Gestion des périphériques

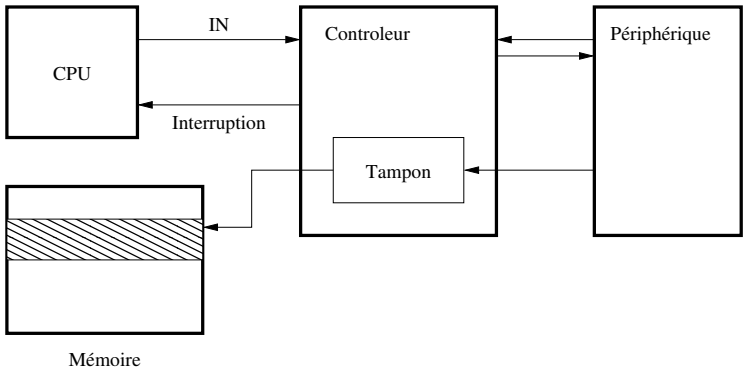
▶▶ Organisation des périphériques ◀◀

Il existe deux catégories de périphérique :

- les périphériques en **mode caractère** (carte réseau, imprimante, terminaux, bande, etc.).
  - ▷ l'unité élementaire d'échange est **l'octet** (un caractère)
  - ▷ les E/S sont souvent **séquentielles** :
    - lecture un octet depuis le périphérique
    - écriture un octet sur le périphérique
- les périphériques en **mode bloc** (disque, disquette, CD-ROM, bande),
  - ▷ l'unité élementaire d'échange est **un ensemble d'octets de taille fixe** (un bloc)
  - ▷ les E/S sont souvent **en accès direct** :
    - lire une bloc à l'adresse a sur le périphérique p
    - écrire une bloc à l'adresse a sur le périphérique p

▶▶ Entrée/Sortie par interruption (ADM) ◀◀

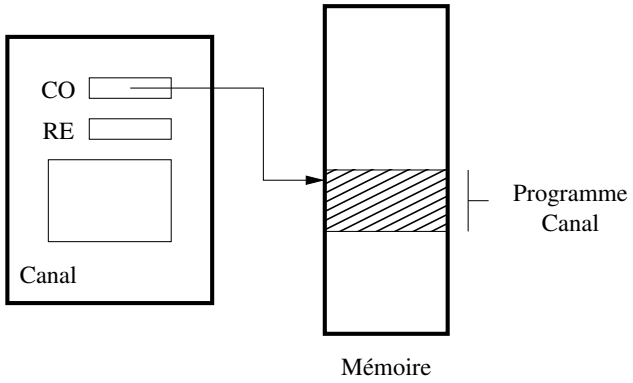
Le contrôleur réalise les écritures et les lectures **directement** en mémoire (**tampon** ou **buffer**). La CPU est donc **libérer** du travail de gestion.



Si le contrôleur ne dispose pas d'un tampon, on parle d'entrées/sorties par **vol de cycles de mémoire**.

▶▶ Entrée/Sortie gérée par un canal ◀◀

Un **canal d'E/S** (ou **circuit d'E/S**) déroule les opérations d'E/S en exécutant un **programme canal** :



Préparation d'une E/S et **lancement** d'un programme canal par le système d'exploitation :

```
sortir(a, b : données, p : périphérique) =
   | ⟨préparer le programme canal à l'adresse α⟩
   |     | mem[α + 0] = ⟨écrire a sur p⟩
   |     | mem[α + 1] = ⟨émettre une interruption fin E/S⟩
   |     | mem[α + 2] = ⟨écrire b sur p⟩
   |     | mem[α + 3] = ⟨émettre une interruption fin E/S⟩
   |     | mem[α + 4] = ⟨stopper le programme⟩
   | ⟨fin de préparation⟩
   | ⟨choisir un canal c libre⟩
   | canal_execute c, α
```

Les entrées/sorties se déroulent de manière **asynchrone**.

A tout moment le système est capable **d'interroger le canal** (instruction **canal_test**) ou **d'interrompre** l'exécution (**canal_stop**).

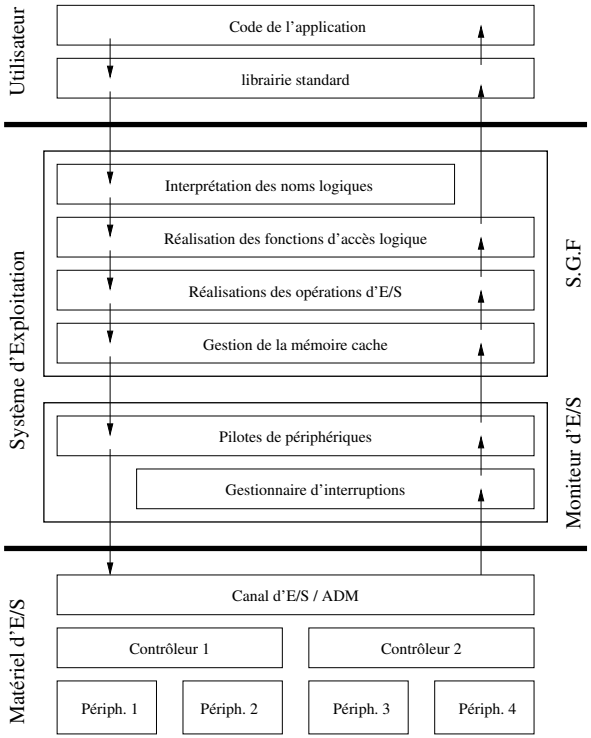
Le système de gestion de fichiers

Définition : Un **fichier** est un ensemble d'informations regroupées en vue de leur utilisation et de leur conservation.

Définition : **L'organisation logique** d'un fichier décrit son contenu vu par les processus utilisateur.

Définition : **L'organisation physique** d'un fichier décrit son implantation sur le support physique.

- Vu des processus, les informations sont repérées par des **adresses logiques**.
- Vu du système d'exploitation, ces informations ont une **adresse physique** sur le support.

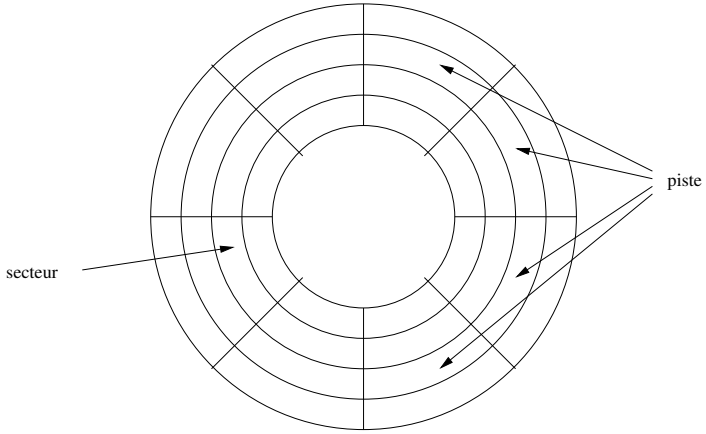


Organisation des disques

- Faces, pistes, secteurs, et cylindres
- Lecture / écriture d'un bloc
- Optimisation des requêtes disque
- Gestion du support
- Gestion et implantation du cache

▸▸ Structure physique des disques ◂◂

Un disque est composé de **deux faces**. Les faces sont découpées en **pistes** (de 20 à 1500) elles-mêmes divisées en **secteurs** (de 8 à 32). La taille des secteurs varie de 512 octets à 4 kilos-octet.



L'adresse d'un secteur est un triplet : ⟨**face** f, **piste** p, **secteur** s⟩.

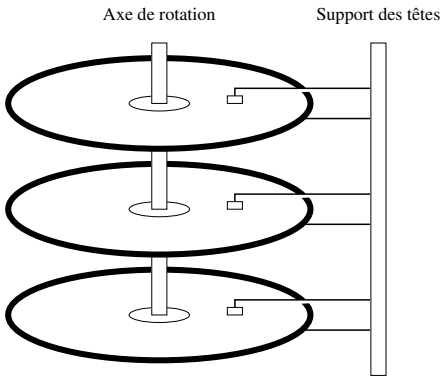
▸ Cylindres et autres... ◂

Un **support** est composé de plusieurs disques.

Un **cylindre** regroupe les pistes de **même numéro** de toutes les faces.

Un **bloc** est formé des secteurs de **même numéro** dans un cylindre.

Le bloc est **l'unité élémentaire** d'entrée/sortie.



Chaque bloc i a une **adresse physique** composée

$$\langle \underbrace{i \textbf{ div } N}_{\text{cylindre}}, \underbrace{i \textbf{ mod } N}_{\text{secteur}} \rangle \text{ avec } N = \text{nombre de secteurs par face}$$

Les blocs B_i et B_j **sont proches** ssi les numéros i et j **sont proches**.

▸ Lecture / écriture d'un bloc ◂

Pour une opération d'E/S il faut :

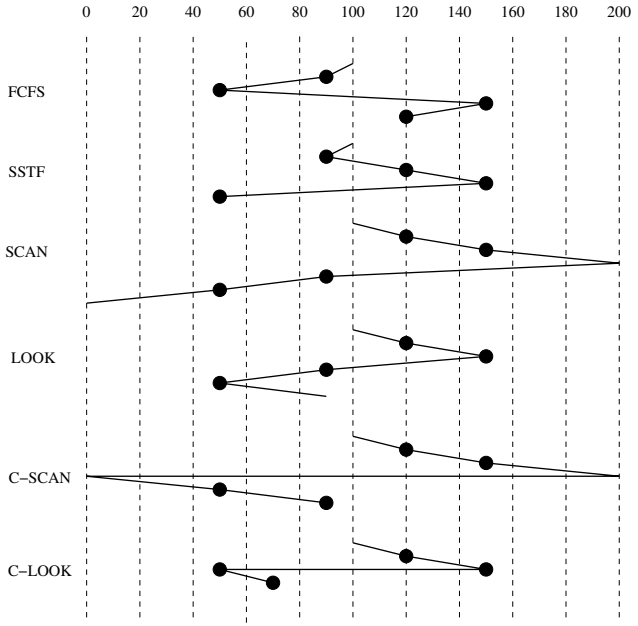
- **positionner** les têtes sur le bon cylindre
- **attendre** que le secteur soit sous la tête
- **lire ou écrire** le bloc (les secteurs de chaque face)

⟨temps d'E/S⟩ = ⟨*temps de latence*⟩ + ⟨*temps de lecture d'un secteur*⟩

Pour diminuer le **temps de latence** on applique un **ordonnancement des requêtes disques**.

- **FCFS** (*First Come First Served*) on respecte l'ordre d'arrivée,
- **SSTF** (*Shortest Seek Time First*) le plus proche en premier,
- **SCAN** (*balayage* ou *Algorithme de l'ascenseur*) parcours entier du disque dans les deux sens (variante LOOK),
- **C-SCAN** (*Circular SCAN*) balayage dans un seul sens (variante C-LOOK).

▶ Optimisation des requêtes disque ◀



▶ Gestion du support ◀

La **gestion du support** c'est l'allocation et la libération de blocs ou de **zones** (ensemble de blocs contiguës).

Un support est caractérisé par

- l'ensemble des blocs **libres**
- l'ensemble des blocs **occupés**
- l'ensemble des blocs **défectueux**

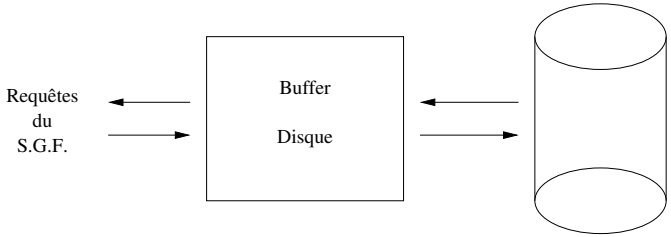
Ces ensembles sont représentés par

- un chaînage des blocs (**couteux en E/S**)
- un chaînage de blocs d'index (**l'allocation de zones est difficile**)
- une table de bits B telle que $B_k = 1$ ssi le bloc k fait partie de l'ensemble.

Exemple : Pour un disque de 512 Go et un bloc de 4 Ko, la table mesure

$$(\frac{2^9 \times 2^{10} \times 2^{10}}{2^2}) \times \frac{1}{2^{10} \times 2^3} = \frac{2^{29}}{2^{15}} = 2^{14} \text{ ko} = 2^4 \text{ mo} = 16 \text{ mo}$$

▶ Gestion du cache ◀



En stockant les derniers blocs utilisés, le **cache disque** permet

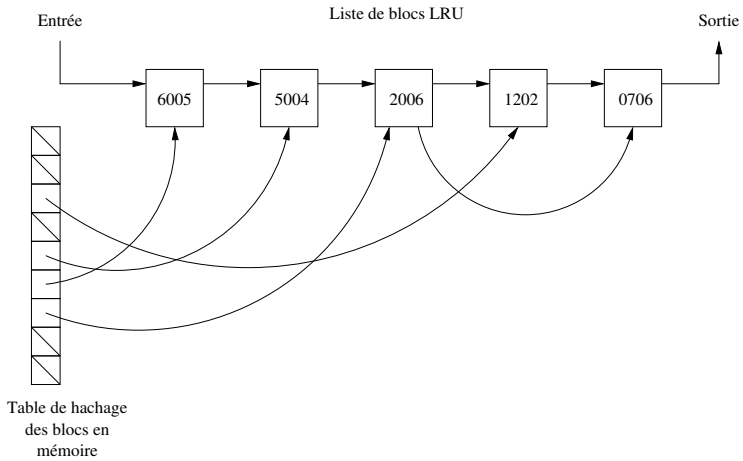
- de **diminuer** le nombre d'entrée/sortie,
- de réaliser des **écritures asynchrones**.

La présence d'un cache pose le problème de la **cohérence** des informations.

Ce problème est réglé par la **mise à jour périodique** du disque.

▶ Implantation du cache ◀

Pour gérer ce cache disque le système utilise une version adaptée de **l'algorithme L.R.U.**



Une **priorité** associée à chaque bloc permet d'affiner l'algorithme L.R.U.

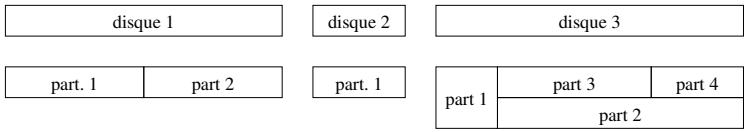
▶▶ Structure logique des disques ◀◀

- Partitionnement d'un disque,
  - ▷ Partitions physiques,
  - ▷ Partitions logiques,
- Répertoires simples,
- Répertoires arborescents,

▶ Partitionnement d'un disque ◀

Un disque est divisé en **partitions** (ou **partitions physiques**).

Chaque partition contient un **système de fichiers** autonome.



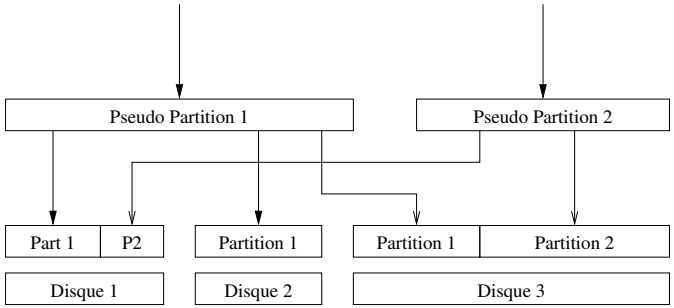
Dans chaque système de fichiers, un **répertoire** (ou **catalogue**) contient la liste des fichiers se trouvant dans la partition.

- **Sécurité** accrue
- Diminution de la **fragmentation interne**
- Possibilité de placer **plusieurs systèmes**

▶ Pseudo Partition ◀

Un **pseudo partition** (ou **partition logique**) regroupe plusieurs partitions physiques.

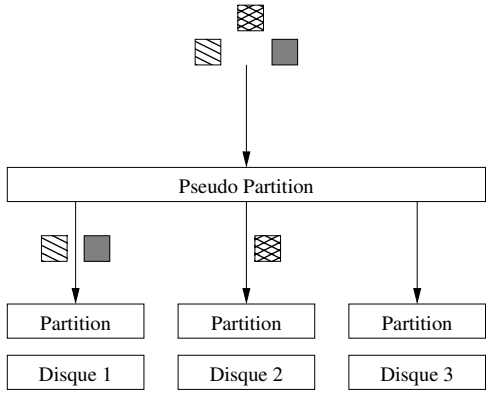
Un système de fichiers peut être **implanté** sur une pseudo partition.



La pseudo partition est une **ressource virtuelle**.

▶ Pseudo Partition (augmentation) RAID linear ◀

La **pseudo partition** est un assemblage de **partitions logiques**.

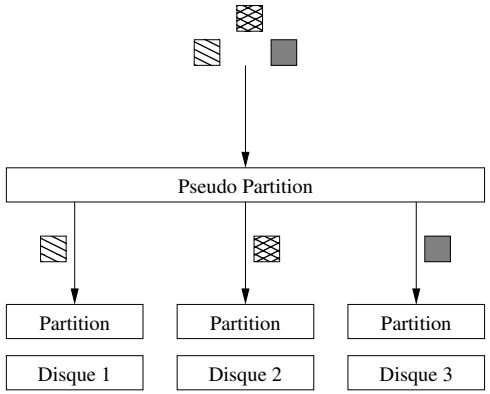


La **partition virtuelle** est égale à la somme des partitions physiques.

Cette architecture est aussi appelée **RAID linear** (**Redundant Array of Independent Disks**).

▶ Pseudo Partition (répartition) ou RAID 0 ◀

Les blocs sont **répartis** sur tous les disques dans un **souci d'efficacité**.

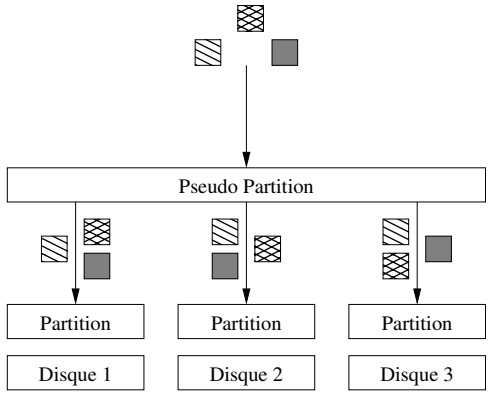


Contrainte : toutes les partitions physiques **ont la même taille** (il est souhaitable que les disques soient de **même qualité**).

On parle aussi de **RAID 0** ou **stripping**.

▶ Pseudo Partition (mirroir) RAID 1 ◀

Dans un souçi de **sécurité**, les blocs sont **dupliqués** sur plusieurs disques.

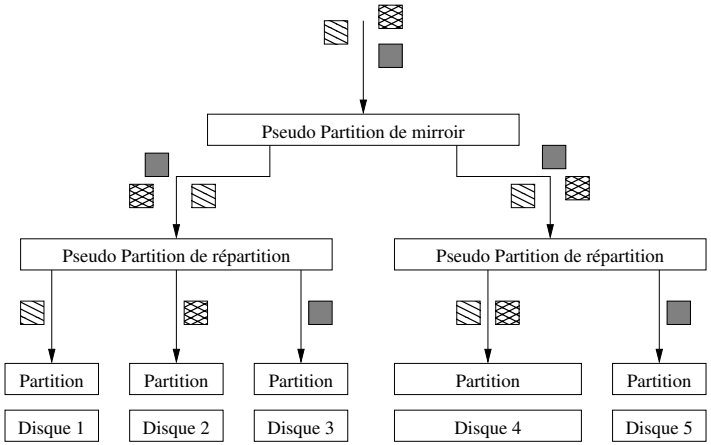


Contrainte : toutes les partitions physiques **ont la même taille** (il est souhaitable que les disques soient de **même qualité**).

On parle aussi de **RAID 1** ou **mirroring**.

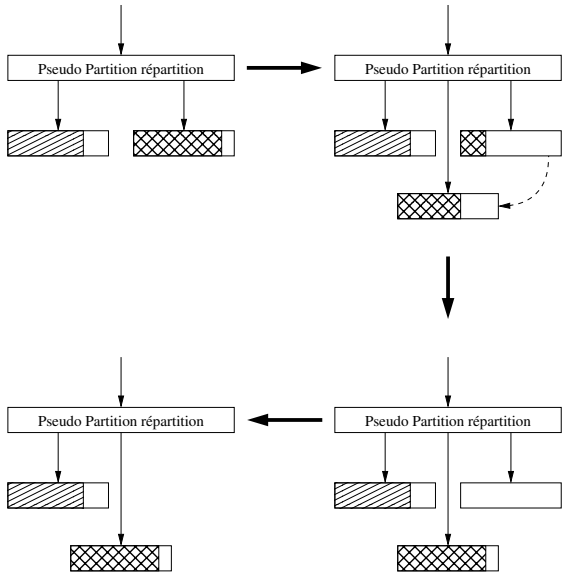
▶ Pseudo Partition (combinaison) ◀

Une **pseudo partition** peut aussi être un assemblage de pseudo partitions.

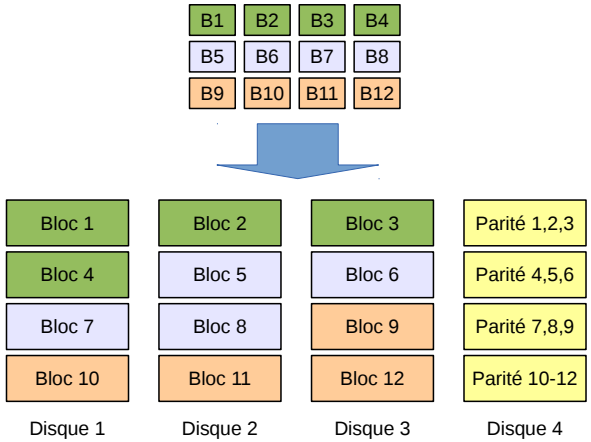


▶ Pseudo Partition (aspects dynamiques) ◀

La **configuration** peut peut être modifiée dynamiquement :



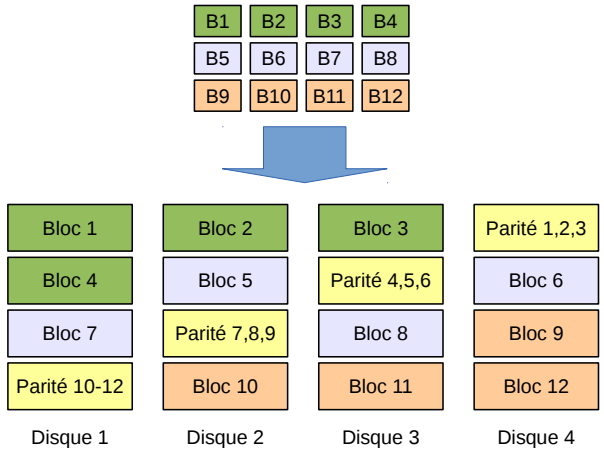
▶ Organisation en RAID 4 ◀



Caractéristiques :

- **Taille du RAID** $= (n-1) \times$ taille d'un disque
- Toutes les partitions physiques ont la **même taille**
- Au moins **trois** disques

▶ Organisation en RAID 5 ◀

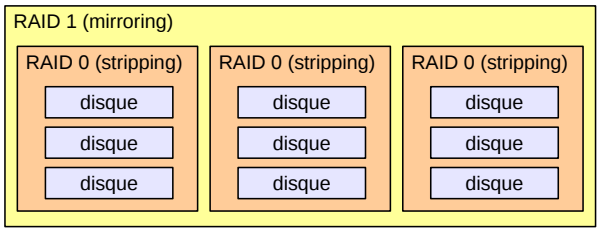


Mêmes caractéristiques que le RAID 4. **RAID 6** :

- n disques de parité (souvent deux)
- Accepte la défaillance de plusieurs disques (en fonction de n)

▶ Organisation en RAID 10 ou 1+0 ◀

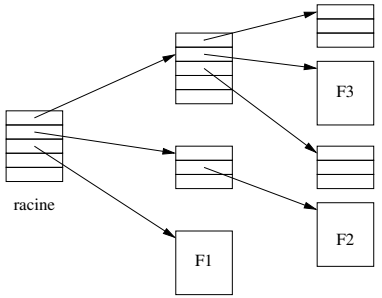
Combinaison d'un RAID 1 sur plusieurs RAID 0 :

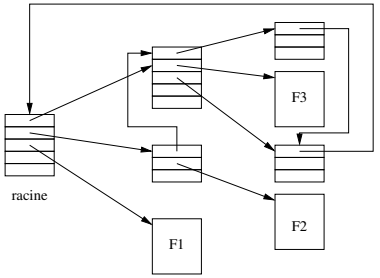


Autres combinaisons :

- **RAID 01 ou 0+1**
- **RAID 50 ou 5+0** : RAID 5 sur de la répartition : **efficacité**
- **RAID 51 ou 5+1** : RAID 5 sur du mirroir : **sécurité**

▶ Répertoires arborescents ◀





Les fichiers ont un **chemin d'accès** et le système d'exploitation définit un **répertoire par défaut**.

codage des répertoires :

- liste linéaire des couples (nom, référence),
- représentation par *hash-coding*,
- utilisation d'une mémoire cache.

Organisation physique des fichiers

- Descripteur de fichier
- Implantation physique d'un fichier
- Les fichiers séquentiels indexés
- Protection des fichiers
- Types de système de fichiers

▶▶ Descripteur de fichier ◀◀

Un **descripteur de fichier** est une **structure de donnée** du système d'exploitation stockée sur **disque** (dans la **table des descripteurs**).

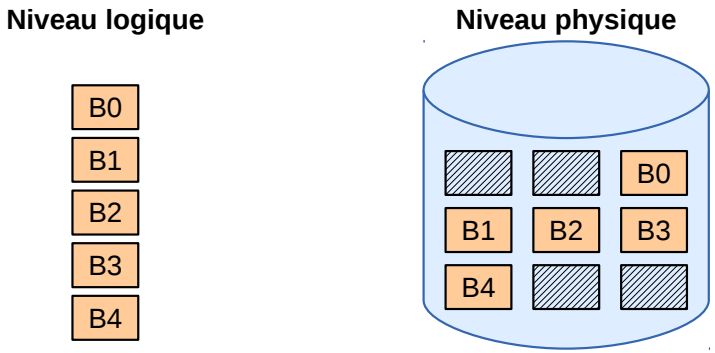
un fichier = ⟨ numéro-de-périphérique , numéro-de-descripteur ⟩

Le descripteur contient des informations sur :

- le **type** du fichier (fichier de données, exécutable, répertoire, ...),
- **l'organisation logique** (taille et structure des enregistrements),
- les **statistiques d'utilisation** (date de dernière modification, nombre d'utilisations, intervalle de temps entre deux utilisations),
- les **permissions** d'utilisation (**ACL** pour **Access Control List**).
- **l'implantation physique** du fichier.

Le **nom d'un fichier** n'est généralement pas stocké dans le descripteur mais dans les répertoires.

▶ Implantation contiguë ◀



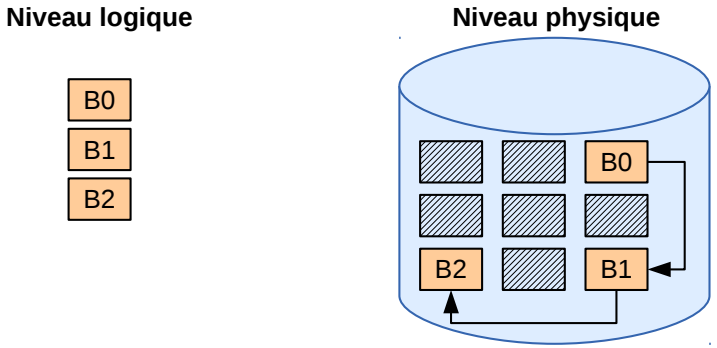
Avantages :

- Le descripteur contient l'adresse du premier bloc,
- le passage logique/physique est **simple**,
- l'accès séquentiel ou direct est **rapide**.

Inconvénients :

- **perte de place** par fragmentation externe,
- obligation de **connaître la taille du fichier**.

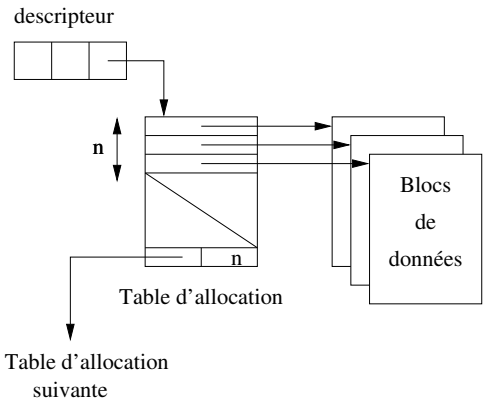
▶ Implantation par blocs chaînés ◀



Caractéristiques :

- le descripteur contient **l'adresse physique** du premier et du dernier bloc logique,
- les accès séquentiels sont **efficaces** (au déplacement de la tête de lecture près),
- les accès directs sont (presque) **impossibles**.

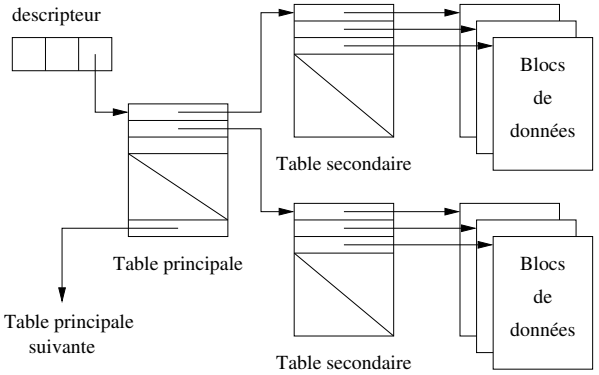
▶ Tables d'allocation ◀



Caractéristiques :

- Chaque table est un bloc
- Accès séquentiels **efficaces**
- Accès directs **améliorés**
- La taille des fichiers est **limitée**

▶ Tables à plusieurs niveaux ◀



Caractéristiques :

- chaque table est un bloc
- il y a au max. **2 ou 3 niveaux**
- les accès directs sont **rapides**
- **insertion** des enregistrements
- fichiers à **trous**
- **couteux** pour les petits fichiers

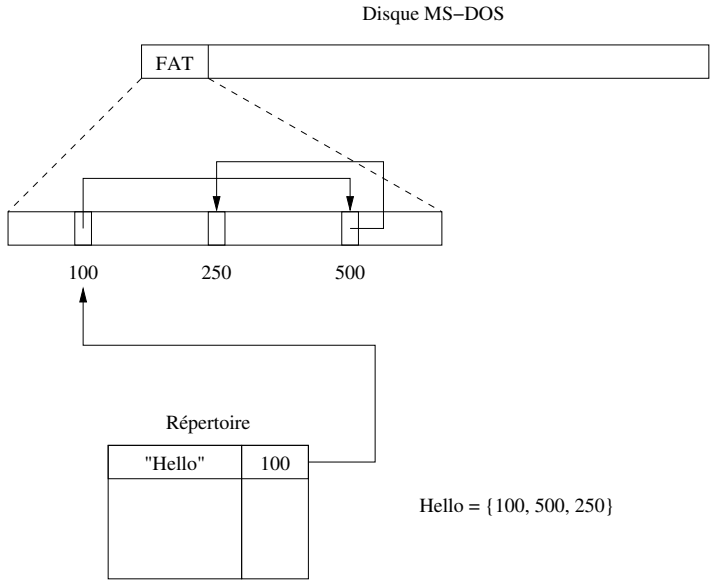
▶ Comparaisons ◀

Avec un bloc de 1 Ko et **250 références** par blocs, on obtient les résultats suivants (en nombre d'E/S) :

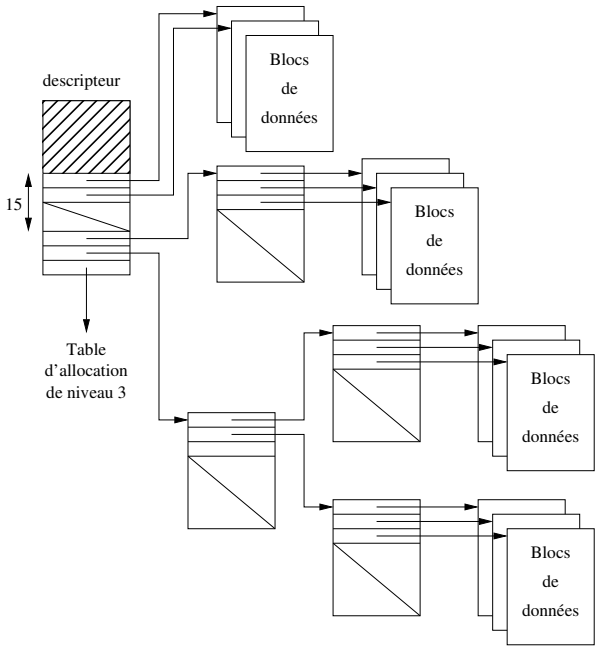
numéro de bloc	blocs chainés	table à un niveau	table à 2 niveaux	2 niveaux avec cache
1	2	2	3	3
10	11	2	3	1
100	101	2	3	1
1000	1001	3	3	2
10000	10001	39	3	2

Dans la **gestion du cache** les **blocs d'index** seront **prioritaires**.

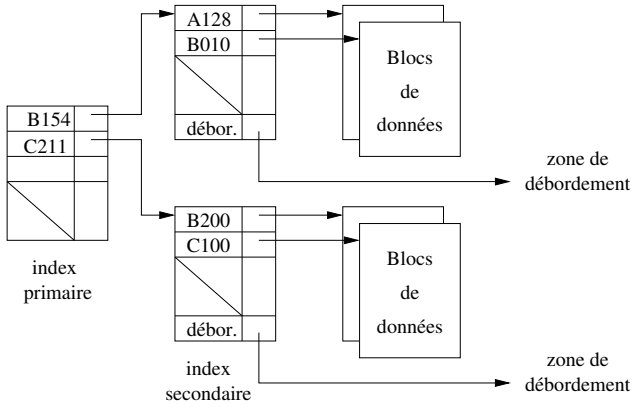
▶ La FAT de Microsoft ◀



▶ UNIX ◀



▶ Le séquentiel indexé ◀



Les blocs d'index et de données sont rangés sur le **même cylindre**.

Une **réorganisation périodique** est possible (voir souhaitable) si les zones de débordement sont **trop importantes**.

▶▶ Protection des fichiers ◀◀

A chaque demande, les **droits**, **limites** et **protections** sont vérifiés.

Pour **sécuriser l'organisation physique** des fichiers, le SGF ajoute des informations redondantes :

- triple chaînage des blocs (père et frères)
- duplication de la table des descripteurs
- duplication de la FAT

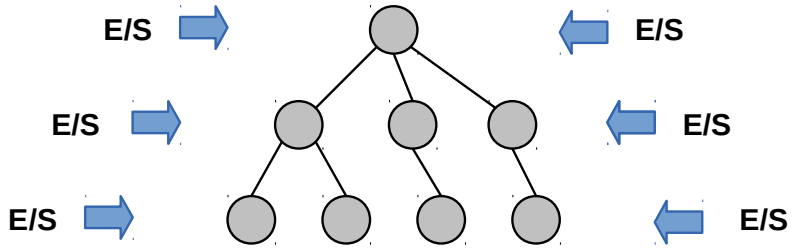
Objectif : pouvoir **reconstruire** ou **réparer** un système de fichiers **incohérent** (`chkdsk` ou `fsck`).

Ces sécurités ne dispensent pas de la **sauvegarde** périodique des fichiers :

- sauvegarde incrémentale,
- sauvegarde totale,
- sauvegarde FIFO sur N jours.

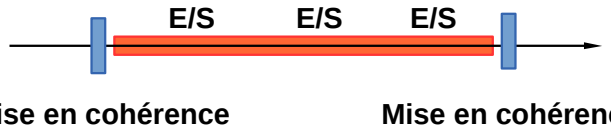
▶ Système de fichiers journalisé ◀

Un système de fichier est une **structure de données persistante** et continuellement **modifiée** par de **nombreux** processus.



▶ Cohérence du SGF ◀

Le SGF sur disque n'est **cohérent** qu'à l'arret des opérations d'entrée/sortie.



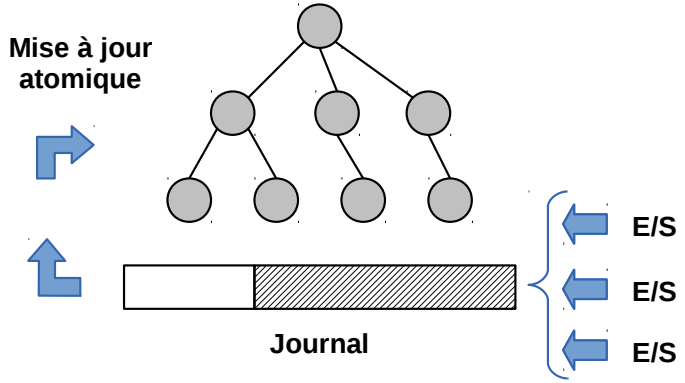
En cas d'arret **intempestif** dans la zone rouge :

- perte d'exploitation,
- **redémarrage** du serveur,
- **réparation** des disques (plusieurs **heures**),
- récupération des données **perdus** (à partir de la sauvegarde),
- mise en exploitation.

▶ Entrée/sortie via un journal ◀

Mise en place d'un **journal** : espace **contiguë** de stockage sur disque (fichier et/ou partition).

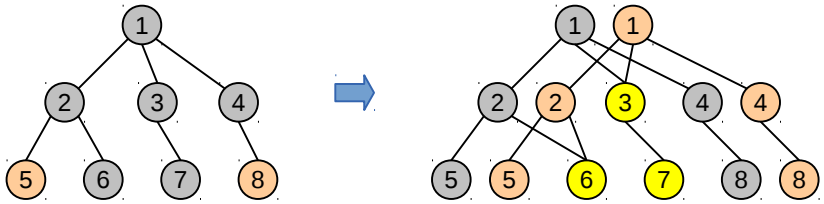
Les opérations **d'écritures** se réalisent dans le **journal** :



Le journal est périodiquement vidé de manière **atomique** afin de **garantir la cohérence** du SGF.

▶ Mise à jour atomique ◀

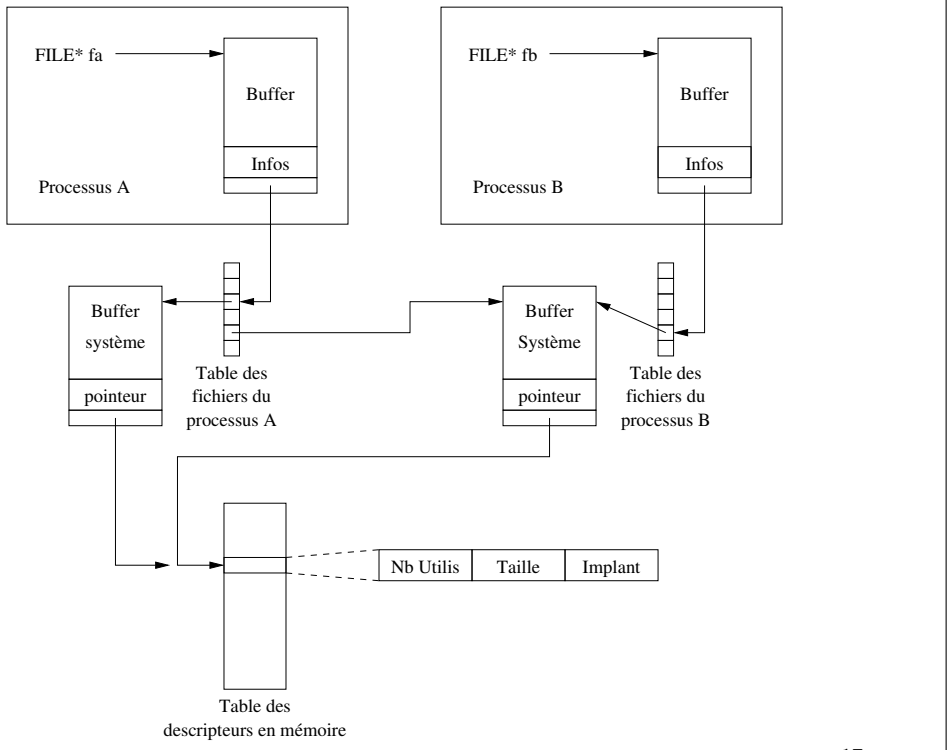
La mise à jour **atomique** est réalisée par la construction d'une nouvelle version :

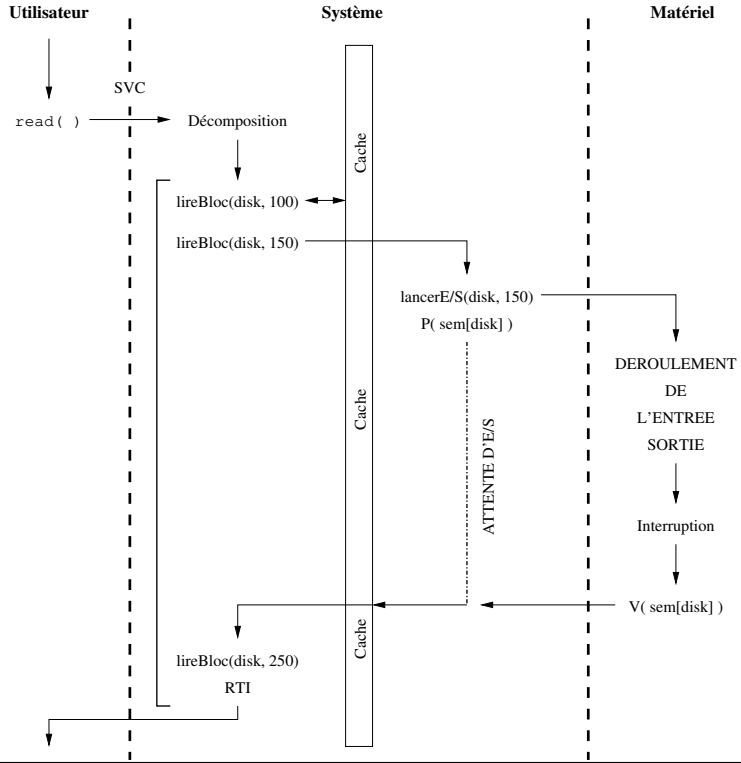


C'est la base d'un SGF :

- avec **clichés** (**snapshots**),
- qui maintient **plusieurs versions** d'un fichier

▶▶ Structure de données du S.G.F ◀◀


FILE* fa
Buffer
Infos
Processus A
FILE* fb
Buffer
Infos
Processus B
Buffer
système
pointeur
Table des
fichiers du
processus A
Buffer
Système
pointeur
Table des
fichiers du
processus B
Nb Utilis
Taille
Implant
Table des
descripteurs en mémoire


Utilisateur
Système
Matériel
SVC
read( )
Décomposition
Cache
lireBloc(disk, 100)
lireBloc(disk, 150)
lancerE/S(disk, 150)
P(sem[disk])
DEROULEMENT
DE
L'ENTREE
SORTIE
ATTENTE D'E/S
Interruption
V(sem[disk])
lireBloc(disk, 250)
RTI