CHAPITRE VI : SYSTEME DE GESTION DE FICHIERS

6.1 INTRODUCTION :

Afin de fournir un accès efficace et pratique au disque, le SE impose un système de gestion de fichiers (SGF) pour permettre de stocker, localiser, et récupérer facilement des données. Un SGF pose deux problèmes de conception très différents : l'***interface*** et l'***implémentation***.

Le problème de l'interface consiste à définir l'allure que devrait avoir le SGF pour l'utilisateur. Cette tâche implique la définition d'un fichier et de ses attributs, des opérations, autorisées sur un fichier et de la structure de répertoires organisant les fichiers.

Le problème d'implémentation consiste à créer les algorithmes et les structures de données pour établir la correspondance entre le système logique de fichiers et les dispositifs physiques de mémoire auxiliaire.

Le SGF lui-même est généralement composé de plusieurs niveaux différents. La figure suivante donne un exemple d'architecture de SGF :

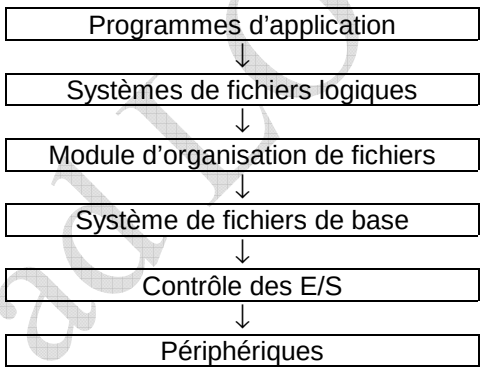


Fig. 7.1 SGF en couches

Le niveau inférieur, le contrôle des E/S, est constitué de *drivers* et des handlers (routines d'interruption) pour transférer l'information entre la mémoire et le système de disques. On peut considérer le driver comme un traducteur : ses entrées consistent en des commandes de haut niveau comme « récupérer le bloc 123 ». ses sorties sont des instructions de bas niveau, spécifiques au matériel, utilisés par le contrôleur du matériel qui relie le périphérique d'E/S au reste du système. Le driver écrit généralement des configurations binaires spécifiques dans des emplacements spéciaux de la mémoire du contrôleur d'E/S afin de lui indiquer sur quel emplacement du périphérique agir et quelles actions entreprendre.

Le *système de fichiers de base* doit seulement émettre des commandes génériques pour le driver approprié afin de lire et d'écrire les blocs physiques sur le disque.

Le *module d'organisation de fichiers* connaît les fichiers et leurs blocs logiques, ainsi que les blocs physiques. En connaissant le type d'allocation de fichiers employé et l'emplacement du fichier, ce module peut traduire les adresses des blocs logiques dans les adresses des blocs physiques pour que le système de transfert de base les transfère. Les blocs logiques du fichier sont numérotés de 0 à N, tandis que les blocs physiques contenant ces données ne correspondent généralement pas aux

numéros logiques, il faut donc une traduction pour localiser chaque bloc. Le module d'organisation de fichiers comprend également le gestionnaire de l'espace libre, qui suit la piste des blocs disponibles et les fournit au module d'organisation de fichiers quand celui-ci les demande.

Enfin le *système de fichier logique* utilise la structure de répertoires pour proposer au module d'organisation de fichiers l'information dont ce dernier a besoin, pour un nom donné de fichier logique est également responsable de la protection et de la sécurité.

Pour créer un nouveau fichier, un programme d'application appelle le SGF. Ce dernier connaît le format de la structure des répertoires. Il lit le répertoire approprié dans la mémoire, l'actualise avec la nouvelle entrée et le réecrit sur disque. Une fois que le répertoire a été actualisé, le SGF peut se l'utiliser pour exécuter les E/S.

La première référence à un fichier (normalement un open) provoque la recherche dans la structure des répertoires et la copie de l'entrée du répertoire pour ce fichier dans la table des fichiers ouverts. On retourne au programme utilisateur l'indice pour cette table et toutes les autres références s'effectuent par l'intermédiaire de l'indice (un descripteur de fichier, ou bloc de contrôle de fichiers). Par conséquent tant que le fichier reste ouvert, toutes les consultations du répertoire sont effectuées sur la table des fichiers ouverts. Toutes les modifications de l'entrée du répertoire sont réalisées sur la table en mémoire. Quand tous les utilisateurs employant un fichier le ferment, l'entrée actualisée est copiée sur la structure de répertoire du disque.

6.2 PROTECTION :

Quand on maintient de l'information dans un système informatique, l'un des principaux problèmes est sa protection contre les dégâts (fiabilité) et les accès non autorisés (protection).

On peut fournir de la protection de plusieurs manières différentes : protection par type, protection par groupe d'accès, … etc.

6.2.1 Protection par type :

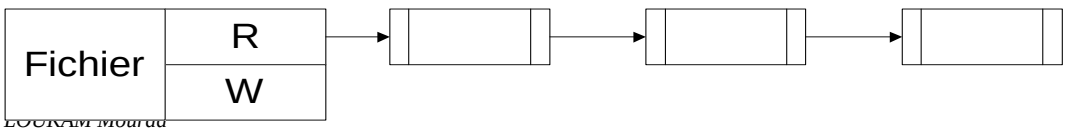
Le besoin de protéger des fichiers est une conséquence directe de la possibilité d'y accéder. Dans les systèmes dont l'accès aux fichiers des autres utilisateurs n'est pas permis toujours, on fournit un accès contrôlé.

Les mécanismes de protection fournissent un accès contrôlé en limitant les types d'accès possibles aux fichiers. On autorise ou on refuse un accès selon plusieurs facteurs, l'un d'eux est le type d'accès demandé. On peut contrôler différents types d'opérations : Lecture, Ecriture, Exécution, Ajout, Destruction, Enumération.

6.2.2 Protection par groupe d'accès :

L'approche la plus commune au problème de la protection consiste à rendre l'accès dépendant de l'identité de l'utilisateur. Divers utilisateurs peuvent avoir besoin de types d'accès différents à un fichier ou à un répertoire. Le schéma le plus général pour implémenter l'accès dépendant de l'identité consiste à associer à chaque fichier et répertoire une **liste d'accès**, en spécifiant le nom de l'utilisateur et les types d'accès autorisés à chaque utilisateur. Quand un utilisateur demande à accéder à un fichier particulier, le SE examine la liste d'accès associée à ce fichier. Si cet utilisateur se trouve dans la liste, l'accès est autorisé, sinon il se produit une violation de la protection.

Le principal problème concernant les listes d'accès est leur longueur. Si on désire autoriser tout le monte à lire un fichier, on doit construire une liste de tous les utilisateurs ayant le droit de le lire ; ce qui peut constituer une tâche pénible pour le système.



Afin de réduire la longueur de la liste d'accès, plusieurs systèmes reconnaissant trois types d'utilisateurs en liaison avec chaque fichier :

- **Propriétaire** : L'utilisateur qui a créé le fichier en est le propriétaire.
- **Groupe** : Le groupe est un ensemble d'utilisateurs partageant le fichier et ayant des besoins d'accès semblables.
- **Univers** : Tous les autres utilisateurs du système constituent l'univers.

La protection par groupe ne peut fonctionner correctement que si l'appartenance au groupe est contrôlée. Dans le système Unix et Windows NT, les groupes ne peuvent être chargés ou créés que par l'administrateur du système. Avec cette classification, il faut seulement 3 bits pour définir la protection ; chacun d'eux autorise ou interdit l'accès. Par exemple, le système Unix définit 3 champs **rwx** pour respectivement : la lecture, l'écriture, et l'exécution ou maintient un champ séparé pour le propriétaire, le groupe et les autres.

Exemple :

	Propriétaire	Groupe	Les autres
Student.doc	rwx	rwx	rwx
	Propriétaire	Groupe	Les autres
Programme.c	rw-	r--	r--

6.7 METHODES D'ALLOCATION :

L'accès direct aux disques permet une grande souplesse dans l'implémentation des fichiers. Le problème principal consiste à savoir comment allouer de l'espace à ces fichiers.

7.7.1 Allocation contiguë :

La méthode de l'allocation contiguë demande que chaque fichier occupe un ensemble de blocs contigus sur le disque. Les adresses disques définissent un ordre linéaire sur le disque.

L'allocation contiguë d'un fichier est définie par l'adresse disque et la longueur , en unités blocs, du premier bloc. Si le fichier est d'une longueur de n blocs et démarre à l'emplacement b, il occupe donc les blocs b, b+1, b+2, …, b+n-1. L'entrée du répertoire pour chaque fichier indique l'adresse du bloc de début et la longueur de la zone allouée au fichier. Exemple :

Répertoire

Fichier	Début	Longueur
Count	0	2
Tr	14	3
Mail	19	6
List	28	4
F	6	2

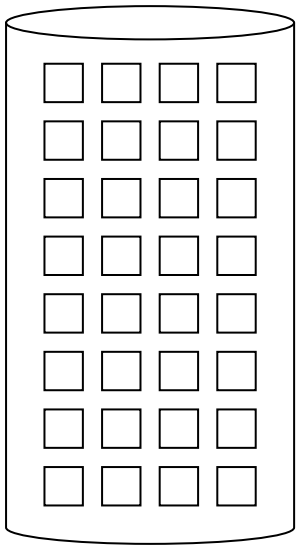


***Fig. 7.4**. Allocation contiguë*

Avantage : Déplacement minimal du bras du disque.
Inconvénient : Fragmentation.

6.7.2 Allocation chaînée :

L'allocation chaînée résout les problèmes de l'allocation contiguë. Avec l'allocation chaînée, chaque fichier est une liste chaînée de blocs de disques. Les blocs de disque peuvent être dispersés n'importe où sur le disque. Le répertoire contient un pointeur sur le premier et le dernier bloc du fichier.

Exemple : Le fichier abc occupe cinq (05) blocs : b9, b16, b1, b10, b25.

Répertoire

Fichier	Début	fin
Abc	9	25

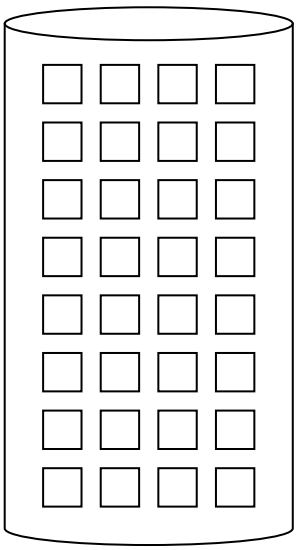


***Fig. 7.5**. Allocation chaînée*

Avantage : réduire la fragmentation.

Inconvénient : Le mouvement du bras du disque peut être important.

Il existe une autre variante importante de la méthode d'allocation chaînée : employer une table d'allocation de fichiers (File Allocation Table : FAT) qui a été utilisée dans les systèmes d'exploitation DOS et OS2. Celle-ci possède une entrée pour chaque bloc disque et elle est indexée par numéro de bloc. On utilise la FAT comme s'il s'agissait d'une liste chaînée. L'entrée du répertoire contient le numéro de bloc du premier bloc du fichier. L'entrée de la table indexée par ce numéro de bloc possède donc le numéro de bloc suivant dans le fichier. Cette chaîne continue jusqu'au dernier bloc, possédant une valeur spéciale de fin de fichier comme l'entrée de la table.

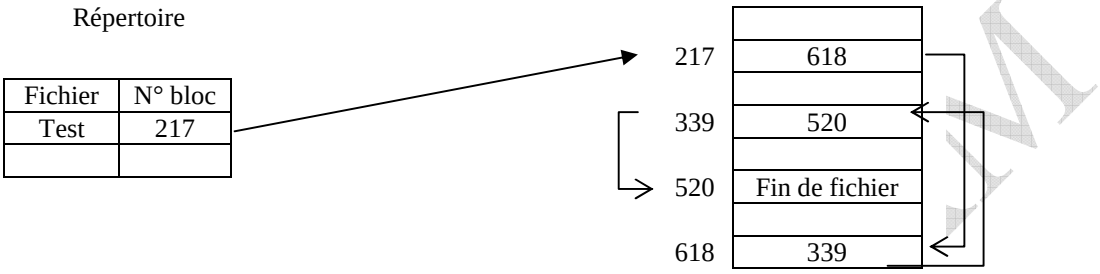


***Fig. 7.6**. Table d'allocation de fichiers*

6.8.3 Allocation indexée :

L'allocation chaînée résout les problèmes de fragmentation, cependant elle ne peut pas supporter l'accès direct de façon efficace, car les pointeurs vers les blocs ainsi que les blocs eux-mêmes sont dispersés dans tout le disque et ils doivent être récupérés dans l'ordre. L'allocation indexée résout ce problème en rangeant tous les pointeurs dans un seul emplacement : le bloc d'index.

Chaque fichier possède son propre bloc index, qui est un tableau d'adresse de blocs disque. La ième entrée dans le bloc index pointe sur le ième bloc du fichier. Le répertoire contient l'adresse du bloc index. Pour lire le ième bloc, on utilise le pointeur de la ième entrée du bloc index afin de trouver et de lire le bloc désiré. Ce schéma est semblable à celui de la pagination.

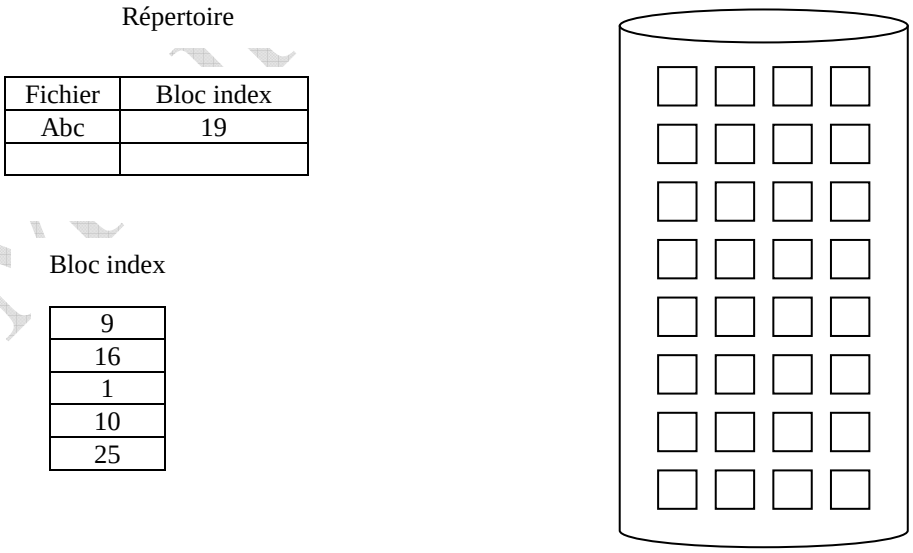


***Fig. 7.7**. Allocation indexée de l'espace disque.*

Quand on crée le fichier, tous les pointeurs du bloc index sont fixés à nul. Quand le ième bloc est écrit pour la première fois, on obtient un bloc du gestionnaire de l'espace libre et son adresse est placée dans la ième entrée du bloc index.

Inconvénient : L'espace occupé par le bloc d'index.

Chaque fichier doit posséder un bloc d'index dont il est souhaitable qu'il soit le plus petit possible. Cependant s'il est trop petit, il ne pourra pas ranger des pointeurs en nombre suffisant pour un grand fichier et il faudrait un mécanisme pour traiter ce détail.

Pour cela plusieurs solutions ont été proposées, dont :

Schéma chaîné : Pour des fichiers importants, on peut chaîner les blocs d'index entre eux.
Index multiniveaux : Cette solution consiste à utiliser un bloc d'index séparé pointant sur des blocs d'index.

6.9 GESTION DE L'ESPACE LIBRE :

Comme il n'existe qu'une quantité limitée d'espace disque, il est nécessaire de réutiliser l'espace des fichiers détruits pour les nouveaux fichiers. Pour cela, le système doit maintenir une liste d'espace libre. Cette liste mémorise tous les blocs libres du disque.

Pour créer un fichier, on recherche dans la liste d'espace libre la quantité requise d'espace et on alloue cet espace au nouveau fichier. Cet espace est ensuite supprimé de la liste. Quand un fichier est détruit, son espace disque est ajouté à la liste d'espace libre. Mais comment implémenter la liste d'espace libre ?. Plusieurs méthodes existent dont : le vecteur binaire et la liste chaînée.

6.9.1 Vecteur binaire :

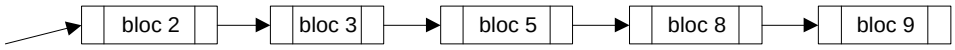
On implémente souvent la liste d'espace libre comme un tableau binaire. Chaque bloc est représenté par un bit. Si le bloc est libre, le bit est à 1, s'il est alloué le bit est à 0. Par exemple, si dans un disque les blocs 2, 3, 5, 8 et 9 sont libres, et les autres alloués, le vecteur représentant l'espace libre est alors :

N°bloc	0	1	2	3	4	5	6	7	8	9	…
Bit	0	0	1	1	0	1	0	0	1	1	

Avantage : Il est relativement facile de trouver le premier bloc libre ou les n blocs libres consécutifs.
Inconvénient : la gestion du vecteur.

6.9.2 Liste chaînée :

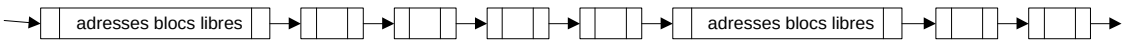
Il existe une autre approche pour représenter l'espace libre. Elle consiste à chaîner les blocs disques libres, en maintenant un pointeur sur le premier bloc libre dans un emplacement spécial du disque et en le mettant en mémoire cache. Exemple :



Avantage : La liste ne représente que les blocs libres.
Inconvénient : Parcours de la liste.

6.9.3 Liste chaînée avec groupement :

Il existe une autre variante de la méthode de la liste chaînée : on stocke les adresses des n blocs libres dans le premier bloc libre. Les n-1 premiers blocs sont réellement libres. Le dernier blocs contient les adresses de n autres blocs libres et ainsi de suite. L'importance de cette implémentation, c'est que l'on peut rapidement trouver les adresses d'un grands nombres de blocs libres, à la différence de la méthode chaînée standard.



6.9.4 Liste avec comptage :

La méthode de représentation d'espace libre avec comptage, profite du fait qu'il existe souvent plusieurs blocs libres contigus dispersés dans le disque. Alors, plutôt que de maintenir une liste de n adresses libres, on mémorise l'adresse du premier bloc libre et le nombre x de blocs contigus libres qui suivent le premier bloc. Chaque entrée dans la liste d'espace libre consiste donc en une adresse disque et un compteur.