Sommaire

INTRODUCTION

Objectif

Être capable de comprendre le fonctionnement d'un système de fichiers

Manipulations

On va travailler dans l'arborescence suivante :

```
$HOME
      |
      - tv
         |
         - TPOS7
```

Créer l'arborescence de répertoires :

```
$ mkdir -p $HOME/tv/TPOS7
```

Se déplacer dans l'arborescence de travail :

```
$ cd $HOME/tv/TPOS7
```

RAPPELS

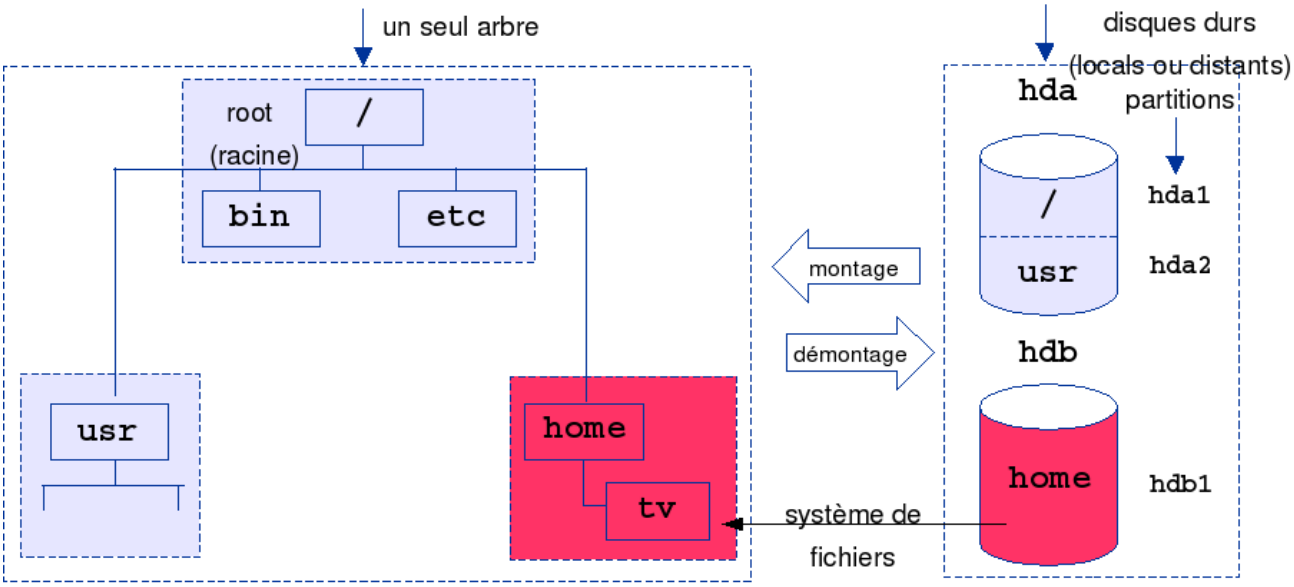
Système de fichiers

Un système de fichiers définit l'organisation d'un disque (ou partition d'un disque). C'est donc une structure de données permettant de stocker les informations et de les organiser dans des fichiers sur ce que l'on appelle des mémoires secondaires (disque dur, disquette, CD-ROM, clé USB, disques SSD, etc.). Il offre à l'utilisateur une vue abstraite sur ses données et permet de les localiser à partir d'un chemin d'accès.

Pour rappel, le fichier est la plus petite entité logique de stockage sur un disque.

Arborescence

Sous Unix/Linux, les utilisateurs voient une arborescence de fichiers unique. Cet arbre est en fait l'unification de plusieurs systèmes de fichiers :



Exemple d'une arborescence unique sous Linux

Dans un système Windows, les périphériques de stockage de données et les partitions sont affichés comme des lecteurs indépendants en haut de leur propre arborescence. Si l'on considère par exemple un système comprenant :

- une partition de disque dur où est installé le système (Windows ou Unix) ;
- une partition de disque dur où se trouvent les données des utilisateurs ;
- un lecteur de disquette.
- un lecteur de cdrom.

Sous Windows, on accédera alors à ces données de manière séparée :

- C: : partition système du disque dur ;
- D: : partition utilisateur du disque dur ;
- A: : disquette (accès-type : A:\chemin\fichier).
- E: : cdrom (accès-type : E:\chemin\fichier).

Sous Unix, l'accès se fera à partir de la racine / :

- / : première partition système du disque dur ;
- /home : partition utilisateur du disque dur ;
- /mnt/floppy : disquette (accès-type : /mnt/floppy/chemin/fichier).
- /mnt/cdrom : cdrom (accès-type : /mnt/cdrom/chemin/fichier).

Le montage et démontage

Le montage d'un système de fichiers consiste à l'attacher à un répertoire (point de montage) d'un système de fichiers déjà actif (l'arbre racine). Cette opération est réalisée par la commande `mount`. Les fichiers et répertoires de ce système de fichiers sont donc accessibles aux utilisateurs (cp, rm, mv, ...).

L'opération de démontage, réalisée par la commande `umount`, détache le système de fichiers de l'arbre racine. Les fichiers et répertoires de ce système de fichiers ne sont donc plus accessibles aux utilisateurs.

Lors du démarrage, le disque qui contient le système de fichiers principal (arbre racine) doit être connu pour être monté automatiquement.

Disque et système de fichiers

L'organisation physique d'un système informatique tient compte des principes suivants :

- Un disque peut contenir plusieurs partitions (commande fdisk).
- On peut créer au plus quatre partitions (soit quatre partitions primaires, soit de 1 à 3 partitions principales puis une partition étendue)
- Une partition étendue contiendra des partitions secondaires (4 max)
- Une partition ne peut contenir qu'un seul système de fichiers.
- Un système de fichiers ne peut s'étendre sur plusieurs disques ou partitions.
- Les disques amovibles possèdent aussi un système de fichiers.

La structure d'un système de fichiers

La structure d'un système de fichiers dépend du système de fichiers utilisé (FAT, NTFS, ext2, . . .).

Dans le cas d'un système de fichiers ext2 (ou ext3), on aura l'organisation suivante (voir cours) :

- le super bloc qui contient les informations clés concernant le système de fichiers
- la table des inodes (la table des descripteurs des fichiers). Chaque fichier physique est identifié de manière unique par un numéro d'inode.
- les répertoires qui assurent une correspondance entre un nom de fichier et un numéro d'inode.
- les fichiers identifié par un numéro d'inode. Un fichier est une suite non ordonnée d'octets.

Remarque : la table des inodes a une taille fixée statiquement à la création du système de fichiers qui fixe donc le nombre maximal de fichiers qu'elle pourra contenir.

L'espace d'un système de fichiers ext2 ou ext3 est découpée en bloc (valeur fixée à la création du système de fichier, par défaut la taille d'un bloc est 4096 octets).

inode

Le terme inode désigne le descripteur d'un fichier. Les inodes (contraction de « index » et « node », en français : nœud d'index) sont des structures de données contenant des informations concernant les fichiers stockés dans certains systèmes de fichiers (notamment de type Linux/Unix). À chaque fichier correspond un numéro d'inode (i-number) dans le système de fichiers dans lequel il réside, unique au périphérique sur lequel il est situé. Il contient les attributs du fichier (affichés notamment pas un ls -l ou la commande stat) et une table d'accès aux blocs de données.

Le numéro d'inode est un entier unique pour le périphérique dans lequel il est stocké. Tous les fichiers, y compris les fichiers spéciaux, sont identifiés par un inode. Le numéro d'inode d'un fichier peut être affiché avec la commande :

```
$ ls -i un_fichier
8131 un_fichier
```

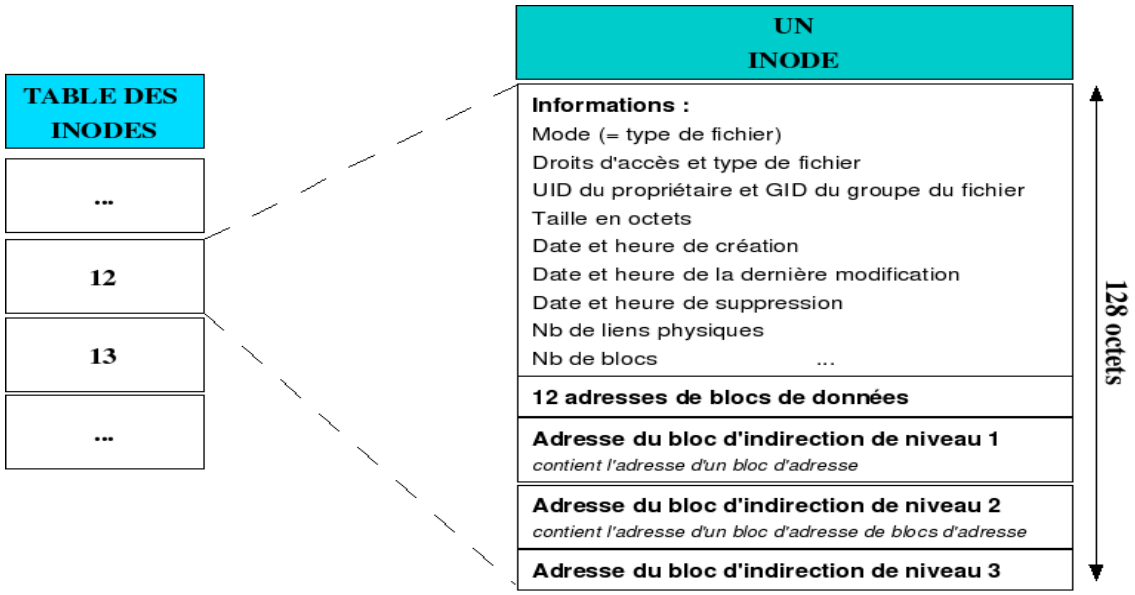
Le standard POSIX s'est basé sur les systèmes de fichiers traditionnels d'Unix. Cette norme impose donc que les fichiers réguliers aient les attributs suivants (« fiche d'identité » d'un fichier identifié par un numéro d'inode) :

- La taille du fichier en octets
- Identifiant du périphérique contenant le fichier
- L'identifiant du propriétaire du fichier
- L'identifiant du groupe auquel appartient le fichier
- Le numéro d'inode qui identifie le fichier dans le système de fichier
- Le mode du fichier qui détermine quel utilisateur peut lire, écrire et exécuter ce fichier
- Horodatage (*timestamp*) pour :
    - La date de dernière modification ctime de l'inode (affichée par la commande stat ou par ls -lc)
    - La date de dernière modification du fichier mtime (affichée par le classique ls -l)
    - La date de dernier accès atime (affichée par la commande stat ou par ls -lu)
- Un compteur indiquant le nombre de liens physiques sur cet inode.

Remarque : les inodes ne contiennent pas les noms de fichier.

Structure d'un inode

Un inode occupera 128 ou 256 octets (taille définie à la création du système de fichiers).



Quelques commandes utiles :

Affiche les informations de base sur un fichier :

```
# ls -il un_fichier
8131 -rw-r--r-- 1 root root 19 2009-10-28 11:26 un_fichier
```

Affiche les informations contenues dans un inode :

```
# stat un_fichier
  File: `un_fichier'
  Size: 19              Blocks: 8          IO Block: 4096   fichier
régulier
Device: 815h/2069d      Inode: 8131        Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2009-10-28 11:26:30.000000000 +0100
Modify: 2009-10-28 11:26:41.000000000 +0100
Change: 2009-10-28 11:26:41.000000000 +0100
```

Affiche le périphérique, la partition et le type de système de fichiers d'un fichier :

```
# df -T un_fichier
Sys. de fich. Type     Tail. Occ. Disp. %Occ. Monté sur
/dev/sdb5     ext3    7,4G  662M  6,3G  10% /
```

Affiche les informations complètes contenues dans un inode :

```
# echo "stat <8131>" | debugfs /dev/sdb5
debugfs 1.41.4 (27-Jan-2009)
debugfs:  Inode: 8131   Type: regular    Mode:  0644   Flags: 0x0
Generation: 944415664    Version: 0x00000000
User:     0   Group:     0   Size: 19
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x4ae81c61 -- Wed Oct 28 11:26:41 2009
atime: 0x4ae81c56 -- Wed Oct 28 11:26:30 2009
mtime: 0x4ae81c61 -- Wed Oct 28 11:26:41 2009
Size of extra inode fields: 4
BLOCKS:
(0):57360
TOTAL: 1
```

Affiche (en hexa et en ASCII) les données contenues dans un bloc :

```
# dd if=/dev/sdb5 bs=4096 skip=57360 count=1 | hexdump -C
1+0 enregistrements lus
1+0 enregistrements écrits
4096 octets (4,1 kB) copiés, 0,0127792 s, 321 kB/s
00000000  4a 65 20 73 75 69 73 20  75 6e 20 66 69 63 68 69  |Je suis un
fichi|
00000010  65 72 0a 00 00 00 00 00  00 00 00 00 00 00 00 00  |
er..............|
```

Affiche en ASCII les données contenues dans un fichier :

```
# cat un_fichier
Je suis un fichier
```

Affiche (en hexa et en ASCII) les données contenues dans un fichier :

```
# hexdump -C un_fichier
00000000  4a 65 20 73 75 69 73 20  75 6e 20 66 69 63 68 69  |Je suis un
fichi|
00000010  65 72 0a                                          |er.|
```

Les différents types de système de fichiers

Il existe de nombreux systèmes de fichiers :

- mimix : le premier FS utilisé par Linux
- ext2/3/4 : le FS standard du système Linux
- msdos : le FS FAT16 de MSDOS et Windows
- vfat : le FS FAT32 de Windows
- smb : le FS réseau utilisant le protocole SMB de Microsoft
- nfs : le FS réseau de Sun
- ntfs : le FS de Windows NT
- iso9660 : le FS utilisé par les CD-ROM

Et des systèmes de fichiers journalisés : ext3, ReiserFS, XFS, ...

Remarques : L'avantage d'un système de fichiers journalisé est de maintenir en permanence la cohérence des métadonnées (données relatives aux structures d'un système de fichiers: emplacement des fichiers, inodes, etc...). Au reboot après un crash (problème matériel, bug du noyau etc), le système va relire le journal, examiner les transactions non terminées, et remettre un système de fichiers sain. En revanche, la journalisation offre peu de garantie quant au contenu des fichiers eux-même. Certains FS supportent les transactions atomiques : une modification sur le disque est effectuée ou non, mais elle n'est jamais effectuée "à moitié".

D'autres systèmes de fichiers : CODA, dtfs, LFS, GFS, PVFS, MOSIX (cf. http://www.inforlad.net/info/computer/linux.htm)

Quelques commandes

```
mkfs et fsck : crée et vérifie un FS (commandes standards UNIX)
mount et umount : monte et démonte un FS
df : indique l'espace libre des disques et partitions
du : indique l'espace occupé par une arborescence
mke2fs, e2fsck, tune2fs : crée, vérifie et paramètre un FS ext2
dumpe2fs : infos sur le super bloc et les groupes de bloc
debugfs : débogue un FS ext2
lsattr et chattr : lister et changer les attributs des fichiers
stat : affiche des informations sur un fichier ou un système de fichier
find : permet de rechercher des fichiers selon différents critères
(notamment par le numéro d'inode avec -inum)
lsof : affiche des informations sur les fichiers ouverts
fuser : identifie les processus utilisant des fichiers
```

Systèmes de fichier ext2/ext3

1) Donner la commande qui permet d'afficher la taille en octets de votre répertoire personnel. Donner le résultat de la commande.

2) Représenter l'arborescence de votre système en ne montrant que les répertoires correspondant à des FS montés et en indiquant leur type.

3) Indiquer le nombre total d'inodes de l'arbre racine (/) et la commande qui permet de l'obtenir.

4) Compléter le tableau suivant pour le système de fichiers racine de votre machine.

le nombre de groupes de blocs	
la taille de la table des inodes	
nb total de fichiers possible	
le nombre de blocs par groupe	
la taille du système de fichiers	
le nombre de copies du *superbloc*	

5) En vous appuyant sur les informations concernant votre système de fichiers racine de votre machine, compléter le tableau suivant.

Le nombre de groupes de blocs ? **Inode count / Inodes per group** =	
Le nombre de blocs de la table des inodes ? (**Inodes per group** x **Inode size**) / **Block size**	
Le nombre de blocs par groupe ? (**Blocks per group**) : Taille de la table d'allocation des blocs (= 1 bloc) x 8 (bits) =	
Le nombre d'inodes (fichiers) par groupe ? **Inode count / Nb de groupes** =	
Le nombre de blocs du dernier groupe ? **Block count % Blocks per group** =	
La taille de ce système de fichiers ? **Block count x Block size** =	

6) En vous appuyant sur les informations fournies par fdisk -l et fdisk -l -u, compléter par le calcul le tableau suivant.

Le nombre total d'octets ? À comparer avec la valeur affichée par fdisk	
Le nombre total de secteurs ? À comparer avec la valeur affichée par fdisk	
Le nombre de secteurs par cylindre ?	
Le nombre total de cylindre ? À comparer avec la valeur affichée par fdisk	

7) Conclure

MONTAGE NFS

Une machine du réseau possède un répertoire `/export` partagé par NFS. On vous demande de monter ce partage sur votre système.

Adresse IP de la machine : 192.168.52.83

Pour cela, créer un répertoire `partage_nfs` dans votre espace personnel. Le répertoire `partage_nfs` représentera le point de montage sur votre système.

1) Monter le répertoire partagé du serveur. Commenter et donner la commande. Avez-vous l'autorisation de faire des montages sous votre compte ?

2) Visualiser le fichier /etc/mtab ? Est-ce que ce montage sera toujours disponible au redémarrage de la machine ? Pourquoi ? Que faudrait-il faire ?

3) Lister le contenu du répertoire partage_nfs. Dans ce répertoire partagé, il y a un programme getNumber.

4) Exécuter ce fichier et donner le numéro affiché par ce programme (ne pas le faire sous le compte root).

Ce programme a créé des fichiers filexxx_yyy.dat (où xxx représente votre UID). Le source du programme getNumber est fourni en Annexe 2.

5) Si on utilise la commande strings ou cat sur le fichier filexxx_littleendian_int.dat, le numéro xxx ne s'affiche pas, pourquoi ? Utiliser la commande od -t x1 -t c ou hexdump -C pour analyser le contenu de ce fichier.

6) Sur combien d'octets est codé un entier (int) ? Sur combien d'octets est codé un entier (short int) ? Sur combien d'octets est codé un caractère (char) ?

7) L'architecture Intel de votre machine fait que le système a inversé les poids forts et poids faibles de cet entier au moment de l'enregistrement (convention little endian). Décoder la valeur hexadécimale de votre fichier. Les conventions big endian et little endian sont expliquées en Annexe 1.

```
$ od -j 19 -t x1 filexxx_littleendian_int.dat   (les 4 octets sont
inversés dans le fichier)
$ od -j 19 -t x4 filexxx_littleendian_int.dat   (comme dans un int)
$ od -j 19 -t d4 filexxx_littleendian_int.dat   (comme dans un int,
affichage décimal)
$ od -j 19 -t dI filexxx_littleendian_int.dat   (I=int)
```

8) Que trouve-t-on dans le fichier filexxx_ascii.dat ? Expliquer la différence avec les autres fichiers (cf. man ascii).

9) Faire un vim /partage_nfs/filexxx.dat et, dans une autre console, tenter de démonter /partage_nfs. Si le démontage est refusé, rechercher avec la commande fuser les processus qui utilise ce montage. Vois aussi la commande lsof.

10) Démonter le répertoire /partage_nfs (en tuant les processus qui l'utilisent si nécessaire), puis consulter le fichier /etc/mtab. Que constatez-vous ?

DÉCODAGE DU MBR

1) Copier le MBR de votre disque dur principal dans un fichier /root/MBR.out.

2) Afficher la table des partitions à partir du fichier précédemment créé.

3) Après le décodage de la question 4, comparer avec les résultats obtenus par les commandes fdisk et cfdisk.

4) Décoder la table des partitions contenues dans le MBR et compléter le tableau ci-dessous :

Adresse relative	Offset MBR	Description	Taille (octets)	Contenu
0x00	0x1BE	**Etat de la partition** : 00 : non active ou 80 : active	1	
0x01	0x1BF	**N° de tête** où commence la partition	1	
0x02	0x1C0	**N° de secteur** et **cylindre** où commence la partition	2	
0x04	0x1C2	**Type de partition**	1	
0x05	0x1C3	**N° de tête** où finit la partition	1	
0x06	0x1C4	**N° de secteur** et **cylindre** où finit la partition	2	
0x08	0x1C6	**Distance** en secteurs entre secteur de partition et secteur de boot de la partition	4	
0x0C	0x1CA	**Taille** de la partition en **nombre** de secteurs	4	
0x00	0x1CE	**Etat de la partition** : 00 : non active ou 80 : active	1	
0x01	0x1CF	**N° de tête** où commence la partition	1	
0x02	0x1D0	**N° de secteur** et **cylindre** où commence la partition	2	
0x04	0x1D2	**Type de partition**	1	
0x05	0x1D3	**N° de tête** où finit la partition	1	
0x06	0x1D4	**N° de secteur** et **cylindre** où finit la partition	2	
0x08	0x1D6	**Distance** en secteurs entre ...	4	
0x0C	0x1DA	**Taille** de la partition en **nombre** de secteurs	4	
0x00	0x1DE	**Etat de la partition** : 00 : non active ou 80 : active	1	
0x01	0x1DF	**N° de tête** où commence la partition	1	
0x02	0x1E0	**N° de secteur** et **cylindre** où commence la partition	2	
0x04	0x1E2	**Type de partition**	1	
0x05	0x1E3	**N° de tête** où finit la partition	1	
0x06	0x1E4	**N° de secteur** et **cylindre** où finit la partition	2	
0x08	0x1E6	**Distance** en secteurs entre ...	4	
0x0C	0x1EA	**Taille** de la partition en **nombre** de secteurs	4	
0x00	0x1EE	**Etat de la partition** : 00 : non active ou 80 : active	1	
0x01	0x1EF	**N° de tête** où commence la partition	1	
0x02	0x1F0	**N° de secteur** et **cylindre** où commence la partition	2	
0x04	0x1F2	**Type de partition**	1	
0x05	0x1F3	**N° de tête** où finit la partition	1	
0x06	0x1F4	**N° de secteur** et **cylindre** où finit la partition	2	
0x08	0x1F6	**Distance** en secteurs entre ...	4	
0x0C	0x1FA	**Taille** de la partition en **nombre** de secteurs	4	

Entrée de Répertoire

Placez-vous dans le répertoire /tmp sous root et créer deux fichiers de la manière suivante :

```
# touch /tmp/un_fichier
# ln -s /tmp/un_fichier /tmp/un_lien_sur_un_fichier
```

1) Afficher le numéro d'inode du répertoire /tmp et des deux fichiers créés. Les convertir en héxadécimal.

2) Afficher les informations sur la partition contenant le répertoire /tmp.

3) Afficher les informations partielles du groupe 0 de la partition contenant le répertoire /tmp.

A partir des informations obtenues, afficher l'entrée de répertoire de la racine en complétant la ligne de commande ci-dessous (remplacer xxx et yyy) :

```
# dd if=/dev/xxx bs=4096 skip=yyy count=1 | hexdump -C
```

4) Déterminer le numéro de bloc indiqué dans l'inode du répertoire tmp (remplacer xxx et zzzz étant le numéro d'inode) :

```
# echo "stat <zzzz>" | debugfs /dev/xxx
```

5) Afficher l'entrée de répertoire du répertoire tmp (remplacer xxx et nnnnn étant le numéro de bloc) :

```
# dd if=/dev/xxx bs=4096 skip=nnnnn count=1 | hexdump -C
```

6) Décoder l'entrée de répertoire de /tmp pour les deux fichiers créés plus haut :

Fichier	Champ	Taille (octets)	Contenu
`un_fichier`	n° d'inode	4	
	taille en octets de l'entrée	2	
	nombre de caractères du nom de fichier	1	
	type de fichier	1	
	nom du fichier (en ASCII)	Max. 255	
`un_lien_sur_un_fichier`	n° d'inode	4	
	taille en octets de l'entrée	2	
	nombre de caractères du nom de fichier	1	
	type de fichier	1	
	nom du fichier (en ASCII)	Max. 255	

EFFACER DES FICHIERS

Effacement ?

Mettre un fichier à la poubelle : ce n'est pas un effacement, (presque) rien n'a changé !

Effacer un fichier = modifier les méta-données qui pointent sur les blocs ou sont stockés le fichier (les blocs de données sont intacts)

Suppression des partitions (fdisk, parted) : un seul bloc modifié sur tout le disque

Reformatage d'une partition («format c:», «mkfs /dev/hda1»...) : Généralement, lecture de tous les blocs pour savoir s'ils sont valides mais écriture de quelques blocs seulement. Tous les reformatages ne ré-écrivent pas tous les blocs de la partition

Techniques d'effacement de données (efficaces)

Effacement sécurisé par démagnétisation (dégausseur) : destruction du support magnétique

Effacement par ré-écriture : Exemple du standard DoD 5220-22.M du Département de la Défense Américaine, en plusieurs passes : la première avec un caractère fixe, la seconde avec son complément et la troisième avec des données aléatoires. Guttman method : ré-écriture 35 fois avec des motifs différents

Logiciels : dban (Darik's Boot and Nuke), wipe, shred, srm, ... commande unix simple (dd, voir le tp)

Effacer un fichier

Créer un fichier :

```
# echo "Je suis un fichier" > /tmp/un_fichier
```

Afficher son numéro d'inode :

```
# ls -il /tmp/un_fichier
8131 -rw-rw-r-- 1 tv   tv     19 2009-10-28 11:26 un_fichier
```

Afficher le contenu de l'inode (pour trouver le bloc de donnée) :

```
# echo "stat <8131>" | debugfs /dev/sdb5
debugfs:  Inode: 8131   Type: regular    Mode:  0664   Flags: 0x0
...
BLOCKS:
(0):47104
TOTAL: 1
```

Afficher le contenu du bloc de donnée associé à l'inode du fichier :

```
# dd if=/dev/sdb5 bs=4096 skip=47104 count=1 | hexdump -C
00000000  4a 65 20 73 75 69 73 20  75 6e 20 66 69 63 68 69  |Je suis un
fichi|
00000010  65 72 0a 00 00 00 00 00  00 00 00 00 00 00 00 00  |
er..............|
```

Effacer le fichier :

```
# rm -f un_fichier
```

Afficher le contenu de l'inode (pour vérifier que le bloc de données a été libéré) :

```
# echo "stat <8131>" | debugfs /dev/sdb5
debugfs:  Inode: 8131   Type: regular    Mode:  0664   Flags: 0x0
ctime: 0x4af03b71 -- Tue Nov  3 15:17:21 2009
atime: 0x4ae87741 -- Wed Oct 28 17:54:25 2009
mtime: 0x4af03b71 -- Tue Nov  3 15:17:21 2009
dtime: 0x4af03b71 -- Tue Nov  3 15:17:21 2009
Size of extra inode fields: 4
BLOCKS:
```

Afficher le contenu du bloc de donnée associé à l'inode du fichier effacé (les données n'ont pas été effacées) :

```
# dd if=/dev/sdb5 bs=4096 skip=47104 count=1 | hexdump -C
00000000  4a 65 20 73 75 69 73 20  75 6e 20 66 69 63 68 69  |Je suis un
fichi|
00000010  65 72 0a 00 00 00 00 00  00 00 00 00 00 00 00 00  |
er..............|
```

Créer un nouveau fichier :

```
# touch un_autre_fichier
```

Afficher son numéro d'inode (le numéro précédent a été réattribué) :

```
# ls -il
8131 -rw-r--r-- 1 root root    0 2009-11-03 15:20 un_autre_fichier
```

Afficher le contenu de l'inode (pour vérifier que l'inode a été mise à jour) :

```
# echo "stat <8131>" | debugfs /dev/sdb5
debugfs 1.41.4 (27-Jan-2009)
debugfs:  Inode: 8131   Type: regular    Mode:  0644   Flags: 0x0
Generation: 944445678    Version: 0x00000000
User:     0   Group:     0   Size: 0
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 0
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x4af03c45 -- Tue Nov  3 15:20:53 2009
atime: 0x4af03c45 -- Tue Nov  3 15:20:53 2009
mtime: 0x4af03c45 -- Tue Nov  3 15:20:53 2009
Size of extra inode fields: 4
BLOCKS:
```

Afficher le contenu du bloc de donnée associé à l'inode du fichier (les données n'ont toujours pas été effacées) :

```
# dd if=/dev/sdb5 bs=4096 skip=47104 count=1 | hexdump -C
00000000  4a 65 20 73 75 69 73 20  75 6e 20 66 69 63 68 69  |Je suis un
fichi|
00000010  65 72 0a 00 00 00 00 00  00 00 00 00 00 00 00 00  |
er..............|
```

Écrire dans le fichier :

```
# echo "Je suis un autre fichier" > /tmp/un_autre_fichier
```

Afficher le contenu de l'inode (logiquement, le système a alloué l'ancien bloc libéré précédemment) :

```
# echo "stat <8131>" | debugfs /dev/sdb5
debugfs 1.41.4 (27-Jan-2009)
debugfs:  Inode: 8131   Type: regular    Mode:  0644   Flags: 0x0
...
BLOCKS:
(0):47104
TOTAL: 1
```

Afficher le contenu du fichier :

```
# hexdump -C /tmp/un_autre_fichier
00000000  4a 65 20 73 75 69 73 20  75 6e 20 61 75 74 72 65  |Je suis un
autre|
00000010  20 66 69 63 68 69 65 72  0a                       | fichier.|
```

Méthode pour effacer un fichier (à répéter plusieurs fois pour plus d'efficacité)

Récupérer la taille du fichier à effacer :

```
# ls -l /tmp/un_autre_fichier
-rw-r--r-- 1 root root 25 2009-11-03 15:24 un_autre_fichier
```

Écrire des données aléatoires dans le fichier à effacer :

```
# dd if=/dev/urandom of=/tmp/un_autre_fichier bs=25 count=1 conv=notrunc
1+0 enregistrements lus
1+0 enregistrements écrits
25 octets (25 B) copiés, 0,000156269 s, 160 kB/s

ou : shred /tmp/un_autre_fichier
```

Vérification :

```
# hexdump -C /tmp/un_autre_fichier
00000000  f8 e6 74 41 84 82 8a 13  78 a6 a5 33 5a 1c f7 2f
|..tA....x..3Z../|
00000010  bb 9a 73 ac 05 38 f6 9a  e6                       |..s..8...|
```

Effacer le fichier :

```
# rm -f /tmp/un_autre_fichier

ou : shred -u /tmp/un_autre_fichier
```

Rechercher des traces dans la mémoire

Créer un fichier puis l'effacer :

```
# echo "halloween" > /tmp/un_autre_fichier
# rm -f /tmp/un_autre_fichier
```

Rechercher dans la mémoire :

```
# dd if=/dev/mem | hexdump -C | grep 'halloween'
```

Remarque : la commande sync vide les tampons disques (force l'écriture des blocs modifiés sur disque et met à jour le super-bloc).

Disque virtuel et Systèmes de fichiers temporaires

ramdisk

Un disque virtuel (ramdisk) est une technique qui permet d'émuler un disque dur à partir d'un espace alloué en mémoire centrale. Les temps d'accès sont extrêmement rapides ; en revanche, leur capacité ne peut excéder la taille de la mémoire centrale et les données seront perdues si la mémoire n'est plus alimentée électriquement.

Initialiser la mémoire (taille : 2048 x 1024 octets)

```
# dd if=/dev/zero of=/dev/ram0 bs=1k count=2048
2048+0 enregistrements lus 2048+0 enregistrements écrits
2097152 octets (2,1 MB) copiés, 0,0214266 s, 97,9 MB/s
```

Créer un point de montage :

```
# mkdir /mnt/ramdisk
```

Créer un système de fichiers ext2 :

```
# mke2fs -v -m 0 /dev/ram0 2048
mke2fs 1.41.4 (27-Jan-2009)
Type de système d'exploitation : Linux
Taille de bloc=1024 (log=0)
256 i-noeuds, 2048 blocs
Premier bloc de données=1
Nombre maximum de blocs du système de fichiers=2097152
1 groupe de bloc
8192 blocs par groupe, 8192 fragments par groupe
256 i-noeuds par groupe
```

Monter le ramdisk :

```
# mount -t ext2 /dev/ram0 /mnt/ramdisk
```

Afficher les informatons de taille :

```
# df -h /mnt/ramdisk
Sys. de fich.         Tail. Occ. Disp. %Occ. Monté sur
/dev/ram0             2,0M   21K  2,0M   2% /mnt/ramdisk
```

Maintenant, on peut démonter le ramdisk :

```
# umount /mnt/ramdisk
```

Il existe d'autres techniques pour obtenir des systèmes de fichiers temporaires : tmpfs, ramfs, ...

ramfs

ramfs est un système de fichiers temporaire monté en RAM très simple.

Il utilise de manière détournée le mécanisme de gestion de cache du noyau Linux en précisant à ce dernier que les pages mémoire concernées n'ont aucune destination en espace de stockage persistant : le cache ne peut donc jamais les écrire en dur, et les garde indéfiniment en mémoire volatile.

Le système ramfs est donc extrêmement simple car il s'appuie sur des mécanismes existants. Ceci par opposition à un ramdisk, fondé sur la simulation d'un disque physique de taille fixe, imposant un formatage par un système de fichier conventionnel, qui provoque de multiples copies mémoires inutiles.

Il a pour défaut de ne pas limiter la taille mémoire allouable, et est donc réservé à l'utilisateur root. Le système tmpfs ajoute ce contrôle de taille maximum pour permettre de sécuriser l'utilisation de ce type de points de montage.

Avec cette implémentation, on passe de la simulation d'un périphérique (ramdisk) à celle d'un système de fichier (ramfs), ce qui a pour avantage entre autre d'utiliser moins de mémoire. ramfs est intégré par défaut dans tous les noyaux 2.6, l'utilisation est simple :

```
# mount -t ramfs none /mnt/ramdisk

# df -a /mnt/ramdisk
Sys. de fich.         Tail. Occ. Disp. %Occ. Monté sur
none                     0     0     0   -  /mnt/ramdisk
```

Vous remarquerez que le répertoire monté n'a pas d'information sur sa taille.

Maintenant, on peut démonter le système de fichiers temporaire (et tout perdre !) :

```
# umount /mnt/ramdisk
```

tmpfs

TmpFS (*Temporary File System*) est le nom générique donné à tout système de fichiers Unix temporaire. Tout fichier créé dans un tel système de fichiers disparaît lors de l'arrêt du système. L'implémentation par défaut du tmpfs des noyaux Linux 2.6.x se base sur ramfs qui utilise le mécanisme de cache pour optimiser la gestion de la mémoire.

Cependant, tmpfs propose en plus par sécurité une limite de taille mémoire allouable fixée au moment du montage et modifiable à la volée avec l'option "remount". Tmpfs permet par ailleurs au système d'utiliser le swap lorsque cela devient nécessaire, ce qui est une garantie supplémentaire. Par opposition à un RAM Disque, il alloue dynamiquement la mémoire de manière à ne pas l'utiliser en excès, et offre de meilleures performances grâce à son extrême simplicité.

```
# mount -t tmpfs none /mnt/ramdisk -o size=100m

# df -a /mnt/ramdisk
Sys. de fich.         Tail. Occ. Disp. %Occ. Monté sur
none                  100M     0  100M   0% /mnt/ramdisk
```

Remarque : Les permissions par défaut du répertoire monté sont 1777 (pareil que /tmp), vous pouvez les modifier soit avec la commande mount (avant de le monter avec l'option "mode") ou tout simplement avec la commande "chmod".

On peut démonter le système de fichiers temporaire (et tout perdre !) :

```
# umount /mnt/ramdisk
```

Afficher les quantités de mémoires libres et utilisées :

```
# free -b
# free -m
             total       used       free     shared    buffers
cached
Mem:          1009        991         18          0          9
589
-/+ buffers/cache:        391        617
Swap:         2153          2       2150

# free -t
             total       used       free     shared    buffers
cached
Mem:       1033960    1016868      17092          0       9816
604180
-/+ buffers/cache:     402872     631088
Swap:      2205076       2624    2202452
Total:     3239036    1019492    2219544
```

Le périphérique loop et le système de fichiers ISO 9660

ISO 9660 est une norme de l'ISO, qui définit le système de fichiers utilisé sur les CD-ROM. Ses objectifs sont de supporter de nombreux systèmes d'exploitation comme Windows ou Mac OS, ainsi que les systèmes qui permettent la spécification Unix. Ce système de fichier est également utilisé sur les DVD-ROM.

Il existe des extensions à cette norme :

- Rockridge est une extension du format ISO 9660 auquel s'ajoute la sémantique des systèmes de fichiers POSIX. Ce format est utilisé par défaut sous les systèmes Unix pour les CD ROM. Il permet : le support de nom de fichier de plus de 255 caractères, peu de restriction au niveau du choix des caractères, les droits sur les fichiers selon les utilisateurs et les groupes dans le style UNIX, les liens symboliques et une hiérarchie plus profonde des fichiers ;

- El Torito (de son nom anglais complet El Torito Bootable CD Specification) est une extension à la norme ISO 9660 sur les CD-ROM décrivant comment un ordinateur peut démarrer (ou booter) à partir d'un CD-ROM ;

- Joliet est une extension de la norme ISO 9660 qui s'applique aux CD-ROM. Cette norme permet d'enregistrer des fichiers dont les noms sont composés de 64 caractères au maximum. À l'origine utilisée sur le système d'exploitation Windows 95, elle est désormais répandue et utilisée par la majorité des systèmes d'exploitation ;

On peut également sous les Unix modernes monter des fichiers qui constituent un système de fichiers à eux-seuls (*loopback*), grâce à l'option -loop. Ceci est particulièrement utile dans le cas d'images représentant des disquettes, CDROMs, DVDs.

Créer un point de montage :

```
# mkdir /mnt/iso
```

Monter l'image ISO (elle est disponible sur le serveur) :

```
# mount -o loop -t iso9660 image.iso /mnt/iso
```

Afficher son contenu :

```
# ls -l /mnt/iso/

# df -Th /mnt/iso/
Sys. de fich.   Type     Tail. Occ. Disp. %Occ. Monté sur
/tmp/image.iso  iso9660   15M  15M     0  100%  /mnt/iso
```

Les commandes dd et mkisofs peuvent aider à fabriquer de tels fichiers. mkisofs est un utilitaire en mode console qui permet de créer des images CD ou DVD au format ISO. Mkisofs est disponible sous Linux, BSD et Windows.

L'image utilisée ici a été crée comme ceci (les fichiers étant dans le répertoire tpOS) :

```
# mkisofs -V "TP-OS" -iso-level 3 -o image.iso tpOS
```

Conseil : il est préférable de créer un fichier « .iso » pour deux raisons :

- respecter la norme, afin que le fichier soit lisible par tous (tout le monde n'utilise pas Nero, ni même Windows) ;
- éviter le surplus d'informations inutiles (comme par exemple dans le format propriétaire de Nero se terminant par .nrg).

Remarque : pour graver les images iso, on pourra ensuite utiliser cdrecord.

ANNEXE 1 : ENDIANNESS

En informatique, certaines données telles que les nombres entiers peuvent être représentées sur plusieurs octets. L'ordre dans lequel ces octets sont organisés en mémoire ou dans une communication est appelé *endianness* (mot anglais traduit par « boutisme »).

De la même manière que certains langages humains s'écrivent de gauche à droite, et d'autres s'écrivent de droite à gauche, il existe une alternative majeure à l'organisation des octets représentant une donnée : l'orientation big-endian et l'orientation little-endian.

Remarque : Les termes big-endian et little-endian ont été empruntés aux Voyages de Gulliver de Jonathan Swift, roman dans lequel deux clans de Lilliputiens se font la guerre à cause de la manière différente qu'ils ont de casser les œufs à la coque : par le gros ou le petit bout.

Big endian (unité l'octet)

Quand certains ordinateurs enregistrent un entier sur 32 bits en mémoire, par exemple 0xA0B70708 en notation hexadécimale, ils l'enregistrent dans des octets dans l'ordre qui suit : A0 B7 07 08 (dans l'ordre, octet de poids le plus fort vers l'octet de poids le plus faible).

Les architectures qui respectent cette règle sont dites big-endian (ou mot de poids fort en tête), par exemple les processeurs Motorola 68000, les SPARC (Sun Microsystems) ou encore les protocoles internet TCP/IP.

Little endian (unité l'octet)

Les autres ordinateurs enregistrent 0xA0B70708 dans l'ordre suivant : 08 07 B7 A0, c'est-à-dire avec l'octet de poids le plus faible en premier.

De telles architectures sont dites little-endian ou mot de poids faible en tête (par exemple, les processeurs x86, qui se trouvent dans les PC).

On a bien compris que ces conventions posent des problèmes dans le portage des logiciels. Par exemple, en lisant des données binaires, selon l'architecture, on ne va pas obtenir la même donnée après lecture si on ne se soucie pas de la convention. Sur les architectures i386, l'ordre des octets de l'hôte est LSB (« Least Significant Byte first ») ou Little endian, c'est-à-dire octet de poids faible en premier, alors que sur les réseaux, notamment l'Internet, l'ordre est MSB (« Most Significant Byte first ») ou Big endian, octet de poids fort en premier.

Les fonctions en langage C htonl, htons, ntohl, ntohs permettent ces conversions d'ordre des octets (un hôte vers un réseau : h to n, et un réseau

vers un hôte : ntoh).

Annexe 2 : Source du programme getNumber

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <arpa/inet.h>

int main()
{
   FILE *fichier; char fileName[48];
   uid_t uid; short int UID;

   uid = getuid();
   if(uid == 0)
   {  printf("Vous etes root ! veuillez recommencer sous votre compte
utilisateur\n\n");
      return 1;
   }

   printf("Votre uid est    : %d (0x%04X)\n", uid, uid);
   printf("Big endian       : %d -> 0x%08X\n", uid, uid);
   printf("Little endian    : %d -> 0x%08X\n\n", uid, htonl(uid));
   printf("Taille int       : %d octets\n", sizeof(int));
   printf("Taille short int : %d octets\n", sizeof(short int));
   printf("Taille char      : %d octet\n\n", sizeof(char));

   sprintf(fileName, "./file%d_littleendian_int.dat", uid);
   fichier = fopen(fileName, "w+");
   if(fichier)
   {  fprintf(fichier, "Votre numero est :\n");
      fwrite(&uid, sizeof(uid), 1, fichier);
      fclose(fichier);
      printf("fichier %s : ok !\n", fileName);
   }
   else
   {  printf("Impossible de creer le fichier %s !\n", fileName);
      perror("fopen"); exit(1);
   }
```

```
   sprintf(fileName, "./file%d_littleendian_shint.dat", uid);
   UID = (short int)uid;
   fichier = fopen(fileName, "w+");
   if(fichier)
   {  fprintf(fichier, "Votre numero est :\n");
      fwrite(&UID, sizeof(UID), 1, fichier);
      fclose(fichier);
      printf("fichier %s : ok !\n", fileName);
   }
   else
   {  printf("Impossible de creer le fichier %s !\n", fileName);
      perror("fopen"); exit(1);
   }

   sprintf(fileName, "./file%d_bigendian_shint.dat", uid);
   UID = (short int)uid;
   UID = htons(UID);
   fichier = fopen(fileName, "w+");
   if(fichier)
   {  fprintf(fichier, "Votre numero est :\n");
      fwrite(&UID, sizeof(UID), 1, fichier);
      fclose(fichier);
      printf("fichier %s : ok !\n", fileName);
   }
   else
   {  printf("Impossible de creer le fichier %s !\n", fileName);
      perror("fopen"); exit(1);
   }


   sprintf(fileName, "./file%d_ascii.dat", uid);
   fichier = fopen(fileName, "w+");
   if(fichier)
   {
      fprintf(fichier, "Votre numero est :\n%d", uid);
      fclose(fichier);
      printf("fichier %s : ok !\n", fileName);
   }
   else
   {
      printf("Impossible de creer le fichier %s !\n", fileName);
      perror("fopen");
      exit(1);
   }

   printf("\nVous pouvez visualiser les fichiers avec la commande : od
ou hexdump -C\n");
   exit(0);
}
```

ANNEXE 3 : SAUVEGARDE DE PARTITIONS

Partimage

Partimage est un utilitaire Linux permettant la sauvegarde des partition du disque dur utilisant un système de fichier supporté dans un fichier image. L'image peut être compressée au format gzip ou bzip2 afin d'économiser de l'espace disque ou scindée en de multiples fichiers pour être copiée sur des médias amovibles comme des CDs / DVDs. Depuis la version 0.6.0, il est possible de déporter le fichier image au travers d'un réseau, mais il est aussi possible de sauvegarder/restaurer l'image par le réseau grace à Samba ou NFS. Si vous ne voulez pas avoir à installer Partimage, on vous conseille d'utiliser SystemRescueCd. C'est un livecd conçu par l'équipe de Partimage qui vous permettra d'utiliser Partimage sur un ordinateur sans système d'exploitation ou qui n'est pas correctement installé. Il sert aussi a sauver une image sur un DVD a la volée.

Partition Image ne prend que les données présentes dans les partitions afin de gagner en rapidité et en efficacité les "blocs" non utilisés ne sont pas sauvegardés dans le fichier image contrairement à l'utilitaire 'dd'. Partition Image gère parfaitement de grandes partitions y compris celles qui sont entièrement remplies de données.

Pour plus de sécurité le fichier image peut être gravé sur un CDR/RW de même si vous ne voulez pas prendre de l'espace disque. Il peut être utilisé afin d'installer (cloner) des ordinateurs identiques. Si vous avez par exemple 50 ordinateurs identiques à installer et que vous souhaitez y installer le même système d'exploitation, vous gagnerez beaucoup de temps puisque vous n'aurez qu'à installer la première machine puis en faire une image. Pour les 49 autres machines il ne restera qu'à utiliser cette image afin de restaurer le disque.

Remarque : le système de fichiers NTFS n'est pas actuellement entièrement garanti. La sauvegarde d'une partition NTFS devrait être possible si la partition originale n'est pas trop fragmentée et si elle n'est pas compressée. Avec ces restrictions il devrait être possible de prendre une image de la partition et la restaurer ultérieurement. Si un problème survient lors de la phase de sauvegarde un message d'erreur apparaît et la sauvegarde sera annulée. Si la phase de sauvegarde a fonctionné la restauration devrait se passer sans problèmes (sauf présence de bugs). La façon la plus simple de vérifier si Partition Image est utilisable est d'essayer de lancer une sauvegarde, si celle-ci n'aboutit pas on pourra essayer de défragmenter la partition avec un outil du type "diskeeper" et essayer à nouveau de faire une image.

Site : http://www.partimage.org/

Autres outils

g4u - Harddisk Image Cloning for Pcs : http://www.feyrer.de/g4u/

CloneZilla : http://www.clonezilla.org/

Norton Ghost : http://www.norton.com/