CHAPITRE I : SYSTEME DE GESTION DE FICHIERS

1.1 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 :

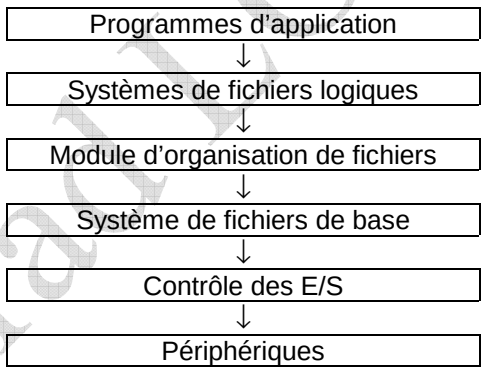


Fig. 1.1 SGF en couches

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.

1.2 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.

1.2.1 Protection par type :

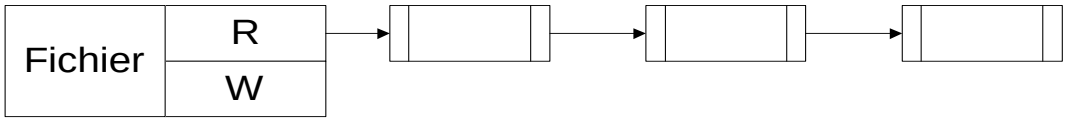
Le besoin de protéger des fichiers est une conséquence directe de la possibilité d'y accéder. Dans les systèmes dont l'accès aux fichiers des autres utilisateurs n'est pas permis toujours, on fournit un accès contrôlé.

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.

1.2.2 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--

1.3 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.

1.3.1 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

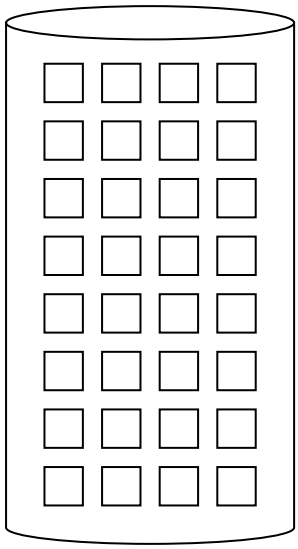


***Fig. 7.4**. Allocation contiguë*

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

1.3.2 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

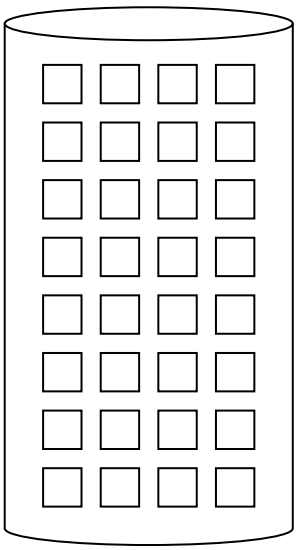


***Fig. 7.5**. Allocation chaînée*

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.

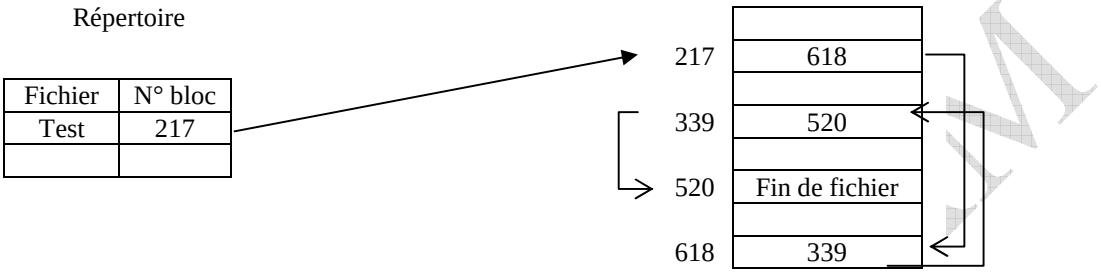


Fig. 7.1. *Table d'allocation de fichiers*

1.3.3 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.

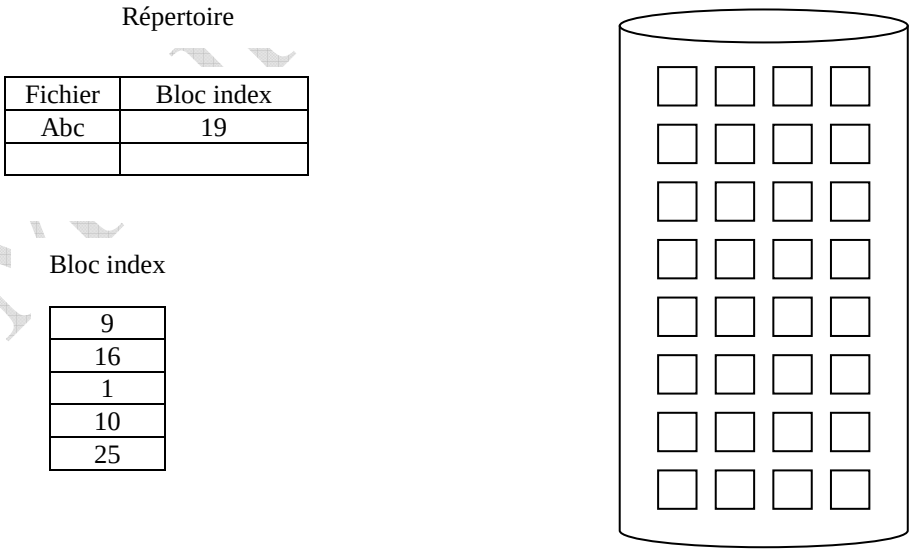


Fig. 7.7*. Allocation indexée de l'espace disque.*

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.

1.4 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.

1.4.1 Vecteur binaire :

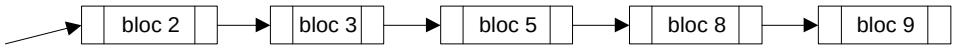
On implémente souvent la liste d'espace libre comme un tableau binaire. Chaque bloc est représenté par un bit. Si le bloc est libre, le bit est à 1, s'il est alloué le bit est à 0. Par exemple, si dans un disque les blocs 2, 3, 5, 8 et 9 sont libres, et les autres alloués, le vecteur représentant l'espace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

1.4.2 Liste chaînée :

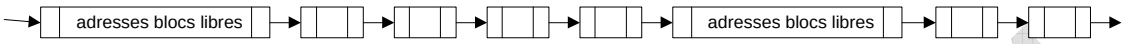
Il existe une autre approche pour représenter l'espace libre. Elle consiste à chaîner les blocs disques libres, en maintenant un pointeur sur le premier bloc libre dans un emplacement spécial du disque et en le mettant en mémoire cache. Exemple :



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

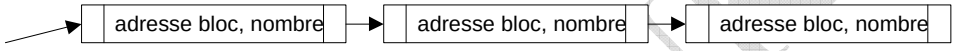
1.4.3 Liste chaînée avec groupement :

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.



1.4.4 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.



1.5 EXEMPLE DE SGF : UNIX

Unix possède de nombreuses variantes. La plupart des fournisseurs de systèmes se rattachent soit à Système V d'origine ATT soit à Berkeley (BSD). Les différences sont peu visibles pour les utilisateurs mais les systèmes de fichiers sont incompatibles. Une normalisation, elle-même incompatible avec les deux premiers systèmes, a été tentée sous l'égide de l'OSF (Open System Fundation).

1.5.1 Eléments d'un système de fichiers

L'espace disque est attribué par blocs de 512 à 4096 octets suivant les systèmes. La structure est composée de trois entités: le superbloc, le fichier des inodes et les fichiers de données : fichiers réguliers et répertoires.

Un système de fichiers est un disque virtuel créé par le responsable du système. Un disque peut être partitionné en plusieurs systèmes. Chez certains constructeurs un système de fichiers peut être réparti sur plusieurs disques. L'utilisateur ne voit que ces disques virtuels. Le superbloc contient des informations sur l'espace utilisé dans le disque virtuel, la liste des inodes des informations sur les fichiers réguliers. Le nom interne d'un fichier est un nombre entier appelé inode.

Pour le système les fichiers sont organisés en deux grandes familles :

1/les **fichiers standards** que sont par exemple les fichiers texte, les exécutables, etc. C'est-`a-dire tout ce qui est manipulé et structuré par les utilisateurs.

2/ Les **fichiers spéciaux** périphériques, mémoire, et autre fichiers "physiques" ou logique. Parmi ces fichiers , on trouve :

- Les répertoires
- Les fichiers physiques dans le répertoire /dev (dev comme devices dispositifs matériels, les périphériques et quelques dispositifs logiques) :

- Character Devices (où ou la communication ce fait octets par octets), comme les terminaux (claviers, écrans), les imprimantes, la mémoire.
- Block devices (périphériques ou la communication ce fait par groupe d'octet appel´es blocs) : les disques, les bandes magnétiques, etc.

- Les fichiers à usages logiques et non physiques
  liens symboliques
  pseudo-terminaux
  sockets
  tubes nomm´es

1.5.2 Le superbloc

Le superbloc décrit l'état d'occupation des secteurs du disque virtuel, alloués au système de fichiers. Lorsqu'on cherche à écrire sur un disque, il est en effet indispensable de connaître la liste des emplacements libres et occupés. Il faut donc construire et tenir à jour une carte d'occupation des lieux. Les inodes sont les noms internes des fichiers reconnus sous la forme de nombres entiers. Le superbloc maintient également la liste des noms libres utilisables, indispensable à la création de nouveaux fichiers.

Parmi les informations les plus importantes que contient le superbloc, on retiendra :

s_isize: la taille en blocs de la liste des inodes (i_list).
s_fsize: la taille en blocs du système de fichiers.
s_fname: le nom externe du système de fichiers.

Remarquons que ces informations déterminent le nombre maximum de fichiers que l'on peut créer et la taille du système de fichiers.
s_free: la liste des blocs libres.
s_inode: la liste des inodes libres
s_tfree: le nombre de blocs libres
s_tinode: le nombre d'inodes libres

Ces indications sont indispensables à l'allocation de l'espace et des fichiers. L'utilisateur peut la consulter au moyen de la commande df.
Le superbloc est chargé et verrouillé en mémoire au moment du démarrage ou lorsqu'un système de fichiers est monté (mount) et devient visible pour les utilisateurs. Il est constamment remis à jour et sauvegardé régulièrement sur disque ou volontairement par la commande sync.

1.5.3. La liste des inodes

La liste des inodes ou i_list est l'élément fondamental du système de fichier. Tout dommage à cette structure détruit irrémédiablement les liens donc les fichiers qui deviennent difficilement récupérables. Pour s'en convaincre il suffit de savoir qu'elle contient les renseignements suivants pour chaque inode :

i_uid, le numéro de l'utilisateur qui est son nom interne dans le système.
i_mode, les droits d'accès sur le fichier.
i_size, sa taille en blocs.
toutes les informations sur sa date de création, de modification et de dernier accès.
un pointeur permettant, comme nous allons le voir, d'accéder à son contenu.

On y trouve également des informations indiquant s'il s'agit d'un fichier, d'un répertoire ou de tout autre fichier spécial. L'utilisateur consulte son contenu, pour la part qui lui est accessible, au travers des protections, au moyen de la commande ls qui liste les informations sur les fichiers contenus dans un répertoire.

On utilise un adressage direct pour accéder au fichier s'il est court, de l'ordre de dix blocs, un adressage indirect pour les fichiers plus longs, comme schématisé dans la figure . La i_list comme le superbloc est chargée en mémoire et régulièrement remise à jour sur disque.

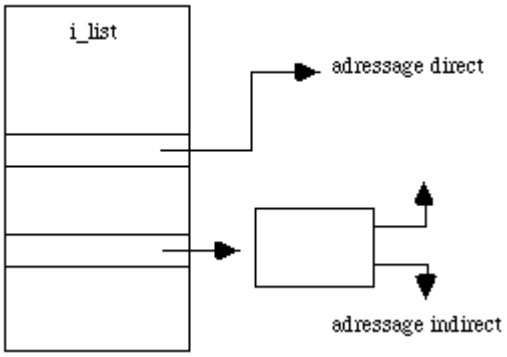


Figure - Structure d'accès à un fichier

1.5.4. Accès à un fichier

Imaginons qu'on recherche le fichier /usr/essai. Le processus pour y accéder est le suivant :

1. Lecture de la i_list pour retrouver la référence du répertoire / (racine).
2. Lecture de ce fichier répertoire. Les répertoires contiennent, entre autre, les noms interne et externe des fichiers et répertoires fils. usr est, par exemple, l'inode xxx
3. Lecture de la i_list pour retrouver les références de l'inode xxx.
4. Lecture des blocs correspondants à cet inode. On y trouve le nom interne yyy correspondant au nom externe essai.
5. Lecture de la i_list pour retrouver les références de l'inode yyy.
6. Accès au fichier yyy soit /usr/essai

Ce simple exemple montre la nécessité de préserver l'intégrité de la i_list. On notera également que l'accès serait extrêmement lent si celle-ci ainsi que le superbloc n'étaient pas chargés dans la mémoire de l'ordinateur. Ce n'était pas le cas des systèmes Multics connus par le passé et dont Unix est dérivé. Seul l'effondrement du prix des mémoires a permis à Unix de devenir un système efficace. Auparavant il souffrait des mêmes défauts que son prédécesseur et cette méthode d'accès était très lente.

1.5.5. Fragilité des systèmes de fichiers

L'information écrite sur un disque évolue rapidement. A un instant donné il n'est pas garanti que la géographie physique du disque corresponde exactement à la copie du superbloc et de la i_list qui existent sur celui-ci puisque seules leurs images dans la mémoire sont modifiées instantanément. La mise à jour est effectuée régulièrement par leur recopie sur le disque mais si le système est arrêté inopinément alors qu'il était en pleine activité, il n'est pas certain que la cohérence du système de fichiers soit préservée. C'est pourquoi il ne faut jamais arrêter brusquement une machine Unix: il faut utiliser la procédure shutdown prévue à cet effet. Entre autre chose celle-ci recopie les informations de la mémoire sur les disques.

Au démarrage de tout système une procédure analyse le système de fichiers pour vérifier la cohérence des informations inscrites dans le superbloc et la i_list. Tout fichier ou morceau de fichier dont le chemin d'accès ne peut être retrouvé est placé dans un répertoire spécial appelé lost+found! Cette vérification se déroule en cinq phases :

1. phase I : vérification de la liste des inodes
2. phase II : vérification des chemins d'accès
3. phase III: vérification de la connectivité des répertoires

4. phase IV : vérification des liens symboliques
5. phase V : vérification du superbloc

Ceci peut prendre un certain temps si la capacité en disques est élevée. Les procédures de démarrage les plus évoluées sont capables de s'apercevoir si un système a été précédemment arrêté proprement. Elles évitent alors cette étape fastidieuse.

Les constructeurs de systèmes informatiques sont conscients de la fragilité des systèmes de fichiers. Certains ont introduit une couche logicielle entre la représentation Unix des fichiers et la structure physique sur le disque (voir figure) qui réalise un certain nombre de fonctionnalités supplémentaires cachées. Par exemple la i_list comme le superbloc sont dupliqués un certain nombre de fois de façon à éviter leur perte. Il devient possible de reconstituer ces informations fondamentales à partir de leurs différentes images.

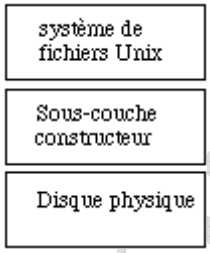


figure. Une démarche pour contourner la fragilité du système