Gestion de l'espace disque

10.1 « FILE SYSTEM »

10.1.1 Organisation des « file systems »

L'**organisation du système de fichiers** est une organisation logique basée sur une organisation physique constituée d'un ou plusieurs disques. Chacun est divisé en une ou plusieurs partitions physiques (P1, P2, etc.). L'espace disque physique disponible est ainsi découpé en disques logiques. Initialiser un disque logique consiste à créer sur ce disque un système de fichiers appelé ***file system***.

Sous Unix/Linux, l'utilisateur ne voit pas les disques physiques, pas plus que les partitions. Il n'y a pas de lettres d'unités. L'arborescence de fichiers est construite au démarrage du système à partir de plusieurs *file systems*. Un *file system* particulier, le **root file system** contient la racine de l'arborescence, le noyau, les fichiers systèmes, et des répertoires vides (*/tmp, /usr,…*) sur lesquels seront greffées des arborescences de fichiers se trouvant dans d'autres *file systems*. (Fig. 10.1).

Lors du démarrage d'une machine, après recherche du **disque système**, chaque ***file system*** est rattaché, à l'aide de la commande `mount`, à un répertoire existant appelé ***point de montage*** (***mount point***). La racine de ce *file system* prend alors pour nom celui du point de montage, et l'arborescence de fichier qu'il contient est alors totalement intégrée à l'arborescence de fichiers du système.

La structure d'un *file system* est globalement la même dans toutes les versions d'Unix/Linux : chaque *file system* est composé d'un grand nombre de blocs contigus. La structure de base d'un *file system* est détaillée à la figure 10.2.

- Le bloc 0 contient le boot et l'identification du disque.
- Le bloc 1, appelé aussi superbloc, contient des informations sur le *file system* :

- la date de dernière mise à jour du *file system*,
- le nombre de fichiers qu'il peut contenir,
- la taille du *file system*,
- un pointeur sur la liste des blocs libres.

• Les blocs 2 à k contiennent les inodes.

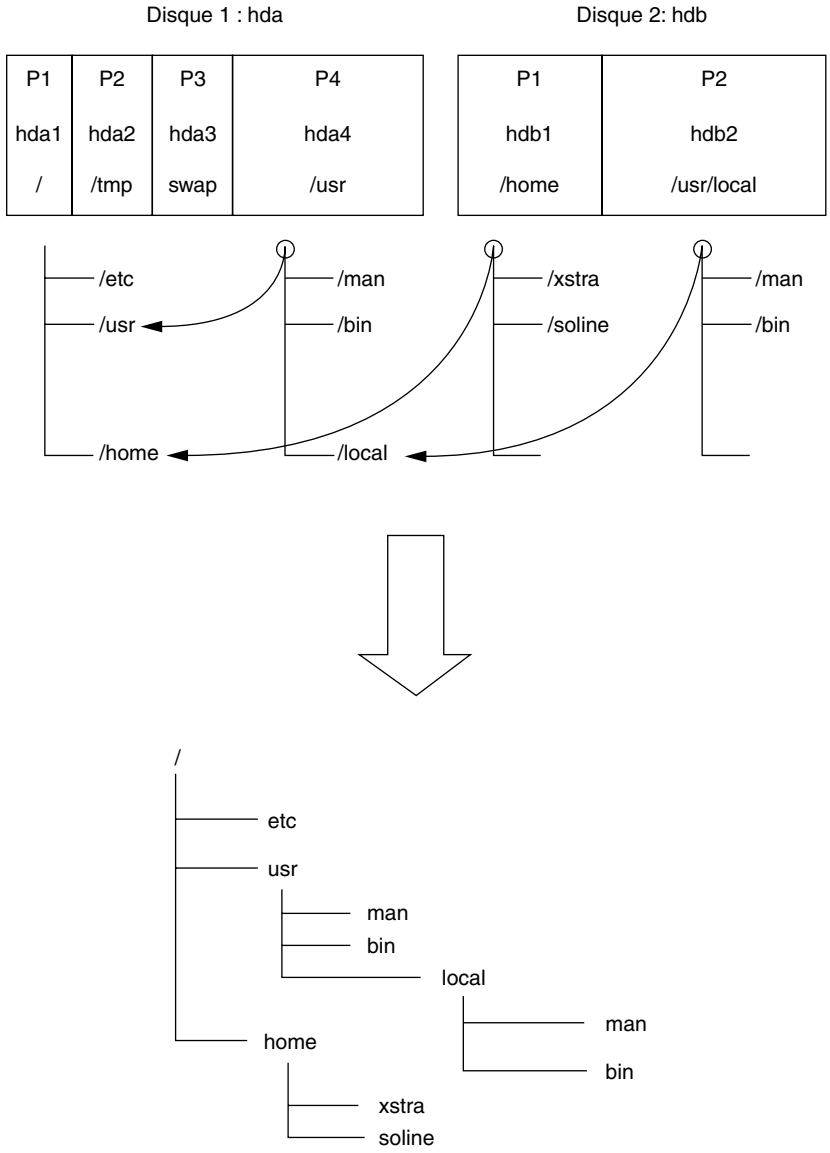

FIGURE 10.1. LES PARTITIONS ET L'ARBORESCENCE DES FICHIERS.

Bloc	*Contenu*
0	Boot bloc
1	Super bloc
2 … … k	Table des inodes
k + 1 … … …	Blocs de données

FIGURE 10.2. STRUCTURES DE BASE D'UN *FILE SYSTEM*.

Sous Unix/Linux, un fichier est une suite d'octets (8 bits) non structurée (byte stream). Toutes les informations concernant un fichier sont regroupées dans une structure de données, appelée **inode**, décrivant la taille du fichier, son emplacement sur disque, son propriétaire, etc. (voir le paragraphe 10.1.2). Un fichier n'est pas repéré par son nom mais par le numéro (index) de l'inode qui le caractérise. Cet index permet de retrouver l'inode dans la table des inodes. C'est donc **l'inode qui caractérise le fichier**.

Le nombre d'inodes, et donc le nombre de blocs qui est réservé à l'enregistrement des inodes, est déterminé à la création du *file system*. Le nombre maximum de fichiers par *file system* est égal au nombre d'inodes alloués à l'initialisation.

- À partir du bloc k+1 sont stockées les données : fichiers et répertoires.

10.1.2 Les inodes

Chaque inode est repéré par un numéro (index) variant de 1 à n et permettant de retrouver son emplacement dans la table des inodes. Aucun fichier n'utilise le numéro 1. Le numéro 2 représente la racine du *file system*. Les informations contenues dans un inode sont :

- le type de fichier (répertoire, fichier ordinaire,…),
- les 12 bits de permissions : suid, sgid, t, rwx rwx rwx
- le nombre de liens,
- le numéro de propriétaire (UID),

- le numéro de groupe (GID),
- la taille du fichier en octets,
- des pointeurs vers les blocs de données correspondants,
- les date et heure du dernier accès,
- les date et heure de la dernière modification,
- les date et heure de la dernière modification de l'inode.

10.1.3 Le répertoire

Le répertoire est un fichier contenant des couples (index, nom_de_fichier) où nom_de_fichier est un nom de fichier ou de répertoire. A sa création, le répertoire contient deux couples dont les noms sont « `.` », le répertoire lui-même, et « `..` », le répertoire père.

Les noms peuvent comporter jusqu'à 255 caractères.

10.1.4 Accès au disque logique

À chaque disque logique est associé un fichier spécial dans le répertoire `/dev` et qui permet d'accéder au disque. Les noms des fichiers spéciaux associés aux disques et partitions sont les suivants :

```
/dev/hda : Hard Disque A : le premier disque IDE
/dev/hdb : Hard Disque B : le deuxième disque IDE
/dev/hda1 : la première partition du disque hda
/dev/hdb3 : la troisième partition du disque hdb
/dev/cdrom : un lien symbolique sur /dev/hdc si votre troi
sième disque IDE est un lecteur de CD ROM
```

10.1.5 La zone swap

Tout système Linux comporte au moins une partition spéciale appelée **partition swap**. Pour des raisons de performances, cette partition n'a pas de structure de *file system*. Elle est utilisée par Linux pour le gestion de la mémoire virtuelle : à un instant donné, la mémoire centrale (RAM) ne contient pas la totalité de la mémoire nécessaire aux processus en cours. Le mécanisme de mémoire virtuelle ne place en RAM que ce qui est indispensable à cet instant, et effectue un va-et-vient entre la RAM et la partition swap.

10.1.6 Commandes ln et mv

Nous l'avons dit précédemment : un fichier, en tant qu'espace disque, est repéré par son index et non par son nom. L'index est le numéro de l'inode décrivant entièrement le fichier sur disque. Un répertoire est un fichier de type particulier contenant des couples (index, nom_de_fichier).

Il est donc possible d'associer deux noms différents au même index. Cette opération est effectuée par la commande `ln` (LiNk) :

ln fichier_existant nouveau_nom

Le même fichier, au sens espace disque, peut alors être accédé par les deux noms : *fichier_existant* et *nouveau_nom*.

Exemples

Dans les exemples suivants, l'option *i* de *ls* permet de visualiser l'index en première colonne :

```
xstra> ls  ilg
6629  rw  r       1 xstra staff 97 Apr 16 17:36 ficha
xstra> cp ficha essai    § copie de ficha sur essai
xstra> ln ficha fichb    § lien entre ficha et fichb
xstra> ls  ilg
6630  rw  r       1 xstra  staff 97 Apr 16 17:42 essai
6629  rw  r       2 xstra  staff 97 Apr 16 17:36 ficha
6629  rw  r       2 xstra  staff 97 Apr 16 17:36 fichb
```

ficha et *fichb* portent le même index alors que *essai* porte un nouvel index. Le fichier *fichb* n'occupe pas de place supplémentaire sur le disque, contrairement au fichier *essai*. *ficha* et *fichb* sont deux noms différents du même fichier disque. Attention : les permissions sont décrites dans l'inode et ne sont donc pas associées à un nom de fichier :

```
xstra> chmod 700 ficha     § change aussi les
                           § permissions de fichb
xstra> ls  ilg
6630  rw  r       1 xstra staff 97 Apr 16 17:42 essai
6629  rwx         2 xstra staff 97 Apr 16 17:36 ficha
6629  rwx         2 xstra staff 97 Apr 16 17:36 fichb
xstra>
```

La commande *rm* (remove) supprime le nom et décrémente le nombre de liens contenus dans l'inode. Le fichier est physiquement supprimé si et seulement si le nombre de liens passe à zéro.

Exemple

```
xstra> rm ficha            § ne supprimera pas fichb
xstra> ls  ilg
6630  rw  r       1 xstra staff 97 Apr 16 17:42 essai
6629  rwx         1 xstra staff 97 Apr 16 17:36 fichb
```

Dans la commande *ln fichier existant nouveau nom*, *nouveau nom* peut se trouver dans un autre répertoire du même *file system*, par exemple :

```
xstra> mkdir test
xstra> ln ficha test/fichierb
```

La commande *mv* (move) remplace le nom du fichier par un nouveau nom :

```
mv fichier_existant nouveau_nom
```

L'inode précédemment repéré par le *nom fichier existant* sera désormais repéré par le nom *nouveau nom*.

```
xstra> mv ficha fichun
xstra> ls  ilg
6630  rw  r       1 xstra staff 97 Apr 16 17:42 essai
6629  rwx         2 xstra staff 97 Apr 16 17:36 fichb
6629  rwx         2 xstra staff 97 Apr 16 17:36 fichun
```

Dans la commande *mv fichier existant nouveau nom*, *nouveau nom* peut se trouver dans un autre répertoire du même *file system*.

Nous avons décrit les commandes *mv* et *ln* telles qu'elles fonctionnent à l'intérieur d'un *file system*. La numérotation des inodes est interne à chaque *file system*. De ce fait, ces deux commandes ne franchissent pas les limites d'un *file system* : deux fichiers différents appartenant à deux *file system* différents peuvent porter le même index (le même numéro d'inode). La logique décrite précédemment est alors inapplicable.

Exemple

```
xstra> mv fichb /tmp/fichierb §/tmp est un autre file system
xstra> ls  ilg
6630  rw r      1 xstra staff  97 Apr 16 17:42 essai
6629  rwx       1 xstra staff  97 Apr 16 17:36 fichun
xstra> ls  ilg /tmp
16    rwx       1 xstra staff   97 Apr 16 17:36 fichierb
```

Nous voyons que :

- il n'y a plus qu'un seul lien sur *fichun* (donc */tmp/fichierb* n'est plus un lien sur *fichun*),
- *fichun* et */tmp/fichierb* ne portent pas le même index.

En fait la commande :

```
mv fichb /tmp/fichierb
```

est exécutée par Linux sous la forme :

```
cp fichb /tmp/fichierb && rm fichb
```

Si la commande *ln* est appliquée à deux fichiers n'étant pas dans le même *file system* on obtiendrait :

```
xstra> ln ficha /tmp/fichb
ln: cannot create hard link '/tmp/fichb' to 'ficha' :
Invalid cross device link
```

La commande `ln` ne franchit pas la limite du *file system*. Il faut dans ce cas utiliser un lien symbolique (option `s` de `ln`) :

```
ln  s fichier_existant nouveau_nom
```

Le fichier `nouveau nom` est créé (nouvel inode), mais n'occupe que peu de place sur disque : il ne contient que la référence à `fichier existant`.

10.2 GESTION DE L'OCCUPATION DISQUE

Le répertoire dans lequel vous vous trouvez après connexion s'appelle le répertoire d'accueil ($HOME). Ce répertoire est dans un *file system* dont la capacité en octets est fixée. La création de fichiers va au fur et à mesure remplir le *file system* jusqu'à ce que le message **file system full** apparaisse. Un *file system* peut être plein pour deux raisons (voir figure 10.2) :

- soit il n'y a plus de place pour un nouvel inode (table des inodes)
- soit il n'y a plus de bloc de données disponible (blocs de données)

Dans le premier cas, le *file system* peut être rempli avec un petit nombre de grands fichiers. Dans le deuxième cas, il peut être rempli avec un très grand nombre de fichiers de taille réduite (ou même nulle). Dans un cas comme dans l'autre, il est impossible de continuer à travailler, et il faudra faire de la place. Il est donc nécessaire de gérer l'espace disque, en veillant à éliminer les fichiers devenus inutiles (*core*, fichiers temporaires,….) et en archivant les fichiers qui ne sont plus utilisés afin de pouvoir les supprimer, ou tout au moins de les conserver sous forme d'une archive compressée. En effet, une archive est **un seul** fichier contenant toute une arborescence, ce qui économise de l'espace dans la table des inodes. Si de plus elle est compressée, il y a aussi économie de blocs de données.

10.2.1 Occupation disque par file system : commande df

La commande `df` indique le nombre de blocs de 1 Ko disponibles par *file system*.

Exemple

```
xstra> df  k
Filesystem  1k blocks    Used  Available  Use% Mounted on
/dev/hda1     8064272 5155608    2499012   68% /
/dev/hda3    68065912  693752   63914572    2% /home
/dev/hdb1    41484300   32828   39344148    1% /data1
xstra>
```

Dans cet exemple, trois *file system* ont été créés par l'administrateur (`/home`, `/data1`). Afin de connaître le *file system* dans lequel l'utilisateur se trouve, il suffit d'utiliser la commande `pwd` qui indiquera le chemin d'accès du répertoire de travail et dont le début correspond au point de montage d'un *file system*.

10.2.2 Occupation disque : commande du

La commande *du noms* affiche le nombre de blocs (1 Ko) occupés par les fichiers et répertoires spécifiés par *noms*. Cette commande est récursive : un nom de répertoire désigne toute l'arborescence sous ce répertoire. Si *noms* n'est pas indiqué, le répertoire de travail (**.**) est pris par défaut.

L'option *s* n'indique que le nombre total de blocs pour chaque *noms*.

L'option *a* (all) liste l'occupation disque de chaque fichier ou sous-répertoire.

Exemples

```
xstra> du  s toto
8  toto
xstra> du  s *
8  toto
4  repertoire1
20  essai
xstra> du  a *
8  toto
4  repertoire1
2  repertoire1/toto
20 essai
xstra>
```

10.2.3 Comment partitionner le disque ?

Il y a beaucoup de réponses différentes à cette simple question, selon l'utilisation du système. La logique est cependant très simple : l'espace disque attribué à un *file system* ne peut pas être utilisé pour une autre *file system*.

- Avantage : si un *file system* est plein suite à une erreur de manipulation ou d'administration, les autres *file system* ne sont pas affectés et le système peut continuer à fonctionner normalement.
- Inconvénient : si un *file system* est plein sans que cela soit le résultat d'une erreur quelconque, il n'est pas possible de l'agrandir en prélevant de l'espace ailleurs.

Quelques règles simples peuvent aider un utilisateur débutant amené à administrer son ordinateur (cet ouvrage ne s'adresse pas aux administrateurs chevronnés) :

- Toujours créer une partition de swap dont la taille est égale à deux fois la taille de la mémoire RAM présente ou prévue dans un avenir proche.
- Pour un système Linux personnel (un utilisateur habituel qui passe root de temps à autre pour administrer le système), la solution la plus simple est d'attribuer tout ce qui reste à une seule partition qui contiendra le root *file system* et donc l'ensemble de l'arborescence en un seul *file system*.
- Pour un système Linux utilisé par un petit groupe, il serait bon, pour simplifier l'administration, de partitionner un peu plus. La quantité de disque utilisée par

chaque utilisateur dépend beaucoup du type de travail effectué par chacun, et il n'est pas possible de donner une règle générale pour */home*. Un partitionnement possible serait par exemple :

une partition swap égale à 2 fois la taille de la RAM
un *file system* pour `/tmp` d'au moins 200 Mo
un *file system* pour `/var` d'au moins 400 Mo
un *file system* pour `/home` (au moins 200 Mo par utilisateur)
tout l'espace restant pour `/` (au moins 3 Go)

10.3 EXERCICES

Exercice 10.3.1

Créez un fichier texte et un lien dur sur ce fichier dans le même répertoire.
Vérifiez que les deux noms correspondent au même inode.
Changez les permissions de l'un et vérifiez que les permissions de l'autre ont suivi.
Modifiez le contenu de l'un et relisez le contenu de l'autre.
Supprimez l'un, que devient l'autre ? Essayez de créer un nouveau lien entre un de ces noms et un nouveau nom dans */tmp*. Expliquez.

Exercice 10.3.2

L'option *-l* de la commande *ls* permet de connaître le nombre de liens sur un fichier régulier (deuxième champ). Curieusement, ce champ n'est pas égal à 1 pour les répertoires, alors qu'il n'est pas possible de créer de liens durs sur des répertoires. Expliquez la valeur de ce nombre sur votre répertoire d'accueil.

Exercice 10.3.3

Créez un fichier texte et un lien symbolique sur ce fichier dans le même répertoire. Quelles sont les permissions du lien symbolique ? Combien de liens possède le fichier texte ? Avec la commande *more*, affichez le contenu du lien symbolique. Effacez le fichier texte. Que devient le lien symbolique ?

Exercice 10.3.4

Le disque est plein de vide ! Écrire un petit script affichant sur la sortie standard la taille cumulée de tous les fichiers réguliers situés sous le répertoire courant, et l'espace disque total occupé par le répertoire courant et tout ce qu'il contient.