Architecture et Systèmes

Stefan Schwoon

Cours L3, 2025/2026, ENS Paris-Saclay

Système de fichiers

Le terme système de fichiers peut désigner :

l'organisation logique des fichiers utilisés imposé par le système (p.ex. `/etc`, `/usr`, `/home`) ;

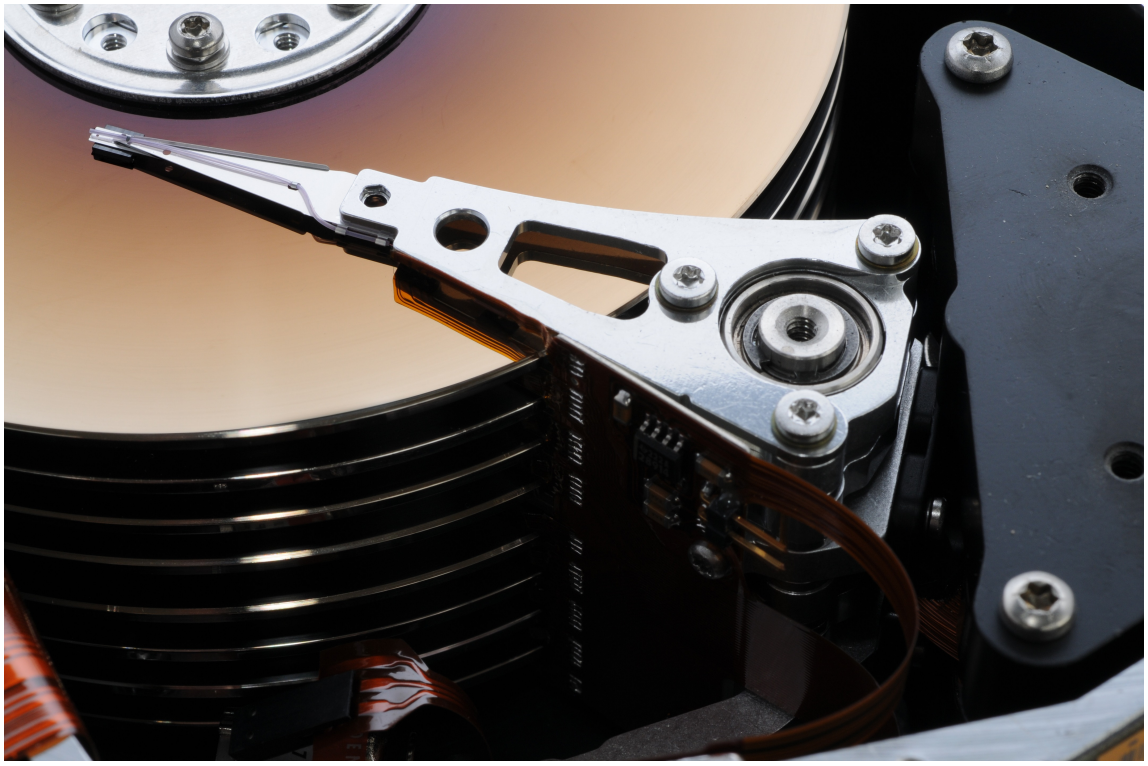
les structures de données qui permettent l'organisation physique des dossiers et fichiers ainsi que les opérations sur ces structures.

Exemples pour ce dernier cas : ext[234] sur Linux, FAT (diverses versions) sur Windows

Rappel : Sous Unix/Linux, un système de fichiers dans le sens logique peut comporter plusieurs systèmes physiques organisés dans une même arborescence.

Disque dur

Image d’un disque dur :



Source: Wikimedia Commons, Utilisateur HubiB

Organisation d'un disque dur

plusieurs plateaux qui tournent, chacun avec une tête de lecture

cylindre : zone équidistante de l'axe

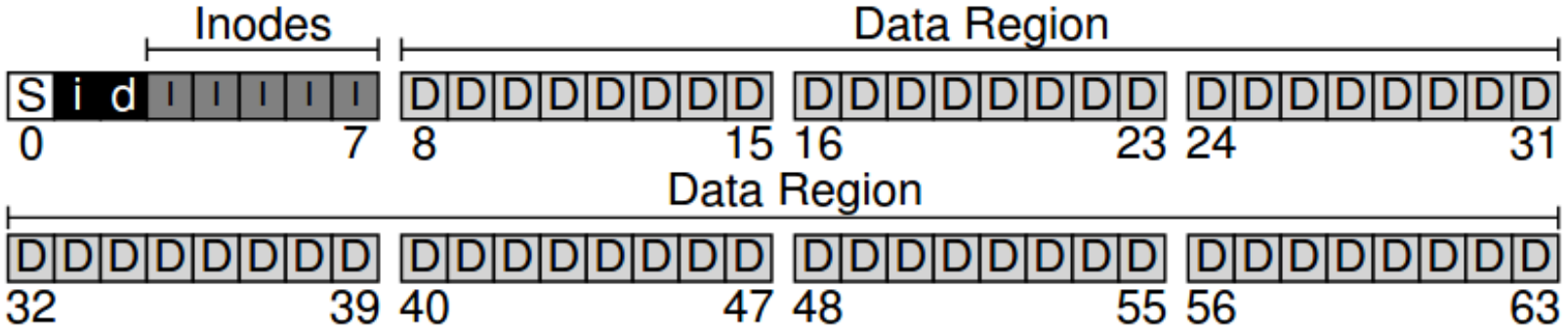
secteur : partie d'un cylindre qui contient un bloc de données
taille typique : entre 512 octets et 4 Ko

Les accès sont par secteur.

Lire ou écrire un secteur consiste à déplacer la tête d'un plateau vers le cylindre, puis attendre le passage du secteur.

Structures de base

La structure ci-dessous est à peu près suivi par l'implémentation originale dans Unix (et par Minix) :



Source: Arpaci-Dusseau, Operating Systems: Three Easy Pieces

Les secteurs de divisent en cinq types :
superbloc, i-tableau, d-tableau, inœuds, blocs de données

Superbloc et inœuds

Un inœud contient les métadonnées associées avec un fichier:

> proriétaire, droits d'accès, type, temps de dernière modif, taille en octets, lieu de stocage, etc
>
> voir `stat(1)`, `stat(2)`

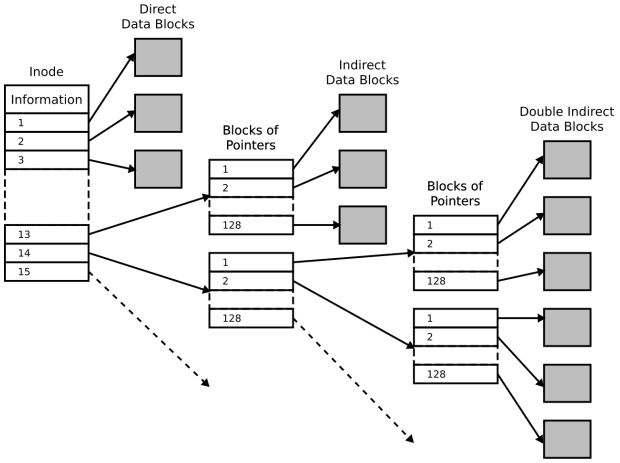
Un seul secteur stocke plusieurs inœuds ; l'ensemble de secteurs reservées aux inœuds en réalise un tableau.

Le secteur appelés i-tableau (resp. d-tableau) contient un bitmap indiquant quels inœuds (resp. blocs de données) sont occupées ou libres.

Superbloc : dans un secteur préalablement connu, contient le nombre d'inœuds et blocs de données, leurs endroits, inœud du dossier racine, ...

Pointeur direct ou indirect

Les inœuds soutiennent des fichiers de tailles diverses :



Wikimedia Commons, Utilisateur timtjtim

Pour des fichiers petits, 12 pointeurs directs indiquent les secteurs qui contiennent le contenu du fichier.

Pour des fichiers de taille moyenne, on réserve un secteur de données qu'on remplit avec les secteurs de données.

Les pointeurs doublement/triplement indirects permettent des fichiers très larges ou géants.

Alternatives

File Allocation Table (FAT): liste liée qui indique, pour tout secteur de données, le secteur suivant du même fichier

approche suivi par certains versions de MS-DOS/Windows

copie gardée en mémoire pour meilleure efficacité

Avantage: structure simple, fichiers de longueur illimitée

Problème : accès aléatoire dans un fichier ?

Extent : indique une zone de secteurs consécutifs (p.ex., dans ext4)

Dossiers

Un dossier est un fichier spécial qui contient une liste d'inœuds et les noms associées.

Note: Au lieu d'une simple liste, on peut utiliser une structure plus sophistiquée.

Exemple : Accès en lecture d'un fichier `/A/B`

1. Consulter l'inœud de la racine (/) pour savoir où est stocké sa liste de fichiers.
2. Consulter cette liste pour y trouver l'entrée `A` et son inœud.
3. Consulter l'inœud de `A` pour trouver sa liste de fichiers.
4. Consulter cette liste pour y trouver l'entrée `B` et son inœud.
5. Une fois l'inœud de `B` est connu, tout accès en lecture utilise un pointeur direct ou indirect pour en trouver le bon secteur.

Exemple : Création d'un nouveau fichier (court) `/A/C`

1. et 2. voir ci-dessus
3. Consulter l'inœud de `a` pour trouver sa liste de fichiers.
4. Vérifier dans cette liste que `C` n'y existe pas encore.
5. Trouver un inœud et un bloc de données disponibles pour stocker les métadonnées et le contenu de `C`.
6. Écrire ces secteurs sur disques et mettre à jour les tableaux.
7. Ajouter `C` à la liste de `A` et la réécrire sur disque.

Note: fsync(2)

Optimisations

En pratique, certaines optimisations s'imposent :

Cache : garder les secteurs les plus utilisés en mémoire

Tamponnage : retarder/grouper les écritures

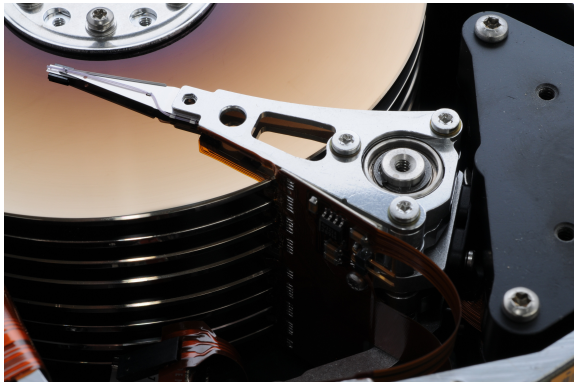
Néanmoins, un implémentation directe s'avère inefficace.

Pire, la performance dégrade avec le temps. Raisons :

manque de localité, fragmentation

Prise en compte de localité

L'efficacité des accès disque dur dépendent de l'ordre des secteurs.



Or, le système proposé précédemment (et implémenté à l'origine dans Unix) traite tous les secteurs comme si leur temps d'accès était toujours identique.

Idée : grouper des données reliés dans des secteurs proches.

Fast File System (FFS)

Solution proposé à Berkeley 1984, idées reprises par ext2 (Linux)

Grouper le disque en plusieurs zones physiques, p.ex. trois cylindres consécutifs.
Note : les disques plus récents proposent de telles zones.

Chaque groupe a sa propre décomposition en superbloc, tableaux, inœds, données.

Superbloc : copie rédondant du master

FFS: Allocation de groupes

Observation : Les accès consécutifs sont souvent dans un même dossier (environ 40% selon des données statistiques, 25% supplémentaires dans des dossiers qui partagent le même parent).

Lorsqu'un dossier est crée, on l'affecte à un groupe (p.ex. un groupe).
Ainsi, son inœud et contenu est stocké dans ce groupe, ainsi que les inœuds et données de ses fichiers.

Exception : pour des fichiers larges, on cherche un zone de secteurs *consécutifs* dans des groupes séparés.

- $\rightarrow$ éviter qu'un seul fichier prend toutes les ressources dans un groupe
- $\rightarrow$ faciliter la lecture en masse

Attention : En fonction du fonctionnement du disque, la lecture de deux secteurs physiquement consécutifs n'est pas toujours la solution la plus efficace (reglé par auto-paramétrage de FFS).

Cohérence/intégrité

Un problème intervient si le système s'arrête au milieu d'une opération d'écriture (perte d'alimentation électrique, bougues, . . .).

Exemple: étendre un fichier par un secteur supplémentaire

Trois écritures : d-tableau, inœud, données

$\rightarrow$ Qu'est-ce qui arrive si le système s'arrête après une seule écriture ou deux ?

Solution : Lorsque le système n'a pas terminé correctement, on vérifie l'état du système de fichiers au redémarrage (opération lourde; voir chkdsk, fsck).

vérifier cohérence entre d-tableau et blocs mentionnés dans inœuds

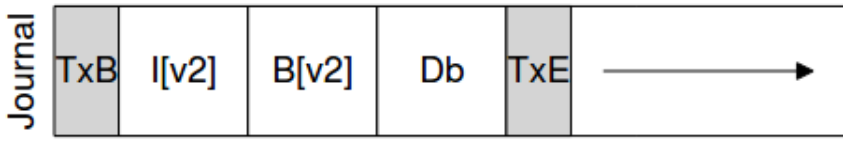
vérifier cohérence entre i-tableau et contenu de dossiers

etc.

Alternative : Journal de transactions

Une transaction est une suite d'écritures qui, ensemble, laissent le système dans un état cohérent.

On réserve une zone du disque pour un journal de transactions.
Exemple de transaction:



Source: Arpaci-Dusseau, Operating Systems: Three Easy Pieces

On écrit d'abord les données de la transaction dans le journal.

Puis, à un moment donnée, le système de fichiers est mis à jour et la transaction marqué comme terminée.

Transactions : traitement d'erreurs

Lors d'un redémarrage du système, toute transaction non-terminée est répétée.

Toute transaction incomplète (sans TxE) dans le journal est écartée.

Attention : le système doit assurer que TxE n'est jamais écrit avant le reste de la transaction.

Exemple d'implémentation: ext3 (Linux)

Points divers:

- tamponnage des mises à jour
- variante : omission des données de fichier dans le journal
- alternative: copy-on-write (toute écriture se fait dans un secteur non utilisé)
- autre alternative : LFS (système construit par une séquence de modifications)

RAID

RAID = redundant array of inexpensive/independent disks

Idée : utiliser plusieurs disques en parallèle afin de gérer les secteurs endommagés. Quelques exemples :

RAID niveau 0 : (rien de spécial, tout secteur est stocké dans un seul disque, comme d'habitude)

RAID niveau 1 : tout secteur est sauvegardé en parallèle sur tous les disques

RAID niveau 5 : l'un des disques stocke la parité d'un secteur

RAID niveau 2 : la même idée avec les codes de Hamming