

# Gestion des périphériques

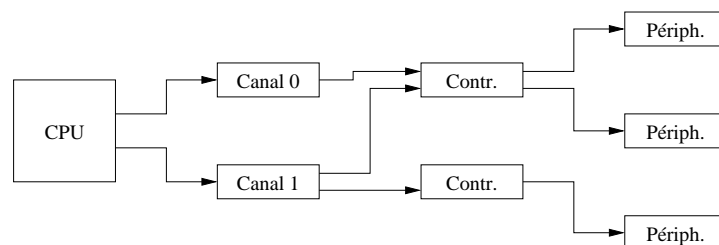
1

## ►► Organisation des périphériques ◀◀

---

Il existe deux catégories de périphérique :

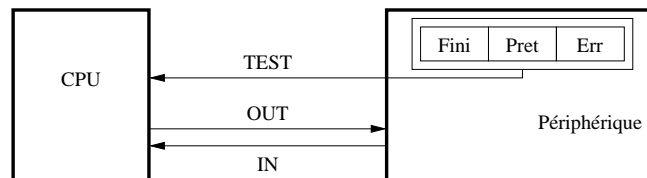
- les périphériques en mode bloc (disque, disquette, CD-ROM, bande),
- les périphériques en mode caractère (carte réseau, imprimante, terminaux, bande, etc.).



⟨nu de canal, nu de control., nu de périph.⟩

2

## ►► Entrée/Sortie par test d'état ◀◀



Gestion des entrées/sorties par attente active :

sortir(valeur  $c$ , périphérique  $p$ ) =

**début**

test  $p$ , R1

**si** (R1.prêt = 0) **alors**

  ⟨erreur⟩

**fin-si**

out  $c$ ,  $p$

**répéter**

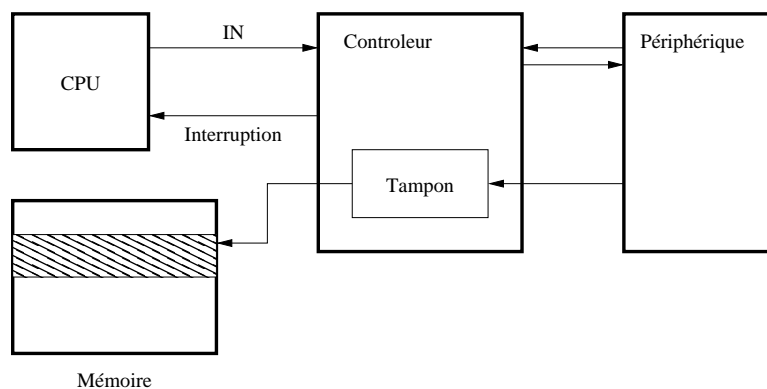
  test  $p$ , R1

**jusqu'à** (R1.fini ou R1.erreur)

**fin**

3

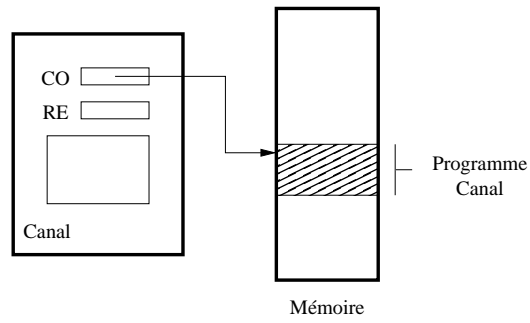
## ►► Entrée/Sortie par interruption (ADM) ◀◀



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

4

## ►► Entrée/Sortie par interruption (CANAL) ◀◀



```
<Soit  $c$  un numéro de canal>  
<préparer le programme à l'adresse  $\alpha$ >  
canal_execute  $c, \alpha$   
⋮  
canal_test  $c, R1$   
canal_stop  $c$ 
```

# Le système de gestion de fichiers

1

---

## ►► Notion de fichier ◀◀

---

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

2

## ►► Les fonctions du S.G.F. ◀◀

---

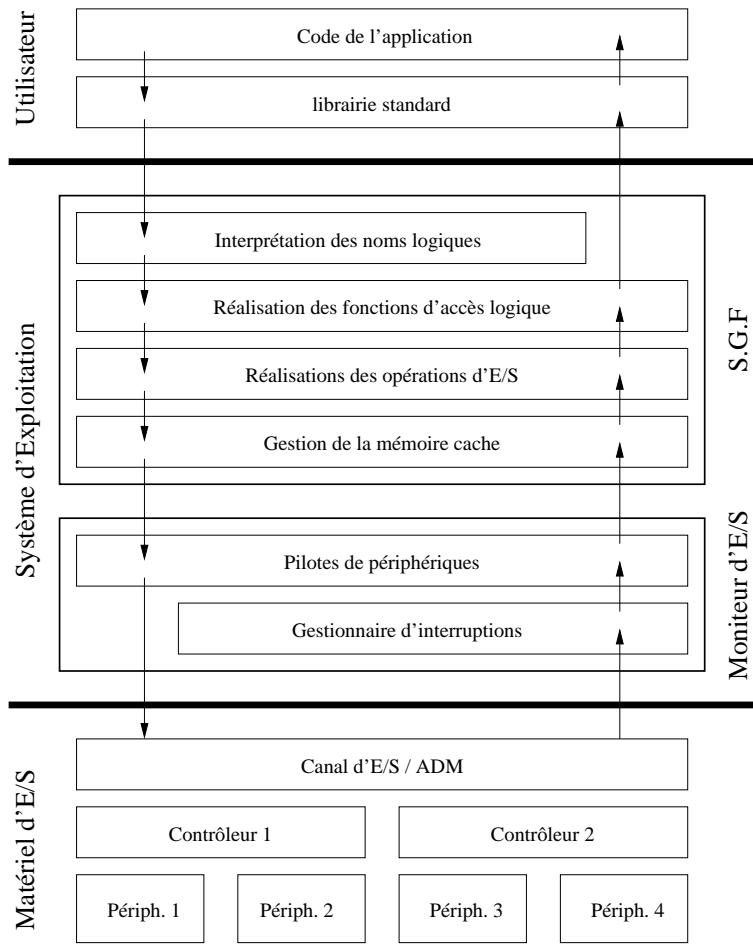
Fourniture des fonctions d'accès au niveau logique qui réalisent :

- le passage du niveau logique au niveau physique,
- le partage et la protection des informations.

Gestion de la mémoire secondaire qui est une ressource partagée entre tous les fichiers.

## ►► La structure interne du S.G.F. ◀◀

---



# Organisation des disques

1

---

## ►► Structure physique d'un support ◀◀

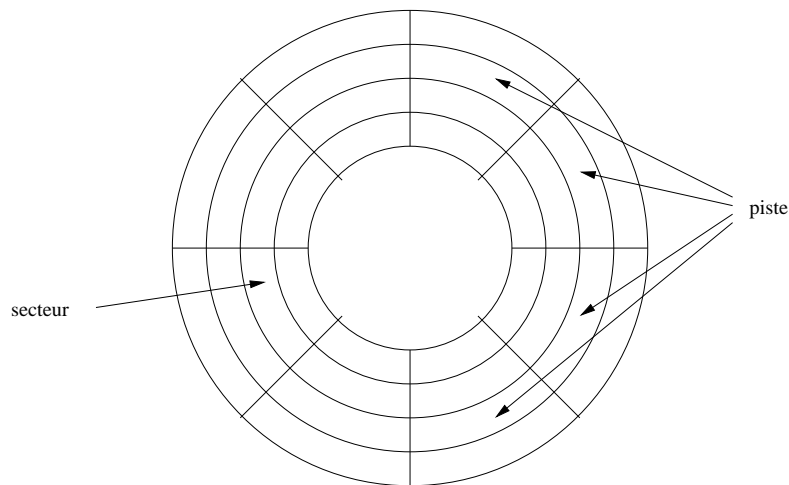
- Faces, pistes et secteurs
- Cylindres et autres...
- Lecture / Écriture d'un bloc
- Optimisation des requêtes disque
- Gestion du support
- Gestion du cache
- Implantation du cache

2

## Faces, pistes et secteurs

---

Un disque est composé de deux *faces*. Ces faces sont découpées en *pistes* (de 20 à 1500).



On trouve de 4 à 32 *secteurs* par piste dont la taille varie de 512 octets à 4 Ko.

L'adresse d'un secteur est un triplet :

$$\langle \text{face } f, \text{ piste } p, \text{ secteur } s \rangle.$$

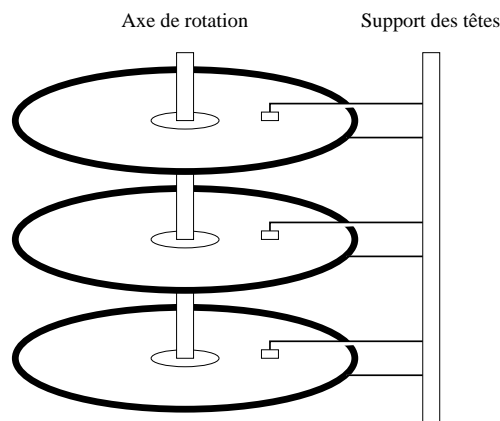
3

## Cylindres et autres...

---

Un *support* est composé de plusieurs disques

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



Le *bloc* est l'unité élémentaire d'E/S. Il regroupe les secteurs de même numéro à l'intérieur d'un cylindre.

$$\text{le bloc } b_i \text{ d'adresse } \underbrace{\langle i \text{ div } N_f \rangle}_{\text{cylindre}}, \underbrace{\langle i \text{ mod } N_f \rangle}_{\text{secteur}}.$$

avec  $N_f$  le nombre de secteurs par face.

4

## 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.

$$\langle \text{temps d'E/S} \rangle = \langle \text{temps de latence} \rangle + \langle \text{E/S} \rangle$$

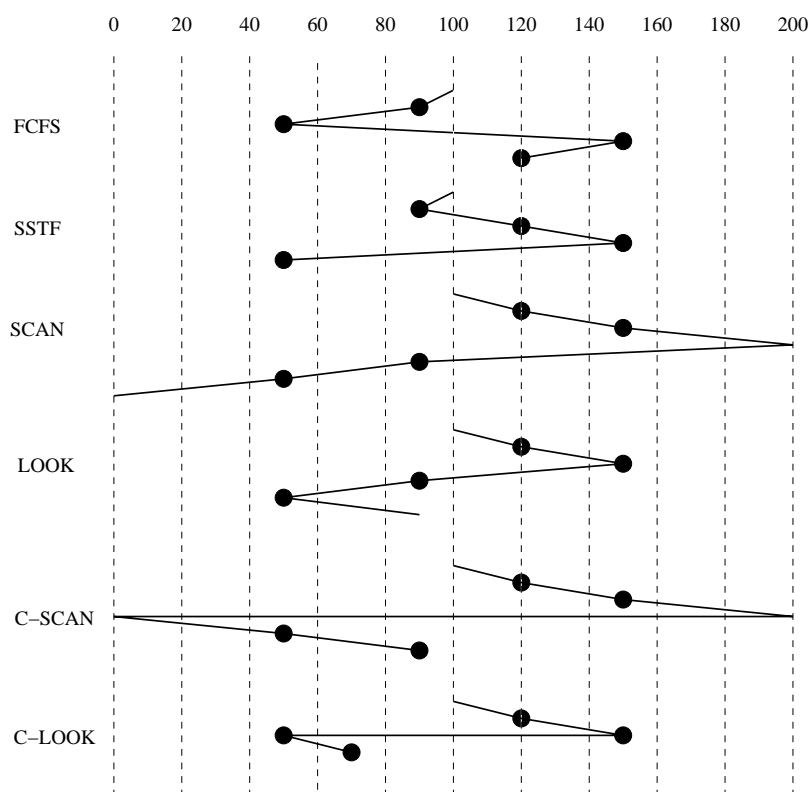
Pour diminuer le temps de latence on applique un algorithme d'*ordonnement des requêtes disques*.

- 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).

5

## Optimisation des requêtes disque

---



6

## 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,
- un chaînage de blocs d'index,
- une table de bits  $B$  telle que  $B_k = 1$  ssi le bloc  $k$  fait partie de l'ensemble.

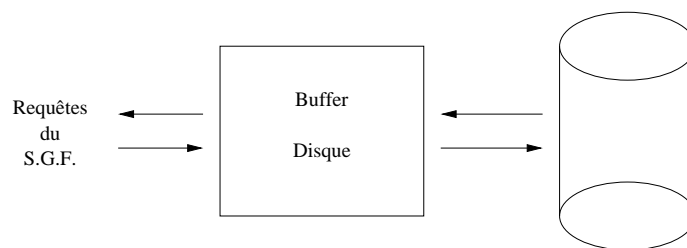
Pour un disque de 1 Go et un bloc de 4 Ko, la table mesure

$$\left(\frac{2^{10} \times 2^{10}}{2^2}\right) \times \frac{1}{2^{10} \times 2^3} = \frac{2^{20}}{2^{15}} = 32\text{Ko}.$$

7

## 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 disque pose le problème de la **cohérence** des informations sur disque.

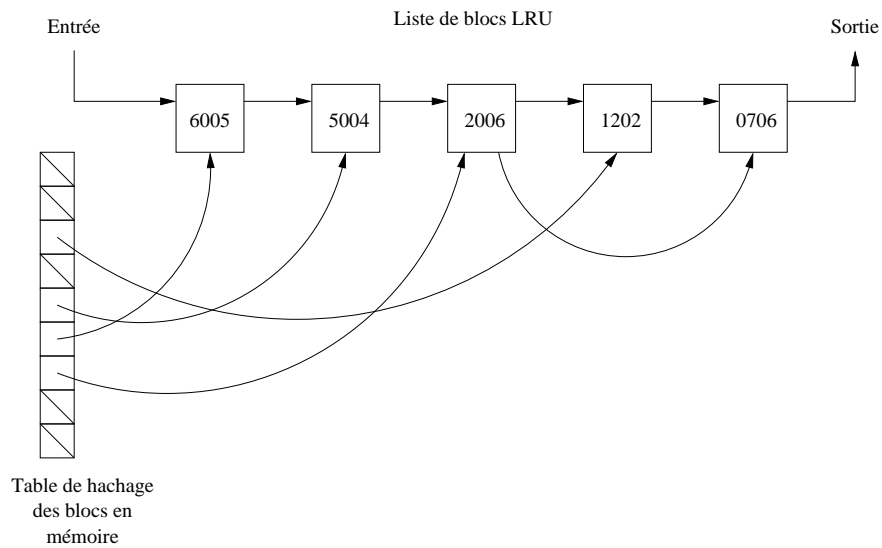
Ce problème est réglé par la mise à jour périodique du disque (par exemple toutes les 30 secondes pour UNIX ou immédiatement pour MS-DOS).

8

## 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.

9

---

## ►► Structure logique des disques ◀◀

---

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

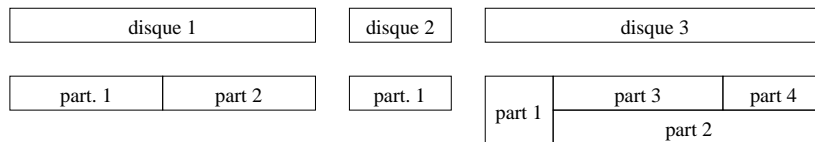
10

## 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 S.E.

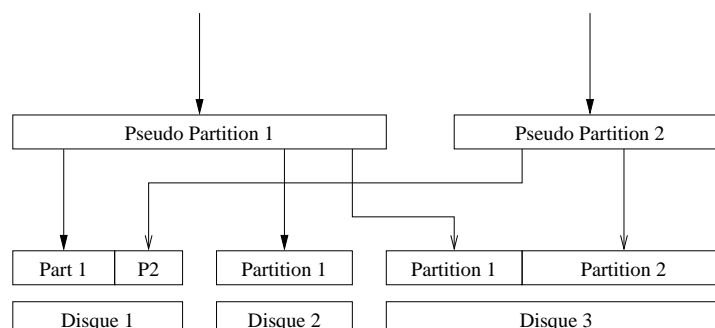
11

## Pseudo Partition

---

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

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

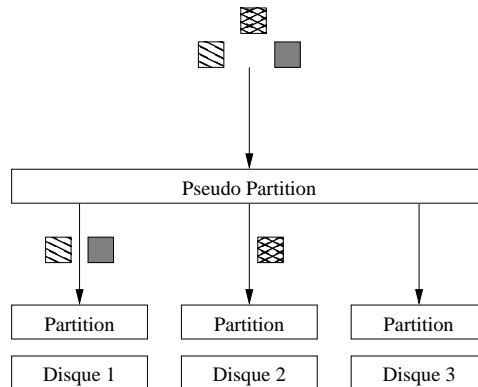


12

## *Pseudo Partition (augmentation)*

---

On peut faire croître une **pseudo partition** en ajoutant plusieurs **partitions logiques**.

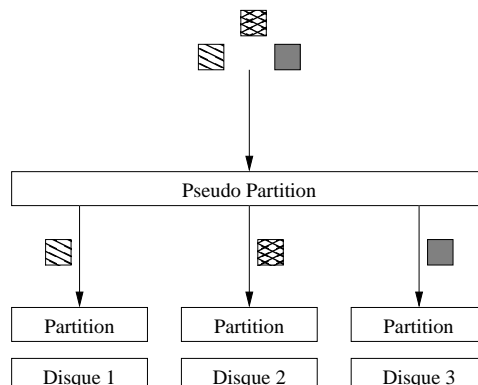


13

## *Pseudo Partition (répartition)*

---

Les blocs sauvés dans une pseudo partition sont répartis sur tous les disques dans un souci d'efficacité.

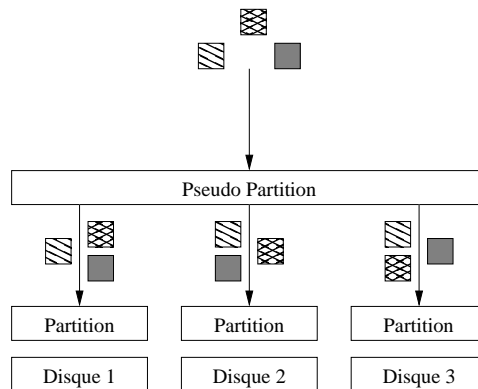


14

## *Pseudo Partition (miroir)*

---

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

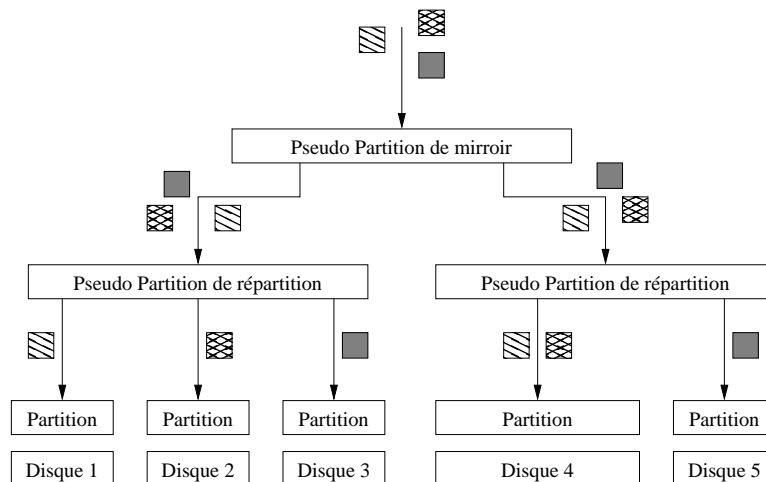


15

## *Pseudo Partition (combinaison)*

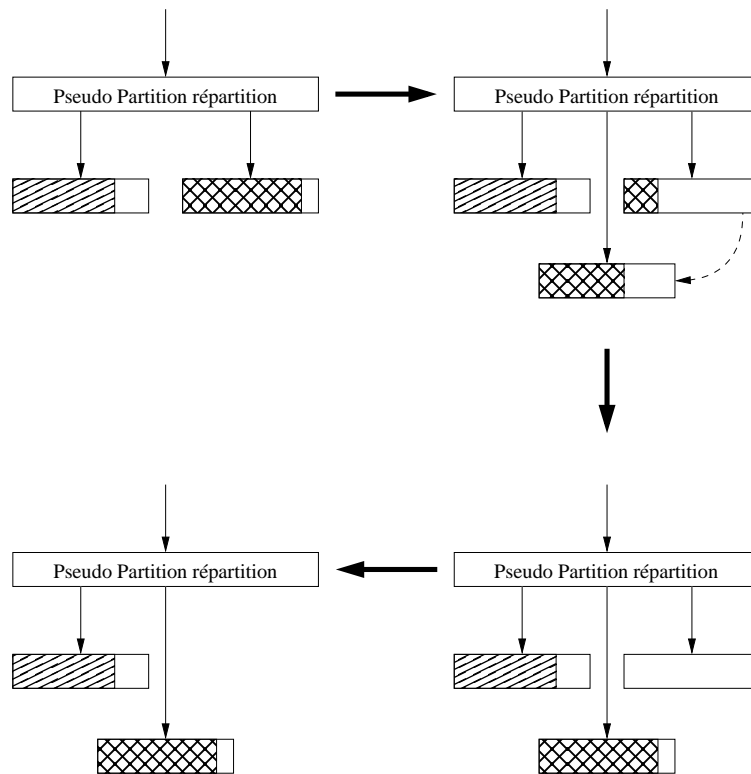
---

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



16

La **configuration** des pseudo-partitions peut être modifiée dynamiquement :



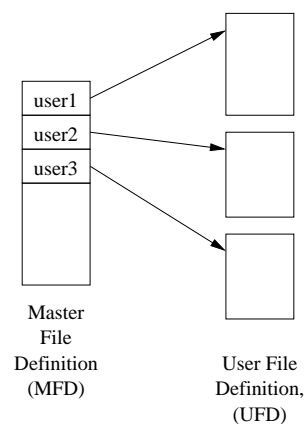
17

## Répertoires simples

---

Les *répertoires uniques* constituent le premier moyen pour représenter un ensemble de fichier.

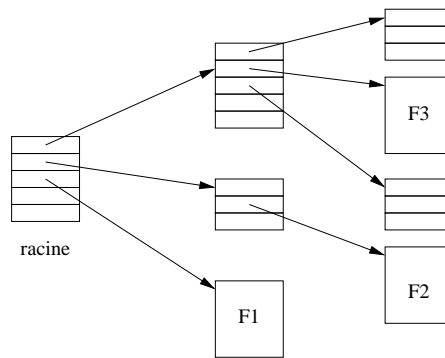
Les *répertoires à deux niveaux* font intervenir l'utilisateur comme clé d'accès.



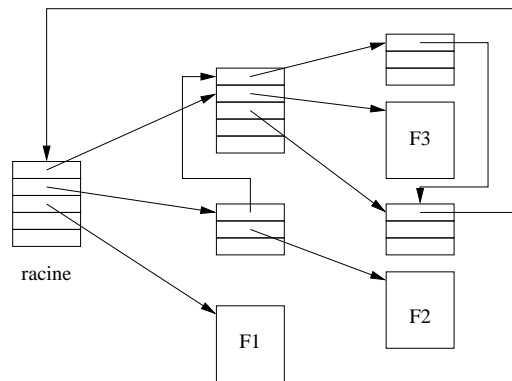
18

## Répertoires arborescents

---



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



19

### codage des répertoires :

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

# Le système de gestion de fichiers

1

---

►► Organisation logique des fichiers ◄◄

2

## Présentation

---

Il existe deux cas de figure :

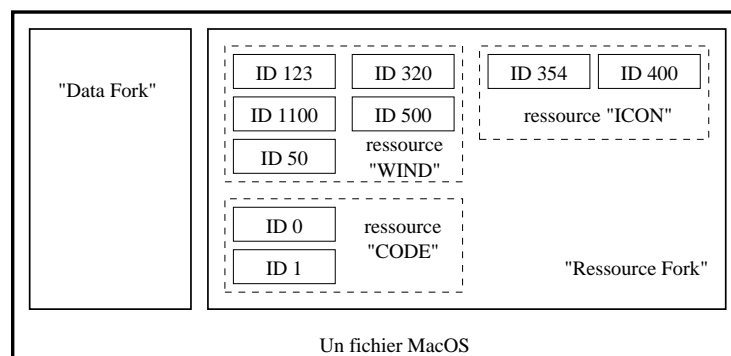
- Le S.E. connaît la structure logique des fichiers (Macintosh, IBM, ...),
  - + les possibilités sont plus importantes,
  - le S.E. est plus complexe.
- Le S.E. considère les fichiers comme des flots d'octets (MS-DOS, Unix, ...).
  - + le système est plus simple,
  - pas d'assurance sur la nature des fichiers.

Un fichier est constitué d'*enregistrements*. Chaque enregistrements est formé d'*attributs*.

3

## Org. logique : L'exemple de MacOS

---



Les fonctions d'accès :

```
lire_ressource(type, id, var r : ressource),  
écrire_ressource(type, id, r : ressource),  
détruire_ressource(type, id),
```

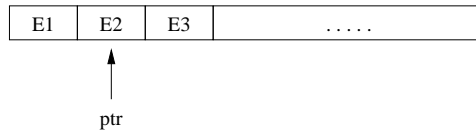
4

## L'accès séquentiel

---

Le fichier se présente comme un *ensemble ordonné* d'enregistrements.

On dispose d'un pointeur *ptr* vers l'un des enregistrements.



Les fonctions d'accès sont :

lire(F : **fichier**, var e : enregistrement)  
écrire(var F : **fichier**, e : enregistrement)  
ajouter(var F : **fichier**, e : enregistrement)  
tronquer(var F : **fichier**)  
se\_déplacer(var F : **fichier**, depl : **entier**)

## L'accès direct

---

## L'accès direct par rang

---

Un *fichier à accès direct par rang* est un tableau d'enregistrements repérés par leur rang.

### **Caractéristiques :**

- On dispose d'un pointeur *ptr* vers l'un des enregistrements.
- L'*adresse logique* est représentée par le rang.
- Le niveau logique est contiguë.

Les fonctions d'accès sont :

```
lire(F : fichier, var e : enreg.)  
écrire(F : fichier, e : enreg.)  
insérer(F : fichier, e : enreg.)  
se_déplacer(var F : fichier, rang : entier)  
tronquer(var F : fichier)
```

7

## L'accès direct par clé

---

Un *fichier à accès direct par clé* est un ensemble de couples (clé, enregistrement)

### **Caractéristiques :**

- La clé *identifie* l'enregistrement.
- Une *clé unique* identifie au plus un enregistrement.
- Une clé unique peut servir *d'adresse logique*.

Les fonctions d'accès sont :

```
lire(F : fichier, clé : donnée, var e : enreg.)  
ajouter(var F : fichier, clé : donnée, e : enreg.)  
supprimer(var F : fichier, clé : donnée)  
remplacer(var F : fichier, clé : donnée, e : enreg.)
```

8

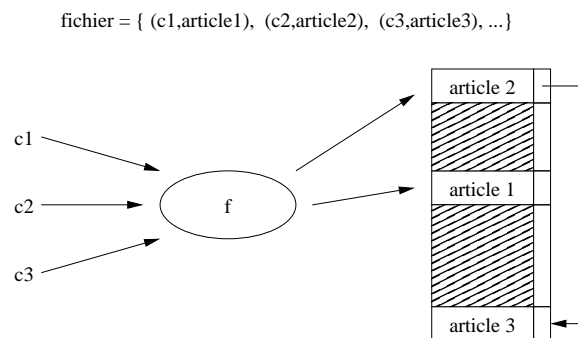
## Adressage dispersé (hash-coding)

---

On se donne  $f$  une *fonction de dispersion* qui est définie de la manière suivante :

- pour chaque clé  $c$ , nous avons  $f(c) \in N$  et  $0 \leq f(c) \leq \langle \text{nb d'enregistrements} \rangle$
- si  $c_1 = c_2$ , alors  $f(c_1) = f(c_2)$ ,

On appelle *collision*, l'existence de deux clés  $c_1$  et  $c_2$  telles que  $c_1 \neq c_2$  et  $f(c_1) = f(c_2)$ .



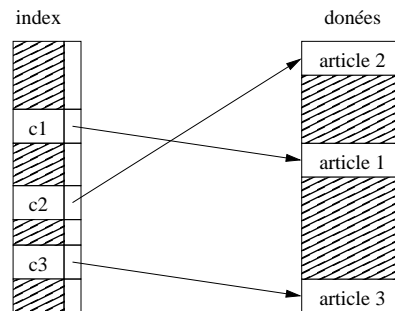
### Inconvénients :

- il est difficile de choisir  $f$ ,
- sur certains système perte de place,
- réorganisation périodique du fichier.

## Adressage direct par indexation

---

Dans l'*indexation*, on accède aux enregistrements par le biais d'une table appelée *index* qui contient l'ensemble des clés



La recherche d'un enregistrement nécessite en moyenne

- $n/2$  lectures si l'index n'est pas trié,
- $\log_2 n$  lectures si l'index est trié.

10

## Le type des fichiers

---

Le S.E. peut associer un *type* à chaque fichier. Ce type précise le **contenu** et la **structure logique** du fichier.

### Avantages :

- définition d'opérations partielles,
- optimiser le codage et le traitement,
- meilleur sécurité.

### Inconvénients :

- Quel type pour quel fichier ?
- manque de portabilité des applications,
- lourdeur du S.E.

11

## Le type des fichiers (2)

---

Il existe au moins deux types de fichier :

- Les **répertoires** :

```
chercher(nom : chaîne, var e : fichier)
ajouter(nom : chaîne, e : fichier)
supprimer(nom : chaîne)
```

- Les fichiers de données,

- Les **exécutables** :

```
exécuter(e : exécutable)
chercher(e : exécutable, fonction : chaîne)
```

12

## Systeme de fichiers sémantique

---

Un fichier texte structuré :

```
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
news:x:9:13:news:/etc/news:
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
```

Peut être vu comme un répertoire :

```
[saphir]$ ls -F
bin/ daemon/ games/ news/ operator/
[saphir]$
```

Pour être exploité :

```
[saphir]$ ls -F daemon/
passwd uid gid gecos home shell
```

OU

13

```
[saphir]$ ls -F operator/home/  
... les fichiers de l'opérateur ...
```

---

**►► Organisation physique des fichiers ◀◀**

## Structure d'un descripteur

---

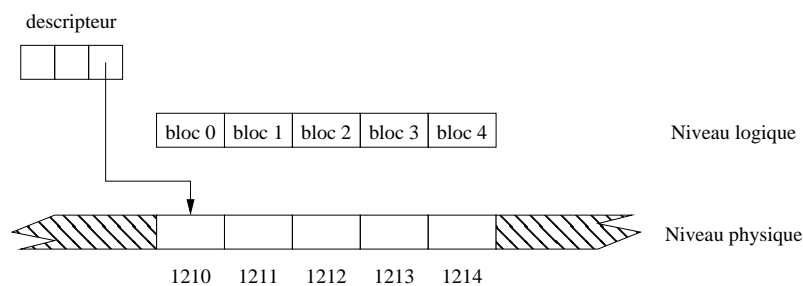
Dans le descripteur on trouve les informations suivantes :

- informations d'état (ouvert/fermé, nombre d'utilisateurs),
- informations sur le contenu du fichier (type : fichier de texte, exécutable, répertoire, . . .),
- informations sur l'organisation logique (taille et structure des enregistrements),
- statistiques d'utilisation (date de dernière modification, nombre d'utilisations, intervalle de temps entre deux utilisations),
- informations sur l'implantation physique du fichier.

15

## Implantation contiguë

---



### Avantages :

- le passage logique → physique est simple,
- l'accès séquentiel ou direct est rapide.

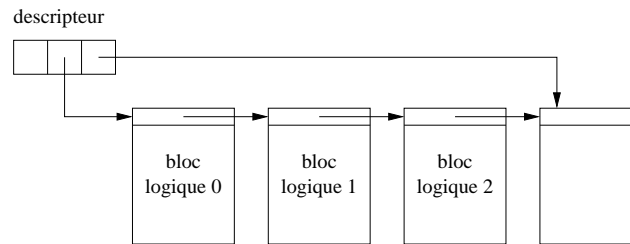
### Inconvénients :

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

16

## Implantation par blocs chaînés

---

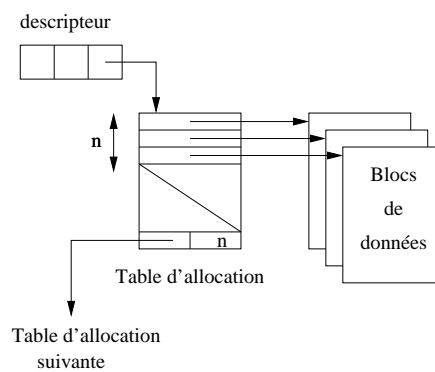


- les accès séquentiels sont efficaces,
- les accès directs sont impossibles,
- on trouve ce système sur les disquettes.

17

## Tables d'allocation

---

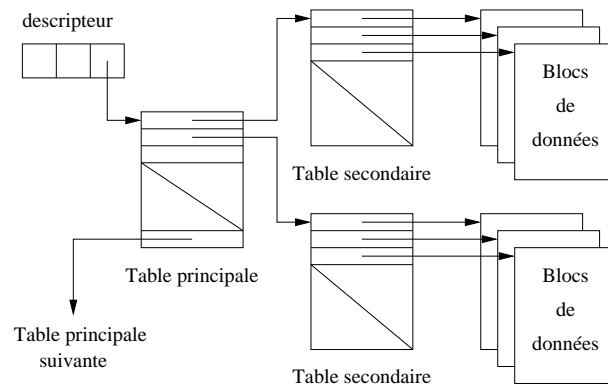


- les accès séquentiels sont efficaces,
- les accès directs sont améliorés,
- la taille des fichiers est limitée.

18

## Tables à plusieurs niveaux

---



- il y a au maximum 2 ou 3 niveaux,
- les accès directs sont rapides,
- on peut insérer des enregistrements.

19

## Comparaisons

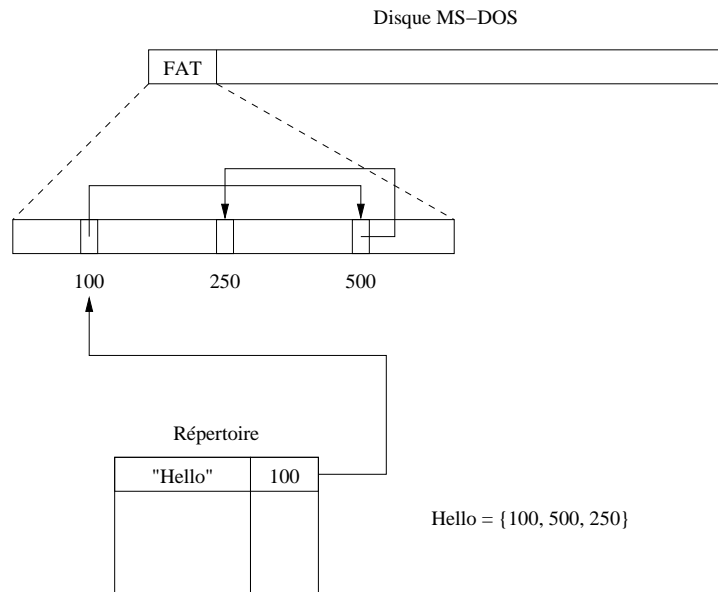
---

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

nu de bloc	chaîne	1 <sup>er</sup>	2 <sup>ème</sup>
1	2	2	3
10	11	2	3
100	101	2	3
1000	1001	3	3
10000	10001	39	3

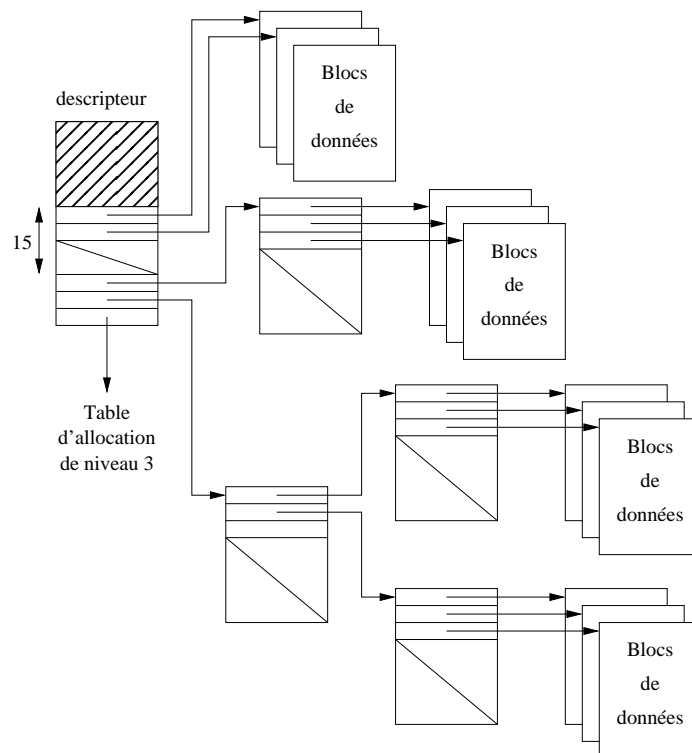
20

## Un exemple : MS-DOS



21

## Un exemple : UNIX

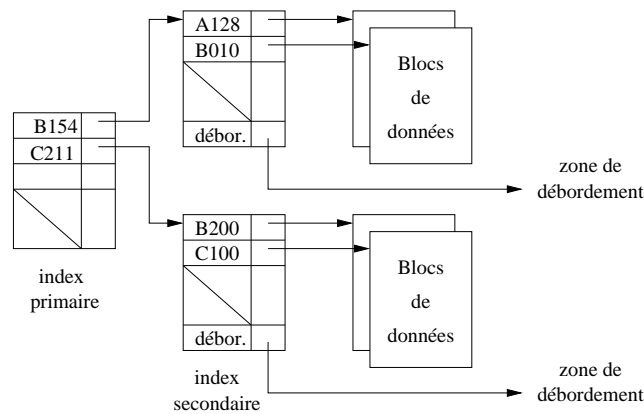


Capacité  $\geq 2^{32}$

22

## Les fichiers séquentiels indexés

---



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

Une réorganisation périodique est possible si les zones de débordement sont trop importantes.

23

## Protection des fichiers

---

À chaque demande faite au SGF, les droits, limites et protections sont vérifiées.

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

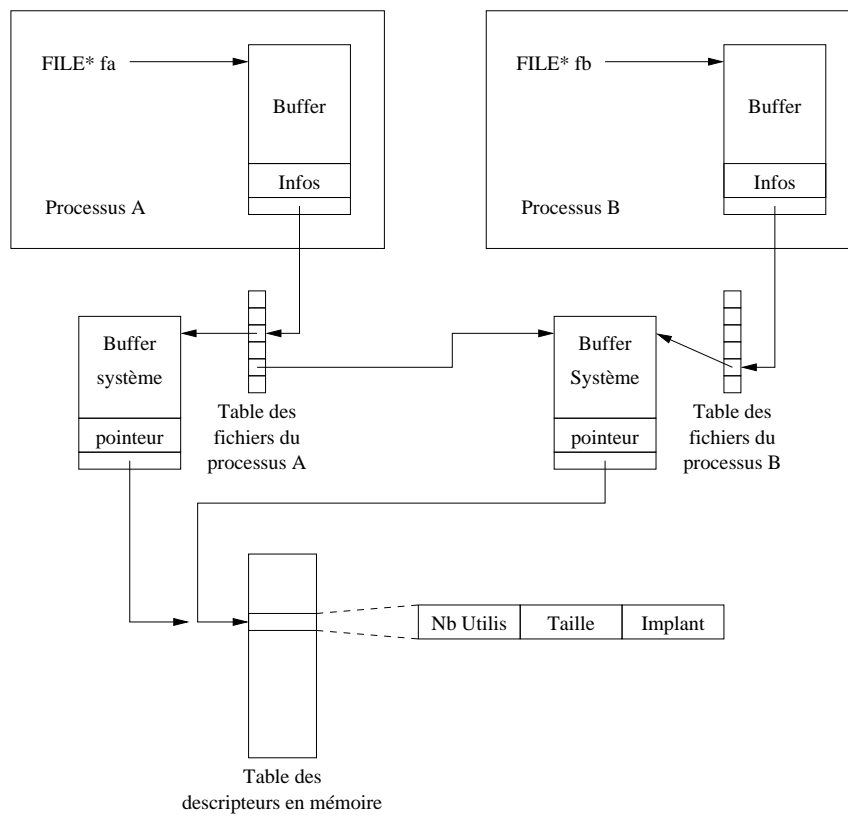
- double chaînage des blocs,
- recopie du descripteur en fin de fichiers,
- chaque nœud pointe vers son père,
- placer une info libre/occupé dans les blocs.

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

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

24

## ►► Structure de données du S.G.F ◀◀



25

## Décomposition d'une opération

26

