Chapitre 4 SYSTEME DE FICHIERS

1. CONCEPTION D'UN SYSTEME DE FICHIERS

1.1 Fichiers

Un fichier est vu comme une suite d'**articles** ou d'**enregistrements logiques** d'un type donné qui ne peuvent être manipulés qu'au travers d'opérations spécifiques. Généralement, elles sont traduites par le SE en opérations élémentaires.

On dispose généralement des opérations suivantes :

fichiers : créer, ouvrir, fermer, détruire, pointer au début, renommer, copier dans un autre fichier, éditer le contenu

articles : lire, écrire, modifier, insérer, détruire, retrouver

Définition : un système de fichiers (SGF) est l'entité regroupant les fichiers mémorisés sur disque. Il contient les données des fichiers et un ensemble d'informations techniques.

Exemple : le SGF d'UNIX. Ses principales caractéristiques sont les suivantes :

- aucune structure interne : les fichiers sont de simples suites d'octets dont chacun peut être adressé individuellement. L'enregistrement logique a la taille d'un octet

- l'expansion des fichiers est dynamique
- les répertoires forment une structure hiérarchique
- l'utilisation est commune pour les fichiers de données et les fichiers spéciaux

Le **superbloc** d'UNIX contient notamment les informations suivantes :

- taille du système de fichier
- nombre de blocs libres
- pointeur sur le premier bloc libre dans la liste des blocs libres
- taille de la liste des i-nodes
- nombre et liste des i-nodes libres

UNIX utilise une technique de zones tampons en MC afin d'optimiser l'utilisation du disque. Une partie du superbloc et de la table des i-nodes est chargée en MC lors de l'exécution de la commande ***mount***. Toute opération d'E/S se fera à travers ces tables afin d'optimiser l'utilisation du disque. Lorsqu'un programme met à jour des informations d'un fichier, cela se fait à travers les zones tampons de la MC. Régulièrement, ces zones sont recopiées sur disque. Les zones concernées du superbloc sont mises à jour. Cette situation est critique car la version du superbloc en MC et celle sur disque peuvent être différentes jusqu'à recopie sur disque de la partie du superbloc qui est en MC.

1.2 Organisation de l'espace disque

Il existe deux stratégies pour stocker un fichier de n octets sur disque :

- **allouer des secteurs contigus** totalisant une capacité d'au-moins n octets. Avec deux inconvénients :
  - le dernier secteur a toutes chances d'être sous-utilisé et ainsi, on gaspille de la place. Le pourcentage de place perdue est d'autant plus grand que la taille moyenne des fichiers est faible, ce qui est la réalité
  - si le fichier est agrandi, il faudra déplacer le fichier pour trouver un nouvel ensemble de secteurs consécutifs de taille suffisante

- **diviser le fichier en blocs** de taille fixe, insécables, que le SE alloue de façon non nécessairement contiguë à des secteurs.

Un **bloc** est défini comme une zone de mémoire secondaire contenant la quantité d'information qui peut être lue ou écrite par un périphérique en une seule opération. La taille d'un bloc est donc attachée à un périphérique d'E/S et fixée par le matériel. Cependant, si le SE n'a pas le choix de la taille d'un bloc, il peut grouper plusieurs article dans un même bloc (packing).

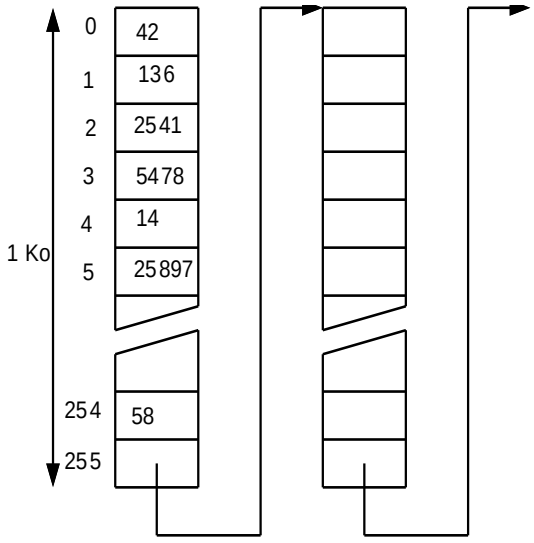
Pour des blocs de taille < 1 Ko, le taux de remplissage du disque est excellent, mais la vitesse de transfert des données est modeste (délai de rotation du disque et de recherche importants par rapport au temps de transfert lui-même). Au-delà de 1 Ko, le taux de remplissage se dégrade, mais la vitesse de transfert s'améliore.

1.3 Gestion des blocs libres

Dès qu'on a choisi une taille de blocs (souvent 1/2 Ko à 2 Ko), on doit trouver un moyen de mémoriser les blocs libres. Les deux techniques les plus répandues sont les suivantes :

- **table de bits (bit map):** on gère une table comportant autant de bits que de blocs sur le disque. A chaque bloc du disque, correspond un bit dans la table, positionné à 1 si le bloc est occupé, à 0 si le bloc est libre (ou vice versa). Par exemple, un disque de 300 Mo, organisé en blocs de 1 Ko, sera géré par une table de 300 Kbits qui occupera 38 des 307.200 blocs

- **liste chaînée de n° des blocs :** par exemple, un disque de 300 Mo, organisé en blocs de 1 Ko : supposons que chaque bloc soit adressé par 4 octets. Chaque bloc de la liste pourra contenir 255 adresses de blocs libres. La liste comprendra donc au plus 307.200/255 = 1205 blocs. Cette solution mobilise beaucoup plus de place que la précédente.



On peut imaginer deux variantes de cette dernière solution :

- **une liste chaînée de blocs :** chaque bloc libre pointe sur le bloc libre suivant

- **une liste chaînée de zônes libres :** chaque bloc de début d'une série de blocs libres (zone libre) contient la taille de la zone et pointe sur le premier bloc de la zone libre suivante.

1.4 Allocation de blocs pour le stockage des fichiers

Quatre méthodes sont utilisées selon les SE :

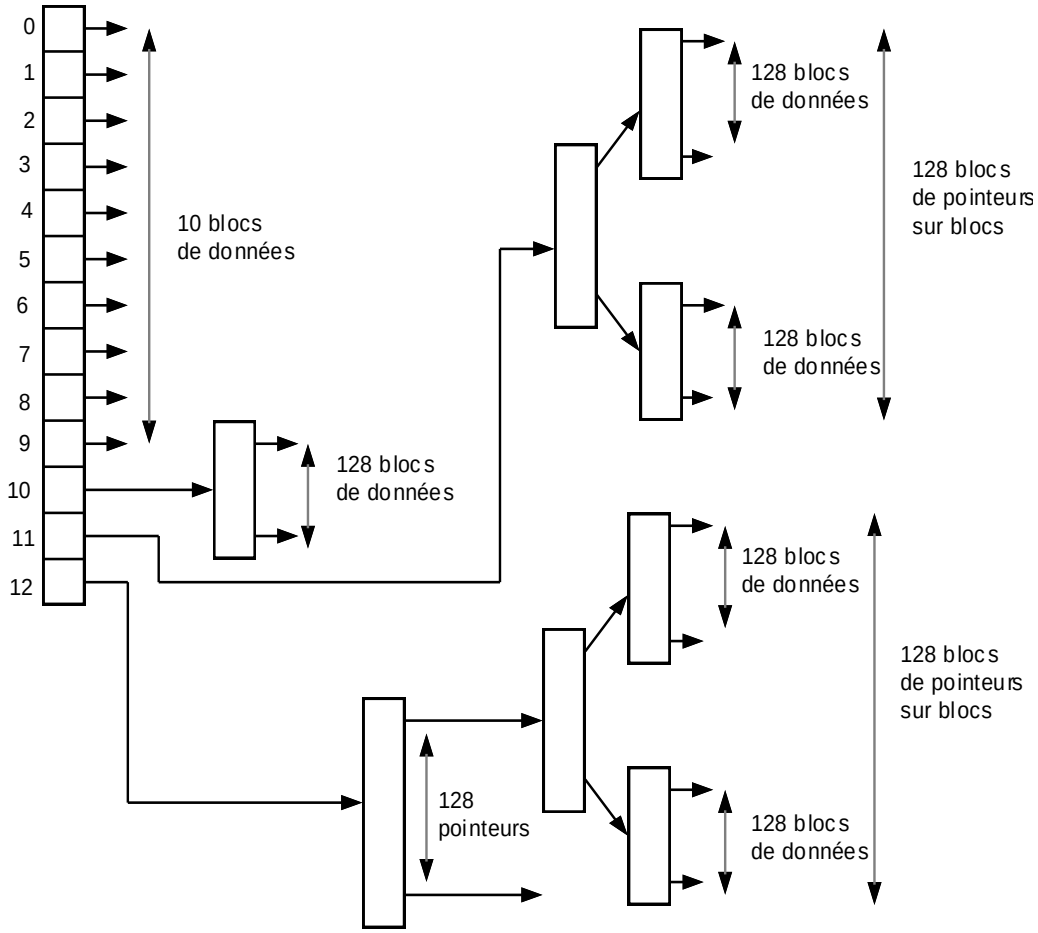
- **allocation contiguë :** on retrouve les solutions utilisées pour l'allocation de mémoire (algorithmes de la première zone libre, algorithme du meilleur ajustement, algorithme du meilleur résidu). L'implantation physique est proche de la vision logique. On limite également les déplacements de la tête le lecture/écriture, coûteux en temps. Toutefois, il y a deux inconvénients :
  - le risque d'absence d'une zone libre de taille suffisante,
  - le déplacement possible du fichier, lorsqu'il s'agrandit, vers une zone libre de taille suffisante.

- **allocation non contiguë chaînée :** chaque bloc du fichier contient un pointeur sur le bloc suivant. Lorsque le fichier change de taille, la gestion des blocs occupés est simple. Il n'y a aucune limitation de taille, si ce n'est l'espace disque lui-même. Deux inconvénients toutefois:
  - la perte d'un pointeur entraîne la perte de toute la fin du fichier ou bien il faut un double chaînage par sécurité
  - le mode d'accès est totalement séquentiel

- **allocation non contiguë indexée :** on peut envisager 3 solutions :
  - une table globale d'index : elle évite la dispersion des pointeurs en rassemblant la liste des numéros de blocs utilisés par tous les fichiers. Cette table occupe beaucoup de place.
  - une liste chaînée : dans le SE, donne la liste des blocs utilisés par les fichiers
  - une table locale d'index : une table, en début de chaque fichier, contient l'ensemble des blocs utilisés. Il est donc aisé d'effectuer un accès direct à un bloc d'un fichier, ou de faire évoluer dynamiquement la taille d'un fichier. Quand cette table est petite, elle est stockée dans un bloc spécial ou **bloc d'index**. La taille des blocs d'index est une question difficile : s'ils sont trop petits, ils limiteront fortement la taille d'un fichier; s'ils sont trop grands, une grande partie de leur contenu est inutilisée ("fragmentation interne").

- **allocation non contiguë mixte :** on peut améliorer la solution précédente en utilisant plusieurs niveaux de tables d'index. Citons par exemple la méthode utilisée par UNIX :
  - chaque fichier se voit affecter une table à 13 entrées
  - les entrées 0 à 9 pointent sur un bloc de données
  - l'entrée 10 pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc de données (simple indirection)
  - l'entrée 11 pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc de données (double indirection)

- l'entrée 12 pointe sur un bloc d'index qui contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc d'index dont chacun contient 128 ou 256 pointeurs sur bloc de données (triple indirection)



Pour les fichiers les plus longs, trois accès au disque suffisent pour connaître l'adresse de tout octet du fichier. Avec des blocs de 2 Ko, on peut aller jusqu'à 128 Go par fichier, mais UNIX limite la taille à 16 Go.

2. CORRESPONDANCE FICHIERS LOGIQUES-FICHIERS PHYSIQUES

Un **répertoire** est une table ou une liste chaînée dans laquelle le système range, pour chaque fichier :

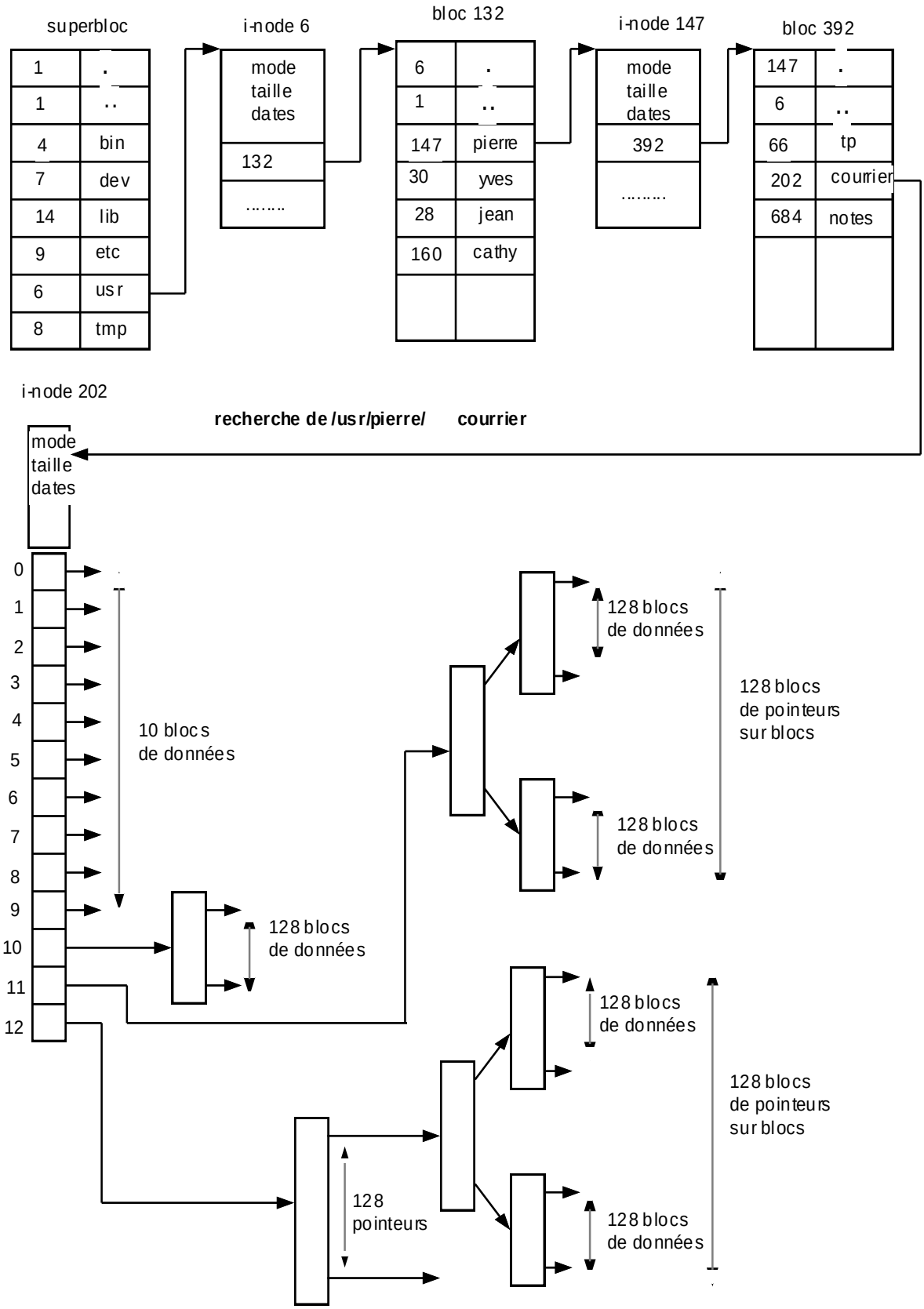
- les informations nécessaires à son exploitation : droits, nom, dates, etc...
- les informations sur son implantation : taille, adresse de la table des index ou du premier bloc ou ...

Il existe diverses solutions d'implémentation d'un répertoire :

- **répertoire à un niveau :** la table est gérée comme une liste. C'est une solution acceptable pour un SE mono-utilisateur, trop contraignant pour un SE multi-utilisateur (deux fichiers ne peuvent porter le même nom)
- **répertoire à deux niveaux :** chaque utilisateur possède son propre répertoire et il existe un répertoire de niveau supérieur qui associe à un nom d'utilisateur l'adresse de son répertoire.
  - Inconvénient : l'utilisation d'un fichier partagé (exemple : fichier de commande) nécessite soit sa recopie dans le répertoire de chacun de ses utilisateurs, soit l'énoncé de son chemin complet. Une autre solution est utilisée pour les fichiers système : on crée un utilisateur fictif qui contient tous les fichiers système, voire d'autres fichiers partagés. Quand un fichier est indiqué par un utilisateur, s'il n'existe pas dans son répertoire, le SE le cherche dans le répertoire fictif.

- **répertoire à structure d'arbre :** c'est le cas par exemple d'UNIX : *cf. schéma page suivante*

- **répertoire à structure de graphe sans cycle :** un fichier peut être dans les sous-arbres de plusieurs utilisateurs. Si un utilisateur le met à jour, il est modifié pour tous les autres. Un tel fichier possède plusieurs chemins d'accès, d'où un algorithme compliqué pour l'effacer. Au total, un répertoire peut contenir un pointeur sur un fichier ou un répertoire qui n'est pas dans le sous-arbre dont il est racine.


superbloc
i-node 6
bloc 132
i-node 147
bloc 392
1 .
1 ..
4 bin
7 dev
14 lib
9 etc
6 usr
8 tmp
mode
taille
dates
132
6 .
1 ..
147 pierre
30 yves
28 jean
160 cathy
392
147 .
6 ..
66 tp
202 courrier
684 notes
i-node 202
recherche de /usr/pierre/ courrier
0
1
2
3
4
5
6
7
8
9
10
11
12
10 blocs
de données
128 blocs
de données
128 blocs
de pointeurs
sur blocs
128 blocs
de données
128 blocs
de données
128 blocs
de pointeurs
sur blocs
128
pointeurs
128 blocs
de données