Systèmes d'exploitation : le système de gestion de fichiers (structures de données implantées)

Licence MIAGE — Université Lille 1
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Semestre 6 — 2012-2013

Notion de bloc physique

Un disque dur est composé de secteurs (contenant de 512 à 16384 octets), de cylindres (contenant de 20 à 200 secteurs) et de plateaux (contenant de 800 à 4000 cylindres par exemple).

On ne manipule (lecture, etc.) pas les octets individuellement mais on utilise une abstraction : le bloc physique composé de plusieurs octets. Comment structurer l'information fichier ? Deux organisations :

1. organisation contiguë : on doit déplacer le fichier chaque fois que sa taille augmente ;
2. organisation non contiguë : le fichier est divisé en blocs et on doit
   - déterminer la taille optimale des blocs ;
   - gérer les espaces libres ;
   - représenter l'ensemble des blocs constituant un fichier.

La taille de bloc utilisée est d'environs 1Ko afin d'optimiser

- l'espace disque occupé (un bloc de 32Ko $\equiv$ 97% de perte) ;
- le temps d'accès aux données.

Organisation de l'espace disque libre

L'OS fournit un niveau d'abstraction machine qui associe à un bloc physique de stockage sur le disque à une adresse i.e. un numéro.
Le disque peut être ainsi vue comme une suite linéaire d'adresses
On veut maintenant avoir des fonctions de bas niveau du type :

- fputc, fputs, etc. — output of characters and strings
- fgetc, fgets, etc. — input of characters and strings

qui prennent en entrée une adresse disque et manipule le bloc correspondant.

Il nous faut représenter l'espace libre, les fichiers et les répertoires — dont les blocs n'ont aucunes raisons d'être contiguës — dans le disque vu comme une suite linéaire d'adresses.
Plusieurs structures de données peuvent être utilisées :

table liste arbre

Trois représentations possibles de l'espace disque libre

Supposons des blocs physiques de 1Ko avec des adresses codées sur 2 octets.

1. FreeList : les blocs libres forment une vaste liste chaînées (les 2 derniers octets du bloc courant formant l'adresse du bloc libre suivant par exemple) ;
2. Liste chaînée de blocs d'adresses : chaque bloc d'adresses contient :
   - 511 adresses chacune associée à un bloc libre ;
   - un numéro pour représenter le bloc d'adresse suivant dans la chaîne.
3. Table de bits (bitmap)
   - à chaque bloc du disque est associé un bits (0 s'il est libre, 1 s'il est occupé) ;
   - la table est plus concise que la liste.

Pour un disque de 20Mo (chaque bloc faisant 1Ko), les différentes représentations des blocs libres occupent :

1 : 20000 blocs 2 : 40 blocs 3 : 3 blocs.

File Allocation Table

Pour structurer l'ensemble des blocs constituant un fichier, il faut :

- stocker l'ensemble des *adresses physiques* — numéro de bloc du disque — constituant le fichier ;
- en faisant la correspondance entre ces adresses physiques et les *adresses logiques* — position dans le fichier — des blocs constituant le fichiers.

Pour ce faire, on peut utiliser plusieurs types de représentations simples (liste et table) :

1. Liste chaînés : chaque bloc contient 1Ko de données moins 2 octets qui représentent un pointeur sur le bloc suivant du fichier. Mais, l'accès aléatoire est difficile à mettre en œuvre car il faut parcourir les blocs de la liste pour accéder aux suivants.
2. Table d'allocation des fichiers (File Allocation Table) a autant de cellules qu'il y a de bloc sur le support. Ce format a été le standard Microsoft jusqu'à l'adoption de NTFS.

Principe de la FAT

Dans une FAT, chaque cellule contient :

- soit l'adresse du bloc suivant du fichier, soit un code End Of File ;
- soit un code indiquant que le bloc correspondant est libre (L), ou endommagés (E).

Un exemple de FAT contenant deux fichiers :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		6	3	10	9	7	14	L	11	13	EOF	E	EOF	4

La FAT code la représentation de tous les fichiers pas celle de l'arborescences du FS.

Remarque sur ce type de représentation

Ce type de structuration est utilisable pour des disques de petite capacité mais ils est moins bon pour des grandes : l'ensemble des fichiers sur le volume dépendent de la même FAT ce qui augmente les risques d'endomagement car les blocs la contenant sont fortement sollicités (de même, il faut manipuler toute la FAT pour un seul fichier).

Ce modèle n'implante pas de droits portant sur les fichiers car si c'était le cas, chaque bloc — (cf. la représentation arborescente de l'ensemble des fichiers) devrait contenir l'information et la taille de la FAT en serait multipliée. Ce modèle a été abandonné (mais continu a être utilisé pour les clefs USB et les disquettes).

La taille maximale des fichiers dépend directement de la taille du disque (la structuration en FAT ne limite pas la taille du fichier).

Représentation arborescente des fichiers

Un *nœud d'information* — aussi appelé un *inœud* — est une structure de données qui

- contient des informations sur le fichier ;
- représente la liste des blocs physiques du fichier à l'aide d'un arbre.

À chaque fichier, l'OS associe un inœud constitué d'un bloc sur le disque et qui contient des informations comme :

- le numéro du nœud. Chaque inœud est connu de l'OS par ce numéro (son *inombre*).
- le type de fichier indexé, le nombre de liens sur ce fichier ;
- Uid du propriétaire, Gid du propriétaire ;
- les droits concernant ce fichier ;
- la taille du fichier, les dates de création, de dernière modification, etc.

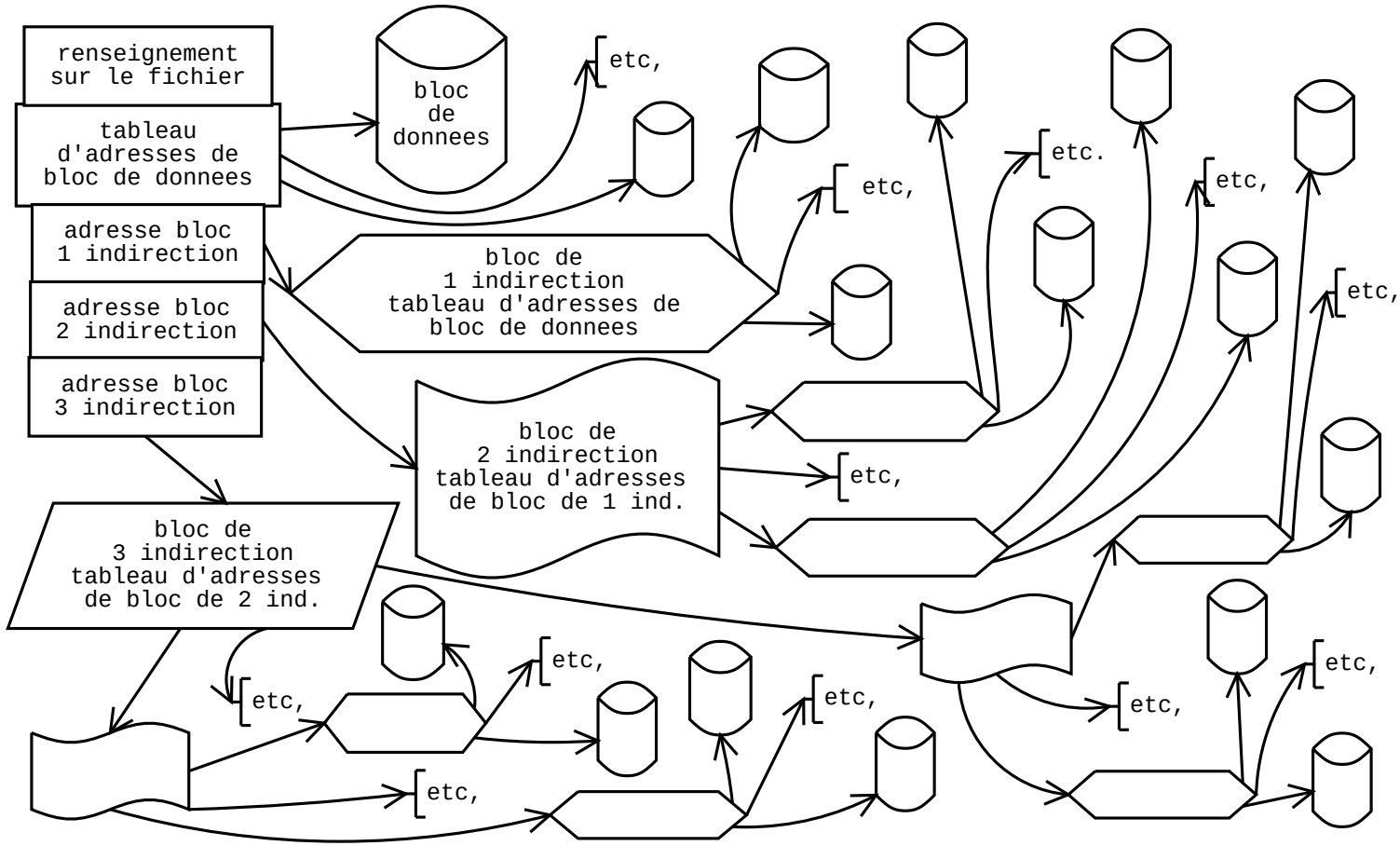
Chaque inœud est désigné par un unique *inombre*.

De plus, l'inœud implante dynamiquement l'arborescence suivante :

- un tableau de x adresses de blocs de données dans l'inœud. Ces blocs du disque constituant le fichier sont directement adressés depuis l'inœud ;
- l'adresse d'un bloc physique — dit de 1 indirection — contenant un tableau d'adresses de blocs de données. Lorsqu'un fichier dépasse x blocs, le FS attribue à l'inœud un nouveau bloc du disque pour contenir un tableau d'adresses de blocs constituant le fichier.
- l'adresse d'un bloc physique — dit de 2 indirection — contenant un tableau d'adresses de blocs de 1 indirection.
- l'adresse d'un bloc physique — dit de 3 indirection — contenant un tableau d'adresses de blocs de 2 indirection. Cette triple indirection est rarement utilisée. Si cela s'avère insuffisant, ce processus se répète récursivement.

Remarquez que la taille de fichier est limitée par le nombre d'indirections.

Arborescence structurant les blocs physiques d'un fichiers



Structure interne d'un répertoire

Il existe généralement plusieurs types de fichiers dont :

1. les fichiers ordinaires : tableaux linéaires d'octets identifiés par un numéro d'inœud
2. les fichiers spéciaux associées aux périphériques ou aux processus
3. les répertoires : ces fichiers permettent de repérer un fichier ordinaire par un nom plutôt que par son inombre.

Les répertoires sont des fichiers comme les autres qui contiennent une table associant le nom des fichiers aux blocs physiques.

- MS-DOS : le répertoire contient les informations suivantes :

8	3 octets	1	10	2	2	2	4
nom	extension	attribut	réservé	heure	date	first bloc num	taille

  Ces informations ne sont pas dans la FAT qui structure les fichiers.

- UNIX : les informations comme le propriétaire, etc. étant dans l'inœud il suffit de stocker la correspondance fichiers-inombre :

4 octets	1 octets	14 octets
numéro de l'inœud	longueur du nom	nom du fichier

Les répertoires suffisent pour implanter l'arborescence des fichiers

Pour faire la correspondance entre chemins d'accès et bloc physique (par exemple pour le fichier /bar/foo), il faut :

1. Localiser la racine qui est toujours située au même endroit (nous verrons ce point en étudiant les volumes dans la seconde partie de ce cours).
2. Faire la correspondance entre le nom suivant et le numéro de l'inœud représentant ce fichier.
3. Demander à l'OS d'ouvrir ce fichier :
   - si c'est impossible retourner une erreur ;
   - si c'est un répertoire, recommencer l'étape 2 ;
   - sinon c'est fini.

Par commodités, un répertoire contient au moins 2 entrées :

le répertoire père et lui même.

Découpage d'un support physique : la notion de volume et son organisation

Un support physique est généralement structuré de la façon suivante :

<table>
<tr><td colspan="3">Master Boot Record</td></tr>
<tr><td rowspan="4">Volume 1</td><td colspan="2">Boot sector</td></tr>
<tr><td colspan="2">super bloc</td></tr>
<tr><td colspan="2">table inombres-numéro absolu de bloc</td></tr>
<tr><td colspan="2">FreeList ou FreeTable</td></tr>
<tr><td colspan="2">blocs de données</td></tr>
<tr><td rowspan="3">Volume 2</td><td colspan="2">Boot sector</td></tr>
<tr><td colspan="2">FAT</td></tr>
<tr><td colspan="2">blocs de données</td></tr>
</table>

Le MBR contient un petit programme qui permet de donner la main à des volumes différents contenant des systèmes d'exploitation différents.

Structuration des volumes : le super bloc

Le volume est l'unité de base du File System : il s'agit de la structure de données qui accueille la représentation des fichiers. Il existe systématiquement un *super-bloc* qui contient :

- le nom du volume et un numéro permettant de déterminer le FS
- le nombre de blocs organisés en volume
- le nombre d'inombres disponibles
- une structure donnant accès aux blocs libres
- une structure donnant accès aux inombres libres
- le bloc décrivant le répertoire racine

En mémoire, il existe des structures de données stockant l'ordonnancement des volumes.

Structuration des volumes : le super bloc

Les blocs constituant un fichier peuvent être structuré par :

- une table FAT
- un inœud

Quand le système manipule un fichier il doit donc disposer dans ces cas :

- de la clef de début du fichier dans la FAT
- de l'inombre qui est associé à un inœud

Il faut organiser au sein du volume les structures de données représentant les fichiers et les blocs libres.

Il existe généralement une table des inœuds qui associe un numéro à un bloc contenant l'inœud afin de le localiser.

Un des premiers inombres est réservé pour la racine.

Dans le superbloc, on trouve toujours l'inombre de la racine de l'arborescence.

À partir des entrées de la racine, on peut se déplacer dans cette arborescence.

Virtual File System : montage de FS différents

Afin de pouvoir monter plusieurs FS dans la même arborescence, l'OS interpose entre le FS qui gére effectivement le volume et les procédures qu'il met à disposition un système de fichiers virtuel. Ce n'est pas une abstraction supplémentaire mais un FS virtuel qui n'existe pas en mémoire persistante mais en mémoire vive. Ce VFS stocke ainsi l'ensemble des informations nécessaires pour la gestion des différents FS montés (superbloc, etc).

```
LABEL=/              /                ext3    defaults          1 1
none                 /dev/pts         devpts  gid=5,mode=620    0 0
none                 /dev/shm         tmpfs   defaults          0 0
LABEL=/home_local    /home_local      ext3    defaults          1 2
none                 /proc            proc    defaults          0 0
none                 /sys             sysfs   defaults          0 0
/dev/hda6            /windows_E       vfat    defaults          0 0
/dev/hda8            swap             swap    defaults          0 0
/dev/hdc             /media/cdrom     auto    pamconsole,
/dev/sda1            /media/usb-storage vfat  pamconsole, ...
```

Utilisation du FS pour les entrées-sorties

Il existe dans le répertoire `/dev/`, des fichiers associés à des périphériques physiques avec 2 méthodes d'accès possibles :

1. Utiliser un appel système — open, etc. — pour écrire dans le fichier. Mais dans ce cas, le processus utilisateur pourrait accaparer et donc bloquer le périphérique en cas de problème ;
2. Passer par un spool :
   - on construit un répertoire dédié avec des droits adéquates ;
   - un processus tourne en arrière fond — on parle de démon.

   Pour transmettre de l'information au service, on place un fichier dans le répertoire. Le démon inspecte régulièrement ce dernier et gère le fichier ainsi placé. Voici quelques exemples de spool

```
% ll /var/spool/
total 88
drwx------  3 daemon daemon 4096 Jan 10 09:56 at
drwx------  2 root   root   4096 Oct 15 16:20 cron
drwxr-xr-x  2 root   root   4096 Aug 12 19:02 lpd
drwxrwxr-x  2 root   mail   4096 Jan 24 11:31 mail
```

Les verrous d'accès à un fichier

L'OS maintient également une *table des verrous*.
Les verrous sont des mécanismes de contrôle d'accès aux fichiers.
Les verrous disposent de 2 caractéristiques :

- sa porté : les numéros logiques de début et de fin auxquels le verrou s'applique. On peut ainsi ne verrouiller qu'une partie de fichier ;
- son type : qui peut soit autoriser la cohabitation avec un autre verrou bloquant le même fichier, soit l'exclure.

Ainsi — en théorie — l'OS doit tester la présence d'un verrou dès qu'un processus demande un accès à un fichier et suivant l'état du verrou autoriser ou non la requête.

New Technologie File System (NTFS)

Il existe d'autre façons de construire un FS. Par exemple dans NTFS, tout est structuré sous forme de fichiers : ce qui remplace les inodes et la FAT est contenu dans un fichier (le MFT). Il existe plusieurs fichiers permettant de manipuler le volume :

- le fichier de boot est un fichier toujours au même endroit. Entre autres informations, il contient l'adresse physique du MFT ;
- le Master File Table qui est une gigantesque table d'*enregistrements* de fichiers.

Un enregistrement est une table dont les clefs sont des *attributs* et les entrées leurs *valeurs*. Un attribut peut être :

- le nom du fichier, les informations : propriétaire, la date de création, les droits du fichiers, etc. ;
- une structure arborescente faisant la correspondance entre adresses physiques et logiques (même principe que les inodes) ;
- les blocs de données d'un fichier.

La notion d'attribut provient du modèle transactionnel sur lequel se base ce FS (Ext3 implante aussi ce type de modèle tout en conservant les inœuds).
Contrairement aux exemples précédants, aucune limite n'est imposée par construction à la taille maximale d'un fichier.

Lorsque le fichier est suffisament petit, les blocs de données peuvent être contenus dans l'enregistrement et donc dans le MFT ! ! !

Voici quelques exemples d'entrées de cette table :

0. l'enregistrement du MFT ;
1. l'enregistrement de la copie du MFT ;
5. l'enregistrement du répertoire racine ;
6. l'enregistrement d'un fichier bitmap pour les blocs libres ;
8. l'enregistrement d'un fichier constitué des blocs défectueux ;

- les fichiers utilisateurs commencent à la clef 16.