuration que seul l'administrateur peut paramétrer.

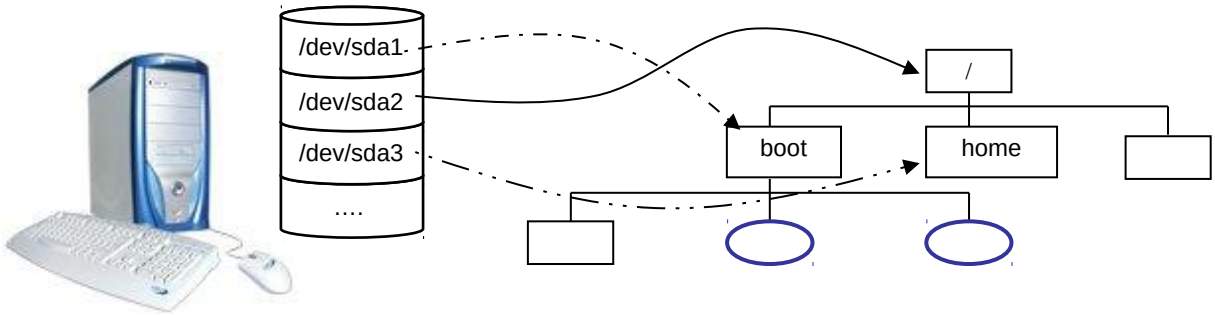
La commande mount

Syntaxe : **mount [-options] source destination**

- **source** est le nom du périphérique logique représentant le système de fichiers à monter
- **destination** désigne le répertoire où sera monté le système de fichiers.

Exemple de montage d'une partition : mount /dev/sda1 /home

- /dev/sda1 représente la première partition du disque « a »
- /home désigne le répertoire où sera montée cette partition



Remarque :

- Lorsqu'un système de fichiers est monté sur un répertoire qui contient déjà des fichiers, ces derniers deviennent inaccessibles pour le système tant que le montage est actif. Ces fichiers ne sont pourtant pas "écrasés". Dès que le système de fichiers "hôte" est démonté, ces fichiers sont à nouveau accessibles.
- En général le système de fichiers racine comporte un répertoire dédié nommé /mnt (répertoire de montage temporaire) sur lequel des systèmes de fichiers supplémentaires (disquettes, Cd Rom) et distribués (NFS) pourront être montés.

Exemples : mount /dev/cdrom /media/cdrom

montage d'un cédérom (volume logique /dev/cdrom) sur le répertoire /media/cdrom

Remarques :

- Tout programme en cours d'exécution sur un système de fichiers empêche tout montage sur celui-ci.
- En conséquence des remarques précédentes, le système interdit le montage d'un système de fichiers supplémentaire au niveau de la racine du système d'exploitation « / ».

La commande umount

Syntaxe : **umount [source]** *ou* **umount [destination]**

- **source** est le nom du fichier spécial représentant le système de fichiers à démonter,
- **destination** est le répertoire où était monté le système de fichiers (en général le répertoire /*mnt*)

Exemple : umount /dev/cdrom ou umount /media/cdrom

Remarque : Pour le démontage, ne pas rester sur (ou en dessous) du point de montage, sinon un message d'erreur rend compte de l'impossibilité de démonter : « device is busy ».

Accessibilité des utilitaires mount et umount par un utilisateur

Les commandes mount et umount ne sont accessibles, en exécution, que par l'**administrateur** du système, ceci pour des raisons de sécurité. Il n'est donc pas possible, en théorie, à un utilisateur de stocker ses programmes et ses données sur un autre périphérique puisqu'il ne pourra pas "monter" celui-ci.

Pour palier cette limitation, l'administrateur dispose d'un fichiers de configuration rendant possible un montage pour un utilisateur : **/etc/fstab**.

Remarques :

- La commande **mount** sans argument provoque l'affichage des volumes logiques actuellement montés :

  # mount

  /dev/sda3 on /home type ext3 (rw)

- La commande **df** (disk free) permet aussi de visualiser les volumes logiques montés et de mesurer la quantité d'espace libre par FS :

File		1K-blocks	Used	Available	Use%	Mounted On
/dev/hda1	4096	3904	24	2%	/	
/dev/hda6	20480	11388	1021	16%	/home	

Les fichiers /etc/fstab et /etc/mtab

Le fichier **/etc/fstab** contient des informations statiques sur les systèmes de fichiers. Il est lu au démarrage par le système d'exploitation. Chaque système de fichiers est décrit sur une ligne indépendante.

Les champs contenus dans ces lignes sont séparés par des espaces ou des tabulations.

L'ordre des lignes est important, car les commandes **fsck, mount** et **umount** utilisent séquentiellement les enregistrements de ce fichier.

Structure du fichier

/dev/sda1	/boot	ext3	defaults	1	2
1	2	3	4	5	6

1	nom du périphérique logique représentant le système de fichiers	/dev/hda1 /dev/sda1
2	désigne le répertoire où sera monté le système de fichiers	/ /home …
3	le type de système de fichiers.	ext2, ext3, vfat …
4	options de montage associées au système de fichiers.	defaults
5	utilisé par la commande « dump » pour déterminer quels sont les systèmes de fichiers à sauvegarder.	1 = sauvegarde, 0 = pas sauvegardé,
6	l'ordre dans lequel la commande "fsck" doit contrôler les systèmes de fichiers lors du démarrage du système.	1 = prioritaire, 2 = pas prioritaire, 0 = pas de contrôle

Tous les montages effectués sont ensuite écrits dans une table prévue à cet effet. Ce fichier est **/etc/mtab**.

Les Systèmes de Fichiers Distribués

Des logiciels supportant le concept de *fichiers partagés* ont été mis au point. Grâce à ce système, un fichier de données ou un programme ne doit plus exister qu'à un seul exemplaire au sein du réseau. De même, les unités de stockage ou de sauvegarde sont utilisables par tous les éléments du réseau.

Le partage des informations parmi les éléments d'un réseau se fait suivant l'architecture *clients/serveur :*

- Un client est une machine qui sollicite une ressource (requête) via le réseau auprès d'une autre machine (serveur).
- Un serveur est la station susceptible d'offrir cette ressource.

Le système NFS

NFS *(Network File System),* développé par *Sun Micro Systems* (1984), est et reste le plus employé.

Il permet de partager les ressources au sein d'un réseau reposant sur l'emploi du protocole Internet.

Il peut inclure des systèmes de fichiers provenant de systèmes d'exploitation hétérogènes. Grâce à lui, des machines de constructeurs différents, disposant de systèmes d'exploitation incompatibles, peuvent échanger des informations si NFS est disponible sur chacune d'entre elles.

Pour pouvoir reconnaître des systèmes de fichiers différents, NFS a conçu une organisation appelée "Système de fichiers virtuels " *(Virtual File System)* qui sert d'interface entre une requête d'information et le système de fichiers local.

Mécanisme de fonctionnement

- Le serveur du réseau détermine ce qu'il offre en ressources (répertoires de systèmes de fichiers) aux clients du réseau.
- Ceux-ci peuvent alors monter cette ressource grâce à une option de la commande ***mount***.
- Pour exécuter cette opération, la connaissance de l'adresse IP du serveur ou de son nom dans le domaine est nécessaire.

Pour visualiser les ressources (ou structures) exportées par un serveur "NFS", on utilisera la commande **showmount**

Exemple : showmount -e @IP_serveur

Pour monter la ressource offerte par le serveur sur votre station cliente, on utilisera la commande **mount**.

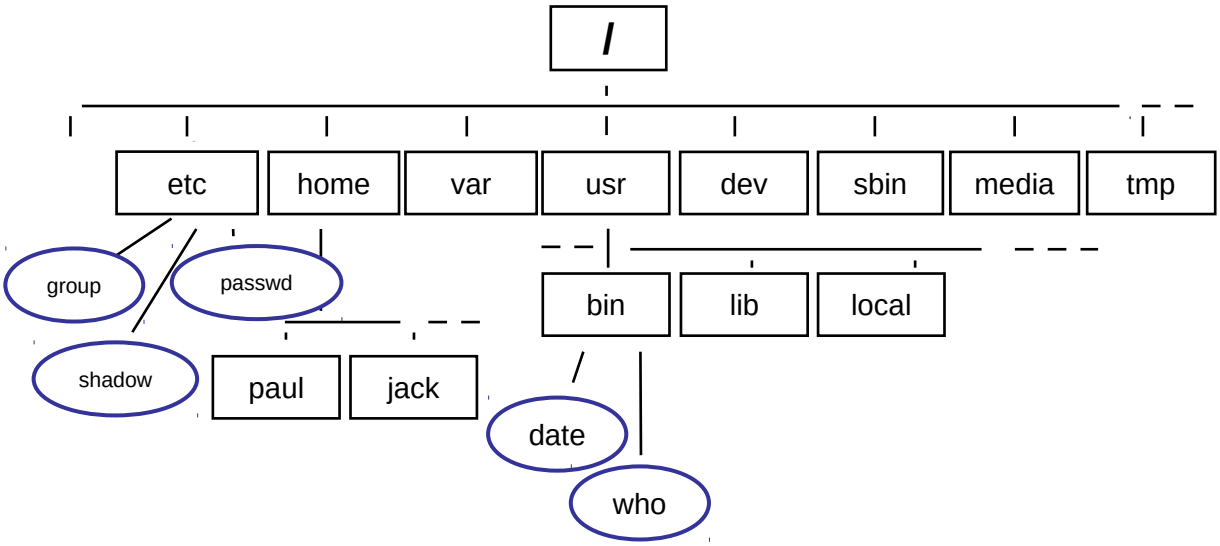
mount -t nfs @IP_serveur:/RessourcesPartagées /media/nfs

Caractère ":" entre l'adresse IP et le nom du répertoire. Pas d'espace de part et d'autre.

Cette commande permet à une station cliente « NFS » de monter le répertoire distant /RessourcesPartagées se trouvant sur la station faisant office de serveur "NFS" d'adresse IP « @IP_serveur » sur sa propre arborescence (/media/nfs).

La commande **mount** sans argument permet de visualiser les montages effectués.

Arborescence type d'un système UNIX/Linux



Chaque répertoire standard est affecté à un domaine bien précis. Cette standardisation permet de retrouver les informations organisées de la même manière d'un système à l'autre.

Dans le détail, cette arborescence varie légèrement selon le système utilisé.

Les répertoires sont utilisés comme suit :

/bin	contiens la plupart des utilitaires et commandes UNIX (les plus utiles)
/dev	contiens les fichiers spéciaux (pilotes de périphériques ou devices)
/etc	contiens les fichiers de configuration (tables du système)
/home	contiens tous les répertoires utilisateurs
/lib	contiens les bibliothèques (libraries) des langages de programmation
/media	contiens les montages temporaires (mount) de systèmes de fichiers additionnels
/tmp	regroupe les fichiers temporaires
/usr	contiens certains sous-répertoires utilisés par le système ou des utilitaires orientés « utilisateurs »
/usr/bin	commandes de l'utilisateur
/usr/local	applications locales et utilitaires liés au système d'exploitation
/sbin	commandes dédiées au système et à l'administrateur
/var	Contiens l'ensemble de fichiers de gestion non figés (files d'attente des imprimantes, boites aux lettres de messagerie, files d'attente des taches, bases de données, fichiers historiques du système,…)

Chemin d'accès à un fichier (pathname)

Pour accéder à un fichier particulier, il faut utiliser, en plus de son nom, un chemin d'accès (*pathname en anglais).* Ce chemin d'accès contient les différents répertoires par lesquels le système doit passer pour accéder au fichier.

Notions liées aux répertoires

- **Répertoire courant** : c'est le répertoire dans lequel se trouve un utilisateur à un moment donné (variable d'environnement **PWD**). Il est symbolisé par un point « . ».
- **Répertoire de connexion** : à l'issue de la procédure de connexion *(login)*, 1'utilisateur se trouve dans un répertoire par défaut défini par le système (variable d'environnement **HOME**). On appelle ce dernier le répertoire de connexion *(home directory).* La commande « **pwd** », immédiatement après votre connexion, vous permet de le connaître. Une deuxième méthode est d'utiliser la variable d'environnement ; « **echo $HOME** ».
- La variable d'environnement **PATH** contient une chaîne de caractères composée des différents chemins d'accès aux commandes accessibles à un utilisateur donné.

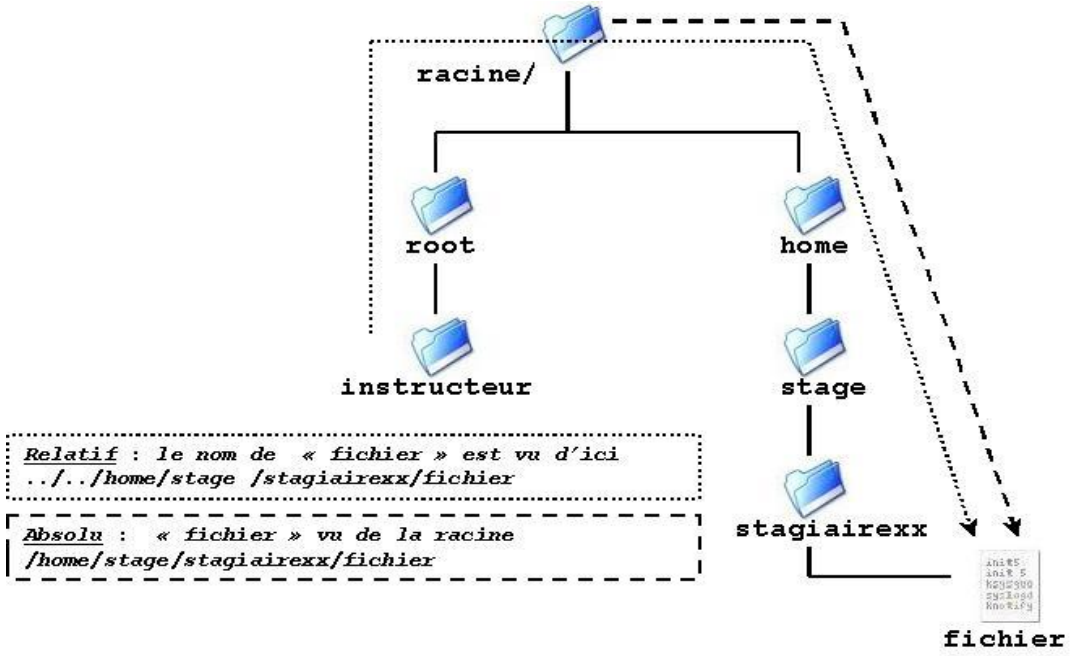
  #echo $PATH

  /bin:/usr/sbin:/….

Types de chemin d'accès

- **Chemin d'accès absolu** : c'est le chemin d'accès complet (*full pathname*). Il commence toujours par le répertoire racine, indiqué par un « / ». Ce caractère est suivi par la liste des noms des répertoires à parcourir, séparés par un « / » et se termine par le nom du fichier.

- **Chemin d'accès relatif** : C'est le chemin d'accès à partir de l'endroit où se situe l'utilisateur dans l'arborescence. Il commence par « ./ », « ../ » ou un nom de répertoire se trouvant dans le répertoire courant.



Les différents types de fichiers

Désignation des fichiers

- Codé sur 255 caractères depuis la version UNIX "System V Release 4" (1989)

- Tout caractère de la table ASCII est utilisable

- Pour un groupe de mots, il faut entourer l'expression choisie de guillemets.

```
mkdir "Mon répertoire"
ls
drwxr-xr-x root root 4096 nov 27 13 :44 Mon répertoire
```

- Majuscules et Minuscules sont différenciées par le système de gestion de fichiers

- Il n'y a pas de notion d'extensions, même si certaines extensions tendent à devenir des standards. Par conséquent, si une extension est donnée à un fichier, elle fait partie intégrante de son nom et doit être spécifiée lorsqu'on l'appelle.

  Exemples de conventions :

.c	fichier source en langage C,
.h	fichier d'entête C,

.o	fichier objet en langage C,
.tar	fichier de données archivées par l'utilitaire "tar",
.cpio	fichier de données archivées par l'utilitaire "cpio",
.gz	fichier de données compressé par l'utilitaire "gzip",
.html	page Web.

Le caractère « . » n' a pas de signification particulière (comme sous Windows), toutefois les fichiers commençant par un point se trouvent **cachés** à certaines commandes du Shell lancées avec leurs options par défaut :

.profile fichiers de configuration d'environnement. Utiliser la commande « ls » avec l'option "-a"

La nature des tâches à effectuer sur eux conduit le système à distinguer sept types de fichiers : ordinaires, répertoires, spéciaux bloc, spéciaux caractère, liens symboliques, tubes, sockets.

Les fichiers ordinaires

Nous distinguerons plusieurs types de fichiers ordinaires suivant leur utilisation :

- **Les fichiers textes** (fichiers ASCII) ou **fichiers de données** et **fichiers multimédias**
- **Les fichiers de programmes (ou sources)**
- **Les fichiers exécutables** de deux types :
  - Soit les fichiers binaires résultant de la traduction par un compilateur approprié des fichiers sources décrits ci-dessus ;
  - Soit des programmes écrits en *langages interprétés* (Shell); les instructions pouvant être directement exécutables par la station.

Exemple : ls –l fic

-rwxr-xr-x 1 prof instruc 26 nov 31 15 :21 fic

le tiret caractérise un fichier ordinaire.

Les fichiers répertoires ou répertoires

Ce sont des fichiers conteneurs qui contiennent des références à d'autres fichiers. Véritable charpente de l'arborescence, ils permettent d'organiser les fichiers par catégorie.

Exemple : ls –l rep

drwxr-xr-x 2 prof instruc 1024 nov 31 15 :25 rep

Le "d" caractérise un fichier répertoire.

Les fichiers spéciaux

Linux communique avec les périphériques comme les disques durs, les modems, les imprimantes par l'intermédiaire de fichiers d'interface appelés fichiers spéciaux (device files ou special file).

On accède à un périphérique comme on accède à un fichier. Ces fichiers sont un peu particuliers, ils ne contiennent

pas de données à proprement parler, mais ils spécifient comment on doit faire pour communiquer avec le périphérique en question.

Il existe deux types de fichiers spéciaux caractérisés par le mode d'accès au périphérique :

- Fichier spécial mode bloc :

Un fichier spécial en mode bloc transfère les données vers le périphérique en utilisant les buffers du système, ce qui permet d'accélérer les transferts. Ce sont des périphériques comme les disques durs, les cédéroms, les disques magnéto optiques.

Exemple : périphériques de stockage

ls –l /dev/fd0

brw-rw-rw- 1 root system 2,0 nov 3 15 :26 fd0

Le "b" caractérise un fichier spécial mode bloc

- Fichier spécial mode caractère :

Un fichier en mode caractère transfère les données vers le périphérique sous forme de flux, c'est à dire un caractère à la fois sans utiliser de buffer. Ce sont les périphériques comme les imprimantes, l'écran ou les bandes DAT. On les appelle aussi les **raw devices**.

Exemple : périphériques de sortie standard, l'écran

ls –l /dev/tty0

crw------- 1 root system 4,0 nov 3 15 :27 tty0

Le "c" caractérise un fichier spécial caractère

Remarque : il existe des pseudo-périphériques

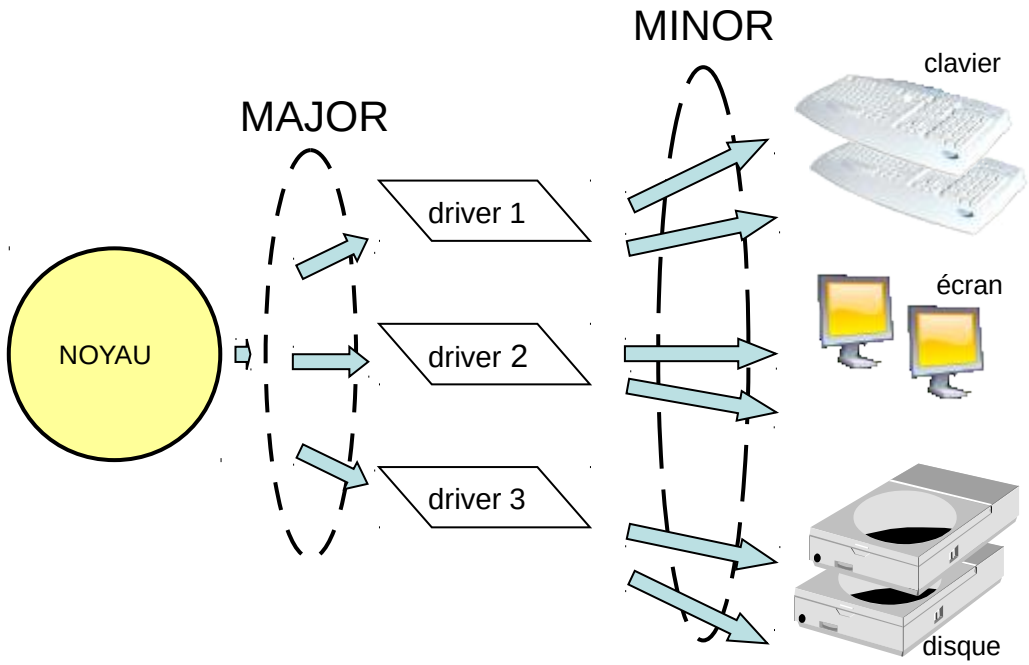
/dev/mem : la mémoire physique;

/dev/null : le "broyeur" pour rediriger les données vers le néant.

/dev/zero : un fichier qui ne contient que des "zero";

Un fichier spécial est caractérisé par un major et un minor :

- le majeur (major number) : ce nombre indique la classe de périphérique à utiliser pour communiquer avec un pilote de périphérique donné (puis son contrôleur).
- le mineur (minor number) : ce nombre est utilisé pour différencier les différents périphériques d'une même classe utilisant le même pilote (même "major number").

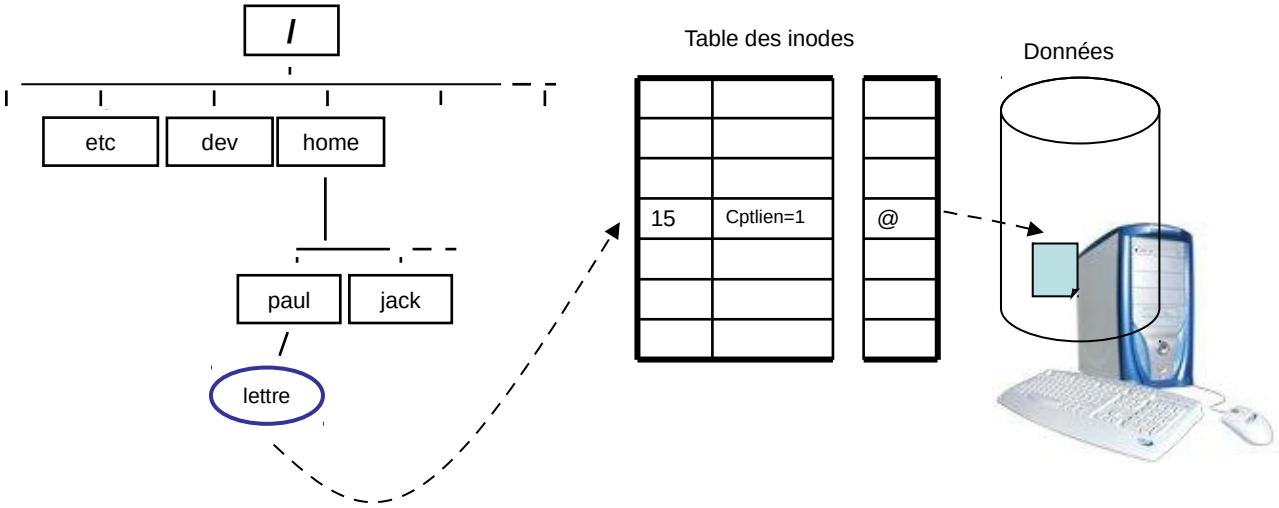


Les fichiers Liens

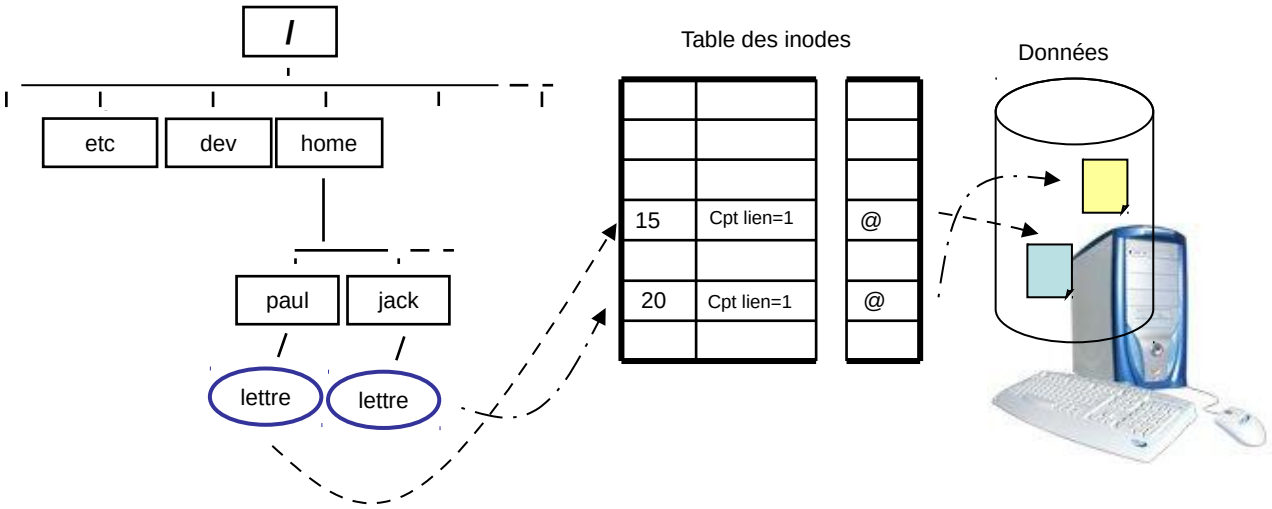
Le système offre la possibilité de donner plusieurs noms à un même fichier physique, sans que celui-ci ait plusieurs copies. Il est dit qu'un nouveau point d'accès au fichier est créé. Contrairement à la copie de fichiers, il n'y pas de duplication du fichier physique.

Dans le but de bien comprendre le mécanisme des liens, un rappel sur la création de fichiers est indispensable. Le but est d'observer les modifications des différentes parties constituant la structure du système de fichiers.

Processus lors de la création d'un fichier



Processus lors de la duplication d'un fichier

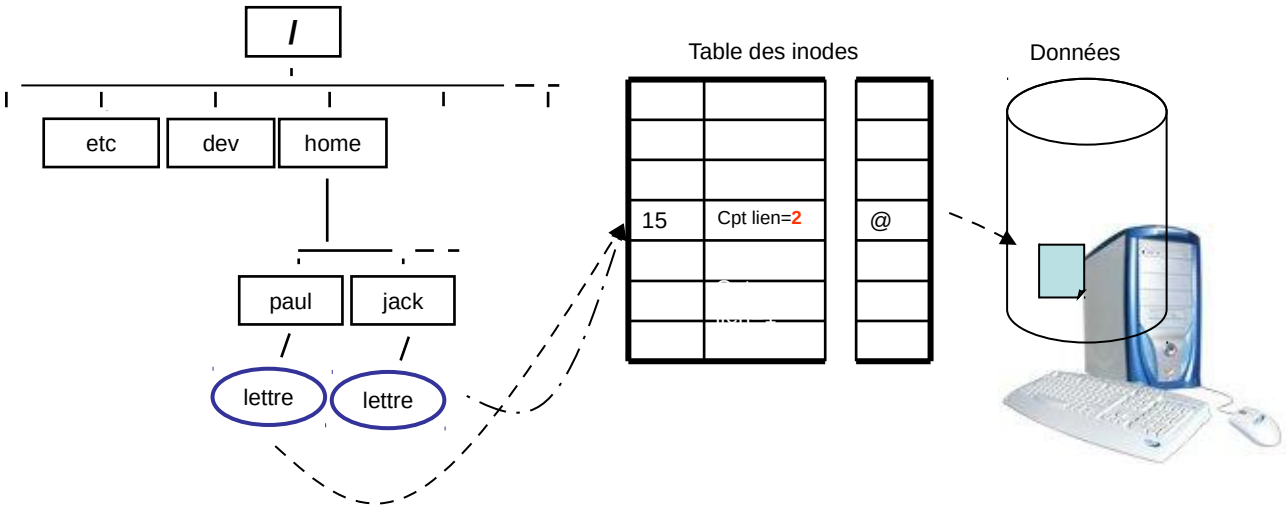


Le **compteur de liens** représente le nombre de chemins d'accès à un fichier donné dans un **même système de fichiers**. Par défaut le compteur de lien est égal à 1.

Les liens physiques

Pour chaque lien réalisé dans un répertoire souhaité, une référence contenant le nouveau nom du fichier et le numéro d'inode du fichier originel est créée. Chacune des entrées, dans les répertoires respectifs, contient donc un nom particulier auquel est adjoint le même numéro d'inode. Le fichier reste "physiquement" unique.

Processus lors de la création d'un lien physique



Exemple :

ls –l /home/paul/lettre

-rwxr-xr-x **2** paul eleve 26 nov 31 15 :21 lettre

compteur de liens

Remarques :

- Les caractéristiques d'un fichier étant stockées dans l'inode, elles sont les mêmes pour tous les fichiers liés.

- Il peut être intéressant de lier un fichier de données afin qu'il soit accessible à plusieurs utilisateurs sans qu'on ait à le copier. Néanmoins, sans mise en place de dispositifs spécifiques, le problème des accès simultanés à un fichier peut surgir.... avec les risques que cela peut produire sur la cohérence des données.

- La notion de ***lien physique*** a quelques limitations : comme elle dépend étroitement de l'unicité des inodes, il est impossible de lier des fichiers appartenant à des systèmes de fichiers différents. De même, il est impossible de lier des répertoires.

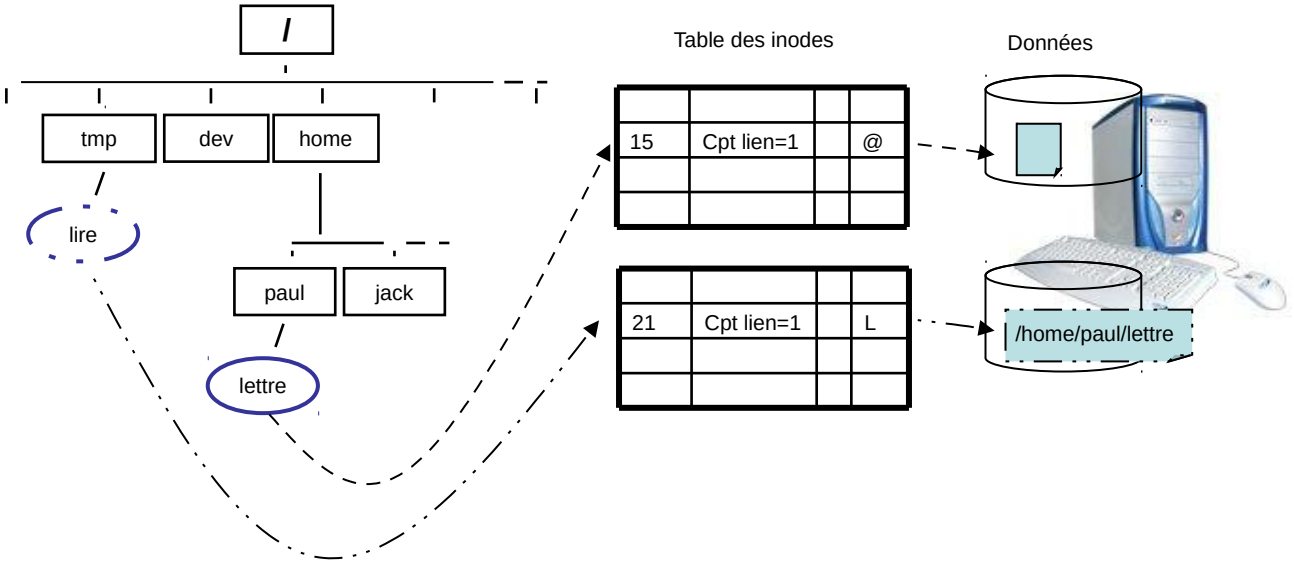
- Ce procédé date du temps (1975 : Version 6 d'AT&T) où les capacités des disques durs étaient moindres. Il est de moins en moins utilisé de nos jours.

Les liens symboliques

Contrairement au lien physique, un lien symbolique implique la création d'un nouvel inode. Un lien symbolique définit un **chemin d'accès** synonyme du chemin d'accès du fichier (ordinaire ou répertoire).

Le fichier créé ne contient qu'une indication sur le chemin permettant d'atteindre le fichier "pointé". Cette notion n'a plus les limitations des liens physiques : il est désormais possible de lier des répertoires et des fichiers appartenant à des systèmes de fichiers différents.

Processus lors de la création d'un lien symbolique



Dans le schéma ci-dessus, on suppose que les répertoires /home et /tmp représentent chacun un système de fichiers différent.

Exemple :

ls –l /tmp/lire

lrwxr-xr-x 1 carine eleve 23 nov 31 15:21 lire → /home/paul/lettre

Le "l" caractérise un fichier de type lien symbolique.

Remarque :

Il est possible de déplacer le fichier lien comme le fichier origine, mais dans ce deuxième cas le lien est coupé, car le chemin à décrire n'est plus le même.

La commande ln

Cette commande crée un lien entre un fichier (le lien) et un fichier existant physiquement. Le lien peut se faire avec un fichier ayant un nom différent ou un fichier ayant le même nom, mais situé dans un autre répertoire.

Syntaxes:

ln fich_source fich_lien : création d'un lien physique, avec le nom de fichier *fich_lien* sur le fichier *fich_source* dans le même répertoire

ln fichier /répertoire : création d'un lien physique avec le même nom de fichier dans ***répertoire***

Rappel : impossibilité de créer des liens physiques sur un répertoire ou sur un fichier appartenant à un autre système de fichiers.

- Un répertoire a toujours au moins deux liens ("**.**" et le nom du répertoire lui-même) créés par le système.
- Le compteur de liens dans ce cas, représente alors le nombre de sous-répertoires (y compris "." et "..").
- L'option *"-s"* de la commande *ln* permet de créer un ***lien symbolique*** :

ln -s fichier_source fichier_lien

Création d'un lien symbolique *fic_lien* sur le fichier *fich_source.*

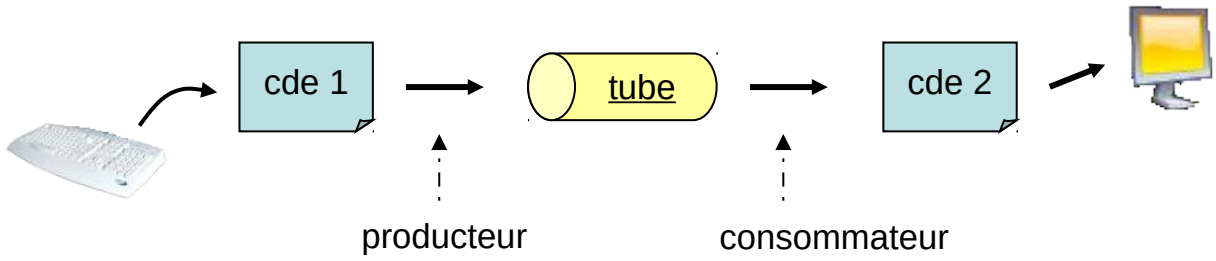
Conseil : lors de la création d'un lien symbolique depuis un répertoire quelconque, pour décrire le chemin, il faut se placer au préalable du côté du fichier lien (fichier final) et voir comment rejoindre le fichier lié (fichier origine).

Les Fichiers de communications

Il s'agit des fichiers *tubes (ou pipe)*, et des fichiers *sockets*.

Les fichiers tubes

Ils permettent de passer des informations entre des processus selon une technique fifo (first in first out) : un processus (le producteur) écrit des informations transitoires dans un fichier *pipe*, et un autre (le consommateur) les lit. Après lecture, les informations ne sont plus accessibles.



Les fichiers tubes sont appelés des pipes nommés (ou pipes fichiers). Ils peuvent être créés par l'utilisateur grâce à la commande *mknod*.

Exemple : mknod fic_tube p

ls –l fic_tube

prwxr--r-- 1 prof instruc 0 nov 31 15 :21 fic_tube

le "p" caractérise un fichier pipe nommé

Remarque :

- Il existe un deuxième type de tube de communication appelé pipe du shell (ou pipe mémoire) : le système d'exploitation crée une zone partagée en mémoire où les deux processus communiquent. (cf. cours processus).
- Pour créer cette zone et relier la sortie standard du premier processus sur l'entrée standard du second, l'utilisateur emploie le raccourci clavier « altgr 6 » ou « | ».

Exemple de deux commandes « pipées » :

ls -l | more

affiche le contenu d'un répertoire en paginant le résultat.

Les fichiers sockets

Les fichiers *sockets* sont des tubes de communication bi-directionnels inter-processus qui utilisent un inode du système de fichiers. De ce fait, ils ne peuvent faire communiquer que des processus situés sur une même station.

D'un autre côté, ces fichiers peuvent être en rapport avec l'outil *socket* de B.S.D. qui permet de communiquer avec d'autres stations via la carte réseau (protocoles TCP/IP ou UDP/IP).

Exemple :

ls –l /dev/log

srw-rw-rw- 1 root system 0 dec 15 06 :20 log

le "s" caractérise un fichier socket

Remarque : Les fichiers de communication ne contiennent que des données transitoires, le temps de la transaction. La commande « ls –l » affichera toujours une taille du fichier nulle.

Attributs des fichiers

Droits d'accès

Cas générique des fichiers

L'accès aux fichiers est contrôlé par un principe de permissions d'accès.
Il existe 4 types de protections qui déterminent les modes d'emploi autorisés d'un fichier :

r	droit en lecture	read
w	droit en écriture	write
x	droit d'exécution	execute
-	aucun droit	

Exemples :

Pour éditer un fichier sous vi et le modifier, il faut que ce dernier ait non seulement les droits en écriture, mais aussi les droits en lecture (ouverture de fichier).
Pour exécuter un script shell, il faut que ce dernier ait non seulement les droits en exécution, mais aussi

les droits en lecture (ouverture de fichier).

Cas particulier des répertoires

Les significations des droits d'accès sont différentes lorsqu'il s'agit d'un fichier répertoire :

r autorise la lecture et le listage du répertoire (commande ls)

w autorise la création ou la destruction d'un fichier

(un fichier protégé en écriture n'est pas protégé contre la destruction)

x autorise le passage (accès aux fichiers et aux sous répertoires ; commande cd)

\- aucun droit.

Remarques : La commande " ls -l" lancée sur un répertoire distant ne peut aboutir si les droits en lecture et en *exécution* ne sont pas posés sur lui, alors que la commande "ls" est possible sur un répertoire sans le droit x (r suffit).

Attention : Un fichier protégé en écriture peut se faire supprimer par un autre utilisateur dans la mesure où les droits du répertoire qui le contient autorisent cet utilisateur à effectuer cette opération.

Classement des utilisateurs

Tout utilisateur déclaré appartient à un groupe déclaré.

La commande "ls -l" affiche le nom du propriétaire du fichier et le groupe d'utilisateurs auquel ce propriétaire appartient.

Exemple : **ls -l fichier**

-rwxrwxrwx 1 paul instruc 60 Nov 2 09:01 fichier

En effet, l'utilisateur qui crée un répertoire ou un fichier en devient le propriétaire.
Le système d'exploitation autorise tout propriétaire et tout administrateur à modifier et supprimer un fichier.
En conséquence, le système d'exploitation classe les utilisateurs potentiels en trois catégories qui déterminent les modes d'emploi possibles des fichiers :

- user pour le propriétaire du fichier
- group pour le groupe d'utilisateurs appartenant au groupe indiqué.
- other pour le reste des utilisateurs du système.

En fonction de cela, chaque catégorie dispose de droits d'accès en lecture et/ou écriture et/ou exécution spécifiques sur le fichier.

-rwx rwx rwx 1 paul instruc 60 Nov 2 09:01 fichier

u **g** **o**

Attributs spéciaux

Sticky-bit

Cet attribut spécial a deux fonctions différentes selon qu'il est posé sur un fichier ordinaire ou sur un répertoire.

Sur un fichier

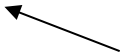
Lorsque le bit de permanence *(sticky-bit* littéralement "bit collant" ou bit de résidence) est positionné **sur un fichier**, le code du programme reste **résidant** en mémoire après qu'il ait été exécuté. Cette possibilité permet

d'éviter des transferts disques à des programmes qui sont appelés fréquemment.

Exemple :

ls –l MonFichier

-rw-r--r-**T** 1 prof instruc 12 dec 25 23:59 MonFichier



le "T" représente un sticky bit positionné

Remarques : Le **sticky-bit** positionné sur un fichier qui n'a pas le droit en exécution pour les autres, est représenté par un "**T**". Si ce droit "x" est positionné, le sticky est représenté par un "**t**".

Sur un répertoire

Comme nous l'avons signalé précédemment, un utilisateur qui a le droit d'écrire dans un répertoire peut également effacer n'importe quel fichier de ce répertoire.
Pour y remédier, on positionne le sticky-bit, ainsi, un utilisateur ne peut effacer que les fichiers qui lui appartiennent.
Rappel : un fichier protégé en écriture n'est pas protégé contre la destruction si les droits posés sur le répertoire qui le contient autorisent les modifications de celui-ci !
Exemple :

ls –ld /tmp

drwxrwxrw**t** 5 root system 1024 fev 29 12:18 /tmp

Set-User-ID (SUID) et Set-Group-ID (SGID)

Il s'agit d'une possibilité supplémentaire de permissions d'accès attribuées à un utilisateur lorsqu'il est nécessaire qu'il dispose des mêmes droits que ceux du propriétaire d'un fichier, ou ceux du groupe concerné.

En effet, un utilisateur peut avoir à exécuter un programme (en général un utilitaire système), mais ne pas avoir les droits d'accès nécessaires.

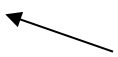
En positionnant les bits adéquats [bits "s" au niveau du propriétaire et (ou) au niveau du groupe], l'utilisateur du programme possède, pour le temps d'exécution de celui-ci, l'identité du propriétaire (ou celle du groupe) du programme.

Par exemple, le fichier /etc/shadow qui contient des informations sur les mots de passe des utilisateurs ne peut pas être modifié par ceux-ci. (droit de /etc/shadow –r-- --- ---).
Le programme /usr/bin/passwd est un utilitaire, appartenant au super-utilisateur, qui permet à chaque utilisateur d'effectuer la modification de son mot de passe. (mot de passe écrit dans le fichier /etc/shadow)
Le changement de mot de passe par un utilisateur est possible grâce au SUID positionné sur la commande /usr/bin/passwd [droit d'exécution du propriétaire à "s"].
Pendant l'exécution de la commande passwd, l'utilisateur bénéficie des permissions d'accès de "root": il peut donc modifier le fichier /etc/passwd.

Exemple :

ls –l /usr/bin/passwd

-r-**s**--x--x 1 root system 12 dec25 23 :59 usr/bin/passwd



le "s" représente un SUID positionné.

Remarques :

- Le *SUID* positionné sur un fichier qui n'a pas le droit en exécution pour le propriétaire, est représenté par un "S"; si ce droit "x" est positionné, le SUID est représenté par un "s".

- Le *SGID* positionné sur un fichier qui n'a pas le droit en exécution pour le groupe, est représenté par un "S"; si ce droit "x" est positionné, le SUID est représenté par un "s".

La commande chmod

Cette commande permet de changer les permissions d'accès pour un ou plusieurs fichiers. La commande peut utiliser des paramètres *absolus ou symboliques.* Elle ne peut être utilisée que par le propriétaire du fichier ou par l'administrateur.

syntaxes:

chmod** [-R] **mode fichier

chmod [ugoa] [+ | - | =] [droits] fichier

L'option **–R** appliquée à un répertoire rend possible la **R**écursivité de la commande. Dans le cas présent, la commande change les droits du répertoire et de tous les fichiers se trouvant dessous.

Méthode symbolique

Principe :

Dans cette méthode, il faut spécifier trois éléments
- indicateur de destination : u (user) pour le propriétaire,
  g (group) pour le groupe,
  o (others) pour les autres,
  a (all) pour tous (u+g+o).
- indicateur d'opération : + ajoute la ou les permissions,
  - retire la ou les permissions,
  = assigne la ou les permissions.
- indicateur de permission : r en mode lecture,
  w en mode écriture,
  x en mode exécution,
  s permet l'activation du S(U)(G)ID concerné,
  t met en place un sticky-bit.

Exemples :

- chmod a+r MonFichier autorise la lecture pour tous
- chmod ug-x MonFichier retire le droit d'exécution au propriétaire et au groupe
- chmod o=rwx MonFichier donne tous les droits aux autres utilisateurs

Méthode absolu(e) ou octal(e)

Principe : Chaque permission possède une valeur numérique :

r = 4, w = 2, x = 1

Il suffit alors d'ajouter ces valeurs pour obtenir les droits dédiés au propriétaire, groupe ou autres utilisateurs.

Exemple :

chmod	7	3	4	MonFichier
	r w x	- w x	r - -	
	4+2+1	0+2+1	4+0+0	
	7	3	4	

L'addition des valeurs donne la valeur des permissions.

La commande umask

Tout fichier créé sur un système Linux possède par défaut des permissions.
La commande umask permet de modifier ces permissions par défaut. Sans paramètre, la commande affiche la valeur d'umask en cours.

Syntaxe : ***umask [valeur_octale]***

Le mode de fonctionnement est basé sur le principe du complément à 7.

Les paramètres de **umask** sont constitués par trois chiffres en octal.

Déduire les permissions d'accès à partir de la valeur de l'umask :

Droits maxima	777	rwx rwx rwx
Valeur umask	-022	--- -w- -w-
Droits obtenus	755	rwx r-x r-x

Déduire l'umask à partir de la valeur des permissions d'accès :

Droits maxima	777	rwx rwx rwx
Droits fichiers	-755	rwx r-x r-x
Valeur umask	022	--- -w- -w-

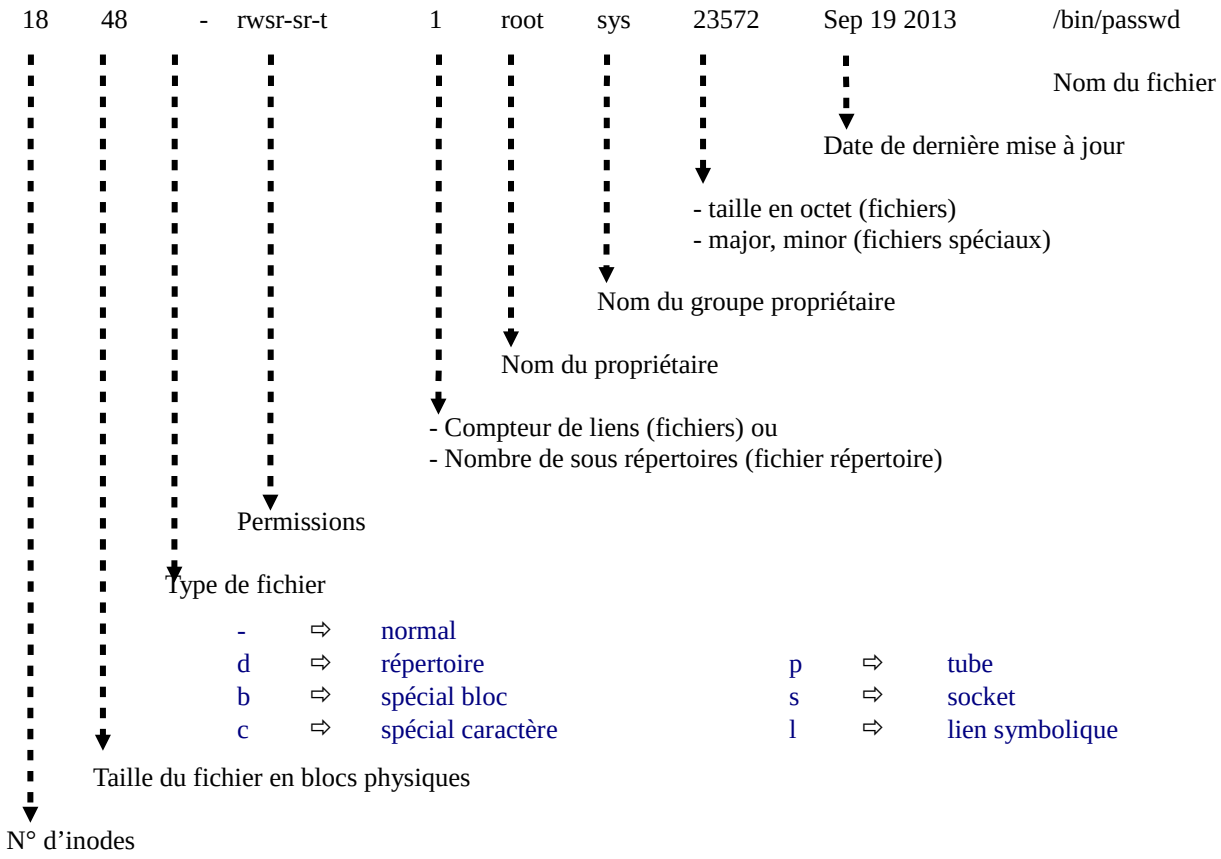
- Les valeurs numériques de la commande umask sont les mêmes que celles de la commande chmod (en mode absolu). Le 1er chiffre représente le propriétaire, le 2ème le groupe et le 3ème les autres.
- Les droits définis ci-dessus sont les droits par défaut attribués à un fichier de type « répertoire »
- Les autres types de fichier ne possèderont jamais les droits en exécution dès leur création. Le système leur retire les droits en exécution « x ».
- Notons que umask fonctionne à l'inverse de chmod en mode absolu : les droits spécifiés sont ôtés dans umask alors qu'ils sont positionnés dans chmod.

Exemple : umask 022

Droits d'un répertoire à la création	rwx r-x r-x	755
Droits des fichiers à la création	rw- r-- r--	644

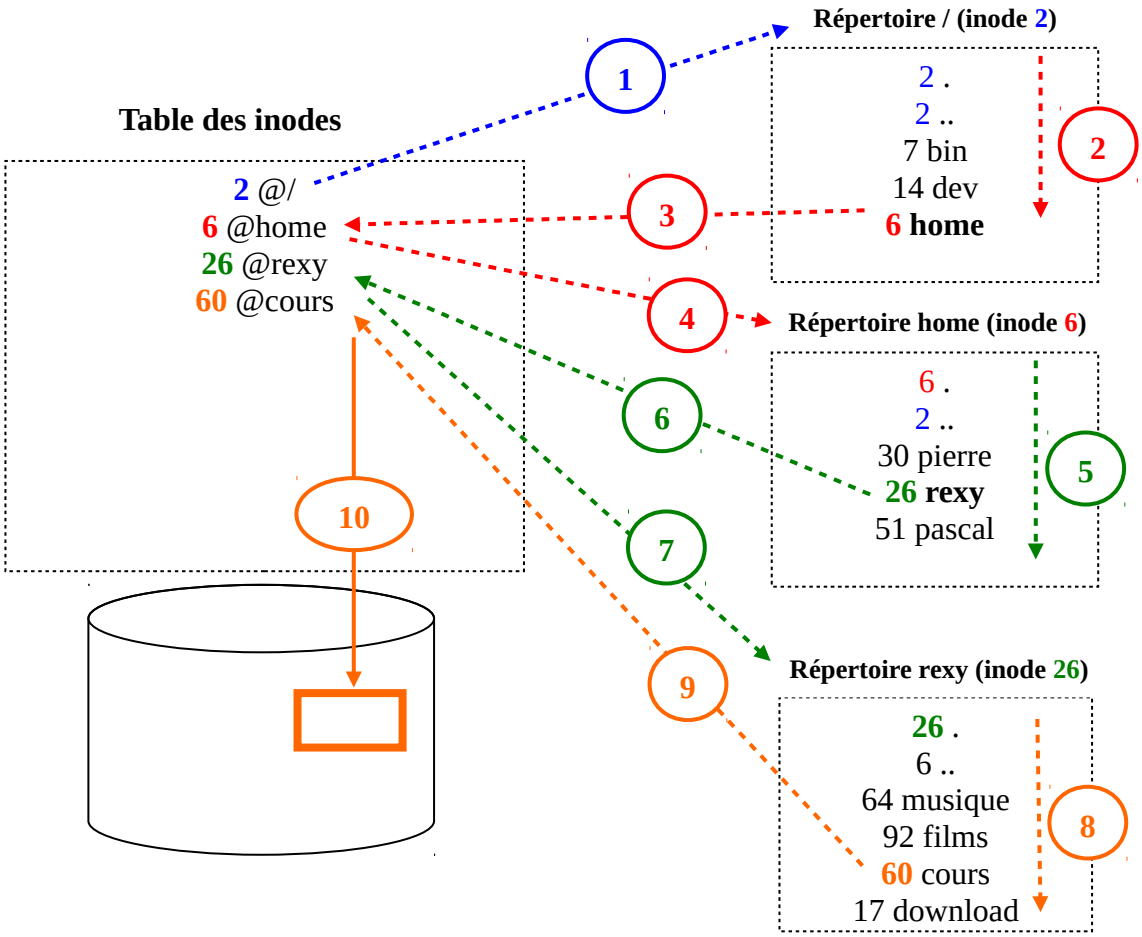
Synthèse des attributs

ls -lisa /bin/passwd



Compléments : procédure d'accès à un fichier

Compte tenu de la manière dont est constitué le système de fichiers, la procédure suivie par le système pour retrouver l'adresse d'un fichier et de ses différents blocs de données est la suivante :
Exemple : accès au fichier « /home/rexy/cours ».



1 " / " est le fichier répertoire racine. Il correspond à l'inode n°2 et il est connu nativement par le système. Le système charge les blocks de cet inode. Comme il s'agit d'un fichier de type répertoire, son contenu est un table de correspondance « *nom de fichiers / inodes* ». Particularité de cet inode « racine » : les entrées « . » et « .. » sont associées au même inode afin de bloquer l'arborescence.

2-3. *En balayant cette table, le système détermine que le* champ « *home* » correspond à l'inode 6.

4 Le système charge les block de données de l'inode 6. Le contenu correspond aussi à un fichier de type répertoire.

5-6. Le champ « rexy » de cette table est lié à l'inode 26.

7-8-9 Le système charge cet inode 26 qui correspond encore à un fichier de type répertoire. Le champ « *cours* » *correspond* à l'inode 60.

10 Dans l'inode n°60 sont stockés les pointeurs vers les blocs de données constitutifs du fichier "cours". Le système peut donc charger le fichier.