

## Fichiers

Fabien Rico

Univ. Claude Bernard Lyon 1

séance 7

Jacques BONNEVILLE	jacques.bonneville@univ-lyon1.fr	TP
Adil KHALFA	adil.khalfa@cc.in2p3.fr	TD + TP
Léo LE TARO	leo.le-taro@inria.fr	TD + TP
Fabien RICO	fabien.rico@univ-lyon1.fr	CM+ TD + TP



## Introduction

- L'informatique est la science du traitement de l'information.
- $\Rightarrow$  entre les traitements il faut la stocker.
- Historiquement :
  - ▶ Fichiers : *A - Ensemble de fiches classées selon un ordre déterminé.*  
*B - Boîte, meuble et/ou local où l'on classe des fiches.*
  - ▶ Les fichiers mécanographiques : machines dédiées au classement et à la gestion des cartes perforées.
- Notion de base de l'informatique.
- Indépendante de la façon dont est réellement stockée l'information.
- Étendue à d'autres sources d'informations (sous unix tout est fichier).



## Propriétés

Outre le stockage de l'information le système doit fournir plusieurs propriétés aux fichiers :

- Indépendance vis à vis des médias de stockage.
- Gestion automatique de la taille.
- Vitesse d'accès.
- Protection des données contre l'accès concurrent.
- Notion d'ouverture et fermeture de fichiers pour gérer les ressources.
- Protection des données privées (droits).
- *[en plus]* Archivage, sauvegarde.



- 1 Les fichiers
  - Définitions
- 2 Structure
  - Structure du disque
  - Structure du fichier
- 3 Outils
  - Gestion des blocs
  - Le buffer cache
- 4 Ordonnancement des requêtes du disque
- 5 Évolution
- 6 Conclusion



## Types de fichiers - I

Nous avons vu que les fichiers peuvent être aussi bien des *tubes* que des répertoires contenant d'autres fichiers. On distingue alors 4 grands *types* de fichiers :

### Fichiers standards

qui contiennent des données utilisateurs.

### Répertoire (ou catalogues)

Fichiers contenant une liste d'autres fichiers et permettant d'organiser l'ensemble des fichiers du système sur un mode hiérarchique.



## Types de fichiers - II

### Fichiers spéciaux de type caractères

qui modélisent des périphériques d'entrée/sortie de type caractère, comme les terminaux (clavier et console), et les outils de communication sous forme de flux d'octets comme les pipes et les sockets.

### Fichiers spéciaux de type bloc

qui modélisent des périphériques d'entrée/sortie qui sont lisibles (inscriptibles) par blocs comme les disques.



## Types des fichiers standards

Les fichiers standards ont plusieurs utilisations (exécutable, fichier texte, images, ...), on parle aussi de *type*.

Le système a alors plusieurs choix :

- **Typage fort** : le fichier a un type défini par son nom (*extension*). C'est le cas par exemple sous Dos ou Windows.
  - ▶ Un fichier exécutable doit se terminer par `.exe`, `.com` ou `.bin`.
  - ▶ Le système reconnaît le logiciel à utiliser en fonction de l'extension (voir `HKEY_CLASSES_ROOT` sous Windows).
- **Typage déduit** : le système détermine la nature du fichier en fonction de leur contenu ou de ses propriétés.
  - ▶ un fichier est exécutable s'il a le droit d'exécution
  - ▶ On utilise un *nombre magique*, i.e., un code placé au début du fichier ou des directives placées au début des fichiers de texte (voir la commande `file` sous unix).



## Type MIME

Le type des fichiers est tellement important que dans le cas où l'extension disparaît, il faut un autre moyen de donner l'information.

- Par exemple sur internet le *type MIME* ou *Content-Type* qui est à la base de l'échange d'informations sur internet :
  - ▶ Tout entête de page web ou de fichier joint à un mail contient le type MIME.
  - ▶ Le navigateur ou le logiciel de lecture a un tableau d'association entre type et logiciel.
  - ▶ À la lecture du type, le navigateur choisit le logiciel à appeler
- Pour générer un « fichier image », il suffit d'envoyer le bon type MIME et le bon contenu (exemple de `php-gd`).
- On peut tricher, pour faire un « export excel », le site `Spiral` envoie le type MIME `application/vnd.ms-excel` suivi d'un tableau `html` puis il laisse le tableur faire le travail.



## Organisation d'un fichier

Les fichiers ont parfois besoin d'une organisation interne :

- Fichiers texte :
  - ▶ Ligne de caractères terminée par des caractères spéciaux CR (`\r`) LF (`\n`).
  - ▶ Encodage des caractères accentués (UTF8 ou iso8859-1). Voir la commande `iconv`.
- Fichier exécutable ELF (sous unix)
  - ▶ Un entête qui décrit la position des différentes parties du fichier, le sens du codage (little ou big Indian), l'architecture du processeur...
  - ▶ Des sections (donnée `.data`, données constantes `.rodata`, code `.text`, la table des symboles `.symtab`...)
- Fichiers de base de données.



## Répertoires

### Définition (Répertoire)

Un répertoire ou catalogue est un fichier spécial dont le rôle est d'organiser l'ensemble des fichiers.

- C'est au départ une liste de fichiers.
- On y accède par des appels système différents `opendir`, `readdir`, `closedir`...
- Un répertoire peut faire partie d'un autre ⇒ Structure arborescente.
- D'autres méthodes d'organisation ont été essayées.
- Le même fichier peut faire partie de plusieurs répertoires *liens* « matériel ».
- Un même répertoire ne peut faire partie que d'*un seul autre*. Pourquoi ?
- Possibilité de fichiers qui sont des pointeurs, les *liens symboliques*
- 2 répertoires spéciaux `.` et `..`



## Informations dans un répertoire

Chaque entrée de la liste contient :

- Le nom du fichier
- Le type de l'entrée (ordinaire, lien symbolique, pipe, socket, fichier caractère ou bloc, ...)
- Sa taille
- Ses droits d'accès
- Date d'accès, de modification...
- Les données permettant de le retrouver sur le disque (*l'inode*).



- 1 Les fichiers
  - Définitions

- 2 Structure
  - Structure du disque
  - Structure du fichier

- 3 Outils
  - Gestion des blocs
  - Le buffer cache

- 4 Ordonancement des requêtes du disque

- 5 Évolution

- 6 Conclusion



## Structure physique du disque - I

Les disques durs sont constitués d'un empilement de disques magnétiques rigides. La lecture se fait par un jeu de bras terminé par une tête de lecture. Cela a donné la terminologie suivante :

- **Piste** : une zone couverte par une tête de lecture lors d'un tour de disque si le bras ne bouge pas ;
- **Cylindre** : zone couverte par tous les bras, donc ensemble de toutes les pistes correspondantes sur tous les disques
- **secteur** : portion de disque représentant une fraction de la surface angulaire

Sur les anciens disques durs les données étaient accédées en fonction de leur position physique sur le disque Cylindre/Tête/Secteur.



## Structure physique du disque - II

### Exemple (Mon disque dur)

```
Disque /dev/sda : 320.1 Go, 320072933376 octets
255 têtes, 63 secteurs/piste, 38913 cylindres,
total 625142448 secteurs, Unités = secteurs de 1 * 512 = 512 octets
Taille de secteur (logique / physique) : 512 octets / 512 octets
taille d E/S (minimale / optimale) : 512 octets / 512 octets
```

Un secteur fait 512 octets il y a 255 pistes  $\times$  63 secteurs par piste et 38913 cylindres de 255 pistes soit  $255 * 63 * 38913 = 625137345 < 625142448$  secteurs. Cela ne tombe pas juste !

### Exemple (Une clef USB)

```
Disque /dev/sdb : 8016 Mo, 8016363520 octets
154 têtes, 11 secteurs/piste, 9242 cylindres,
total 15656960 secteurs, Unités = secteurs de 1 * 512 = 512 octets
Taille de secteur (logique / physique) : 512 octets / 512 octets
taille d E/S (minimale / optimale) : 512 octets / 512 octets
```

Une clef USB n'a pas réellement de disques, à quoi correspond un cylindre ?

## Les blocs

Les disques sont des périphériques de lecture par blocs. Ça signifie :

- Les données sont lues par ensemble de secteurs : le **bloc**.
- La lecture de toutes les données d'un bloc est donc rapide.
- Entre 2 blocs, le déplacement du bras de lecture et l'attente du passage du bon secteur est beaucoup plus lent.
- On ne peut pas lire ou écrire moins d'un bloc, c'est donc la taille minimale d'un fichier.
  - ▶ Fragmentation interne
  - ▶ Perte de place (différence entre quota et taille des fichiers)
  - ▶ stéganographie
- Ressemble à la notion de page mémoire.



## Pages mémoires et bloc

Les deux notions se ressemblent fortement :

- Importance de la taille :
  - ▶ Trop petit, ils imposent en grand nombre de blocs par fichier et donc beaucoup de déplacements du bras.
  - ▶ Trop gros, ils entraînent beaucoup de pertes de place.
- Problème d'allocation de bloc, gestion des blocs libres.
- Utilisation d'une table de blocs par fichier.

Il y a cependant des différences :

- Notion de blocs endommagés.
- Pas d'algorithme de « paging » (il n'y a rien après le disque).
- Les blocs ne sont pas tous accessibles en temps constant.
- Mémoire persistante (notion de cohérence après l'arrêt brutal).



## 1 Les fichiers

- Définitions

## 2 Structure

- Structure du disque
- Structure du fichier

## 3 Outils

- Gestion des blocs
- Le buffer cache

## 4 Ordonnancement des requêtes du disque

## 5 Évolution

## 6 Conclusion



## Allocation de blocs

- Objectifs :
  - ▶ allouer les blocs aux fichiers ;
  - ▶ pouvoir retrouver les blocs dans le bon ordre ;
  - ▶ la plupart des fichiers sont petits (qq blocs) ;
  - ▶ certains sont très gros.
- Allocation contiguë
  - ▶ Les fichiers sont stockés en un seul morceau.
  - ▶ Le répertoire ne contient que le numéro du premier bloc et la taille
  - ▶ Les accès sont très rapides
  - ▶ Cause une grosse fragmentation externe.
  - ▶ Problème pour augmenter un fichier.



## Organisation par listes chaînées

- Le répertoire ne contient que le premier bloc du fichier.
- Ce bloc renvoie au suivant . . .
- Le chaînage peut être séparé du fichier pour accélérer l'accès aux blocs lointains.
- C'est le système des *File Allocation Table = FAT*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			EOF	13	2	9	8		4	12	3		EOF	EOF	
Répertoire															
A	6	→ 8		→ 4		→ 2		→ EOF							
B	10	→ 3		→ 13		→ EOF									
C	5	→ 9		→ 12		→ EOF									

- Cela ressemble à la table des pages :
  - Il faut une case dans la table même si le bloc n'est pas alloué.



## Avantages des indirections

Discutez de l'efficacité de cette organisation pour retrouver les données :

- En considérant de gros fichiers.
- En considérant de très petits fichiers.

Que proposeriez-vous ?



## Organisation par *I-node*

Comme pour la table des pages, l'idée est d'associer à chaque fichier une table avec plusieurs niveaux d'indirections.

- Une table qui pointe sur une table . . . qui pointe sur un bloc.
- Comme la plupart des fichiers sont petits, on utilise un système de capacité variable.
  - Supposons des blocs d'1Ko (donc 256 entiers de 32 bits).
  - l'*inode* contient 14 adresses de blocs
  - Les 10 premiers blocs contiennent les données du fichiers (max 10Ko)
  - le bloc 11 contient un bloc d'indirections simple
    - 1 bloc qui contient 256 blocs de données
    - 256Ko max
  - Le bloc 12 contient des indirections doubles
    - 1 bloc qui contient 256 blocs qui contiennent chacun 256 blocs données
    - 64Mio max
  - le bloc 13 contient des indirections triples
    - 1 bloc qui contient 256 blocs qui contiennent chacun 256 blocs qui contiennent chacun 256 blocs de données.
    - 16Gio Max



## Contenu de l'*I-node*

Sous unix, outre les blocs eux-mêmes le i-node contient :

- Son propre numéro qui identifie le fichier dans le système de fichiers.
- La taille du fichier en octets.
- La taille du fichier en block.
- Identifiant du périphérique contenant le fichier.
- L'identifiant du propriétaire du fichier.
- L'identifiant du groupe auquel appartient le fichier.
- Les droits (lecture/écriture/exécution, pour 3 groupes de personnes plus setuid bits...)
- Le temps de
  - La dernière modification de l'inode (ctime).
  - La dernière modification du fichier (mtime).
  - Le dernier accès (atime).
- Un compteur indiquant le nombre de liens matériels sur cet inode.



**Il n'y a pas son nom**



## Contenu des répertoires

Pour retrouver le fichier, le répertoire contient :

- Pour les systèmes de blocs chaînés ou contigus :
  - L'adresse du premier bloc.
- Pour le système FAT :
  - Le numéro du premier bloc dans la table.
- Pour les systèmes ext 2/3 (avec inode) :
  - Le numéro de l'inode unique pour la partition.
  - Nécessite une table des inodes sur le disque.
  - Cette opération donne un nom au fichier.
  - Si le même inode est référencé plusieurs fois, ce fichier a plusieurs noms : *lien matériel*.



## Blocs libres

Le système doit maintenir la liste des blocs libres pour pouvoir les attribuer rapidement à un fichier.

- Utilisation d'une chaîne de blocs libres :
  - Chaque bloc contient l'adresse du bloc libre suivant.
  - Mais il contient aussi le plus grand nombre possible d'adresses de bloc.
  - Cela permet d'obtenir une liste de blocs libres en quelques lectures de disque.
- Utilisation d'une « carte » des blocs du disque :
  - Une suite de bits, chacun correspond à un bloc et est égale à 1 si le bloc est libre.
  - Vu la taille des blocs, cette liste est relativement petite. Par exemple pour un disque de 2Gio décomposé en blocs de 2Kio, il suffit de 64 blocs de 2Ko.



## Fragmentation/Défragmentation

Comme pour la mémoire, au bout d'une certaine utilisation, les blocs libres d'un disque se retrouvent répartis en petits groupes. Mais :

- Un fichier peut être coupé en petit morceaux (fragmenté).
- La fragmentation du fichier a un coût.

Lors de l'écriture d'un fichier, il y a plusieurs stratégies :

- Remplir les trous (génère de la fragmentation - FAT)
- Choisir les emplacements suffisamment grands (pas toujours possible si le disque est rempli - NTFS, ext)

Pourquoi y a-t'il plus d'utilitaires de défragmentation sous windows que sous unix ?



## Gestion des blocs endommagés

Certains blocs du disque peuvent devenir défectueux :

- Il faut une table des blocs endommagés pour éviter de les utiliser.
- Cela peut être fait directement par le matériel :
  - ▶ Table des blocs en interne.
  - ▶ Blocs de remplacement (blocs supplémentaires cachés au système).
- Ou par le système :
  - ▶ Table des blocs défectueux.
  - ▶ Ceux-ci ne peuvent plus aller dans la table des blocs libres.
  - ▶ Utiliser l'option `-c` de `mkfs`.



## Cache

- En théorie chaque écriture d'une donnée entraîne l'écriture du bloc.
- C'est peu efficace surtout dans un système multi-tâches avec un grand nombre de lectures/écritures.
- On peut utiliser un cache en mémoire vive.
  - ▶ Les blocs lus sont stockés en mémoire.
  - ▶ Pendant un certain temps, les écritures sont uniquement effectuées en mémoire.
  - ▶ De temps en temps le système synchronise les blocs.
  - ▶ Cela ne vous rappelle rien ?
- Lorsqu'il faut libérer des blocs dans le cache, on utilise souvent l'algorithme LRU. Pourquoi ?



## Avantages et désavantages

- + Le buffer cache fait interface entre le système et le disque. Cela permet un accès uniforme et une programmation modulaire.
- + Réduit beaucoup le trafic sur le disque (principe de localité)
- + Permet de régler les problèmes d'accès concurrent
  - Impose une écriture mémoire supplémentaire
  - Problème lors de crash du système

Le système de fichiers est persistant, il est nécessaire de savoir ce qu'il devient en cas d'arrêt brutal de la machine.



## Vérification de la cohérence - I

Suite à un arrêt brutal, le système de fichier doit être inspecté scandisk sous Windows ou `fsck` sous Unix.

- Pour chaque bloc on calcule 2 compteurs, le nombre de fois où il est référencé par un fichier et le nombre de fois où il est dans la liste des pages libres.
- On passe en revue les fichiers pour le premier compteur.
- On passe en revue la liste des blocs libres pour le 2<sup>e</sup> compteur.
- Si chaque bloc est marqué : soit une seule fois libre, soit une seule fois membre d'un fichier, le système est cohérent.



## Vérification de la cohérence - II

- Sinon :
  - ▶ Si les deux compteurs sont à 0 le bloc a été perdu et est réaffecté dans les blocs libres.
  - ▶ Si le compteur des fichiers est 0 et celui des libres  $> 1$ , on enlève le doublon de la liste des pages libres.
  - ▶ Si le compteur des fichiers est égale à 1 et l'autre  $> 1$ , le bloc est supprimé de la liste des blocs libres.
  - ▶ Si le compteur des fichiers est supérieur à 1, on copie le bloc pour l'affecter séparément aux fichiers (mais le système est forcément corrompu).



## Vérification de la cohérence - III

Il faut aussi vérifier la cohérence entre le nombre de références stockées dans chaque inode et l'ensemble des répertoires.

- On parcourt l'ensemble de l'arborescence des répertoires en comptant les références.
- On compare avec le nombre stocké dans chaque inode.
- Si les 2 ne sont pas identiques :
  - ▶ Si le compteur dans l'inode est inférieur au nombre réel de références, le fichier risque de ne jamais être libéré.
  - ▶ Si le compteur est inférieur, il risque d'être libéré alors qu'il est encore utilisé.
- On corrige sans réellement libérer de fichiers : ceux dont le nombre réel de références est nul sont placés dans un répertoire spécial (lost+found).



## Performance

Pour qu'un bloc soit lu il faut :

- Que le bras se positionne sur le bon cylindre.
- Que le secteur concerné passe sous la tête de lecture.
- Que les données passent de la tête de lecture à la mémoire centrale.

Le temps de ces actions est décroissant. La performance de la lecture dépend beaucoup de la position de ce qu'on lit.



## Ordonnement FIFO

Les lectures sont effectuées en fonction de leur ordre d'arrivée.

- Équitable
- Simple
- Peu efficace
  - ▶ Par exemple 2 lectures séquentielles de 2 fichiers.
  - ▶ Chaque requête impose un changement de position du bras.



## Ordonnement SSTF

*Shortest Seek Time First* : on choisit la requête du bloc le plus proche de la position actuelle de lecture.

- Réduit la distance de parcourt des bras de lecture.
- Problème de famine :
  - ▶ un gros fichier sur des blocs contigus est lu,
  - ▶ une autre requête arrive,
  - ▶ le bloc demandé est forcément plus éloigné.



## Ordonnement SCAN

On utilise le principe précédant, mais on impose aux bras de lecture de balayer continuellement le disque d'un bord à l'autre puis dans l'autre sens. À chaque fois on s'arrête pour servir les requêtes proches.

- Cela rend l'algorithme équitable.
- Temps d'accès très différent :
  - ▶ Une lecture est demandée au bord du disque.
  - ▶ Si elle a de la chance, le bras arrive et elle est servie immédiatement.
  - ▶ Sinon, le bras repart, elle doit attendre un aller-retour et n'est servie qu'après toutes les autres requêtes

Les données sont plus rapidement atteintes si elle sont au milieu du disque.



## Ordonnement C-Scan

Modification de l'algorithme précédant pour éviter le problème. Les requêtes ne sont servies que sur l'aller d'un bord à l'autre. Pendant son retour, les requêtes ne sont pas servies.

- Le temps d'accès est plus uniforme.
- C'est en moyenne le même pour toutes les données.

On peut améliorer cet ordonnancement, en limitant le parcourt du bras à l'intervalle contenant les requêtes.



## Résultats

- Dépend beaucoup des requête :
  - ▶ Fichiers séquentiels (film)
  - ▶ Fichiers indexés (base)
- Position des fichiers d'inode et de leurs tables d'indirection.



## Évolution

- Évolution du matériel
  - ▶ Plusieurs formes de « disque ».
    - ★ Cartes mémoires (répartition des écritures, besoin de confidentialité, ...)
    - ★ Disque de plus en plus efficaces (blocs de secours, ré-ordonnement des tâches, ...)
    - ★ Nouvelles fonctionnalités (branchement à chaud, RAID,...)
  - ▶ Disques de plus en plus gros (>To) :
    - ★ temps nécessaires aux outils;
    - ★ le taux d'erreurs reste important.
- Évolution des besoins
  - ▶ Sécurité des données (redondance, vérification, réparation)
  - ▶ Confidentialité (chiffrement, droit).
  - ▶ Sauvegarde (incrémentale, clichés).
  - ▶ Disque réseau.



## Amélioration

- Journalisation :
  - ▶ chaque modification est d'abord écrite dans un journal;
  - ▶ le journal est effacé lorsque la modification est terminée;
  - ▶ le système garde sa cohérence;
  - ▶ exemple ext3, ntfs, HFS+...
- Pré-allocation de zone continue :
  - ▶ lorsqu'une écriture est demandée, une zone plus grande est allouée (extend);
  - ▶ lorsque le fichier est agrandi, il est possible d'utiliser la zone;
  - ▶ cela évite la fragmentation;
  - ▶ exemple ext4, ntfs, HFS+, Btrfs, ...
- Vérification/Défragmentation en ligne
  - ▶ ces opérations sont faites durant l'utilisation normale;
  - ▶ permet la remise en route rapide.



## Volumes Logiques

- Couche de séparation entre le système et les volumes physiques.
- Agrège les volumes physiques (de différente forme) en groupes.
- Crée des volumes logiques sur les groupes.
- Permet :
  - ▶ RAID logiciel;
  - ▶ modification des disques à chaud (taille);
  - ▶ cliché (snapshots);
  - ▶ chiffrement!



## Fichiers en réseaux

### Partage des fichiers sur un réseau

- Un (ou des) serveurs partagent leurs fichiers.
- Le système client présente les fichiers comme des fichiers locaux
  - ▶ les applications ne font pas la différence
  - ▶ permet la centralisation des utilisateurs
  - ▶ permet le partage des machines
- Systèmes « historiques » :
  - ▶ NFSv3 (unix)
  - ▶ SMB (windows)
  - ▶ réseau local
- Autre systèmes :
  - ▶ Lustre, GFS (RedHat), GoogleFS, OCFS (Oracle)...
  - ▶ réseaux grande distance;
  - ▶ cluster de serveurs.



## Conclusion

- Toutes les données sont stockées dans des fichiers
  - ▶ Type de fichiers.
  - ▶ Fichiers spéciaux.
  - ▶ Répertoires.
- Notion de block.
- Notion d'inode.
- Buffer cache.

