qui incrémente le compteur de l'i-node du fichier (pour garder une trace du nombre de répertoires contenant le fichier) est parfois appelé **lien matériel** (*hard link*).

8. `Unlink`. Une entrée du répertoire est supprimée. Si le fichier qui est délié est uniquement présent dans un répertoire (le cas normal), il est effacé du système de fichiers. S'il est présent dans plusieurs répertoires, seul le nom du chemin d'accès spécifié est effacé. Les autres demeurent. Dans UNIX, `unlink` est l'appel système qui sert à effacer des fichiers (voir plus haut).

La liste ci-dessus présente les appels système les plus importants, mais il en existe d'autres, comme ceux qui gèrent les informations de protection associées aux répertoires.

Une variante du concept de lien est le lien symbolique (*symbolic link*). Au lieu d'avoir deux noms qui pointent sur la même structure interne de données symbolisée par le fichier, un nom est créé qui pointe sur un minuscule fichier contenant le nom d'un autre fichier. Quand le premier nom est utilisé, par exemple pour ouvrir le fichier, le système parcourt le chemin et trouve le nom à la fin. Il lance ensuite le procédé de recherche en utilisant le nouveau nom. Les liens symboliques présentent l'avantage de pouvoir traverser les frontières des disques physiques et même de voir désigner un fichier qui se trouve sur une autre machine du réseau. Leur implémentation est parfois moins efficace que pour les liens matériels.

4.3 Architecture d'un système de fichiers

À présent, il est temps de passer du point de vue de l'utilisateur à celui du concepteur. Les utilisateurs se préoccupent de la manière dont les fichiers sont nommés, des opérations autorisées, de l'arborescence des fichiers, etc. Les concepteurs, eux, sont plutôt intéressés par la façon dont les fichiers et les répertoires sont stockés, par l'organisation de l'espace du disque et par la manière de rendre le système de fichiers efficace et fiable. Nous allons étudier dans les sections qui suivent un certain nombre de ces aspects pour connaître les problèmes et les solutions utilisées.

4.3.1 L'organisation du système de fichiers

Les systèmes de fichiers sont enregistrés sur des disques. La plupart des disques peuvent être divisés en une ou plusieurs partitions, avec des systèmes de fichiers indépendants sur chaque partition. Le secteur 0 du disque, appelé **enregistrement d'amorçage maître** (*MBR = Master Boot Record*), sert à amorcer (*boot*) la machine. La fin du MBR comprend la table de partitions, laquelle indique l'adresse de début et de fin de chaque partition. Une de ces partitions est marquée comme étant la partition active. Quand l'ordinateur est amorcé, le BIOS lit et exécute le MBR. En premier lieu, le programme MBR détermine la partition active, y lit le premier bloc, appelé **bloc d'amorçage** (*boot block*), et l'exécute. Le programme du bloc d'amorçage charge le système d'exploitation contenu dans cette partition. Pour une question d'uniformité, chaque partition commence par un bloc d'amorçage, même si elle ne contient pas de système d'exploitation. Du reste, elle pourrait en contenir un ultérieurement.

Mis à part le démarrage avec un bloc d'amorçage, l'organisation d'une partition de disque varie fortement d'un système de fichiers à l'autre. Le système de fichiers contiendra souvent quelques-uns des items présentés à la figure 4.9. Le premier d'entre eux est le **superbloc** (*superblock*). Il contient tous les paramètres clés concernant le système de fichiers et est mis en mémoire quand l'ordinateur est amorcé ou quand le système de fichiers est modifié. Les informations caractéristiques du superbloc incluent un nombre magique qui identifie le type du système de fichiers, le nombre de blocs du système de fichiers et d'autres informations importantes relatives à l'administration.

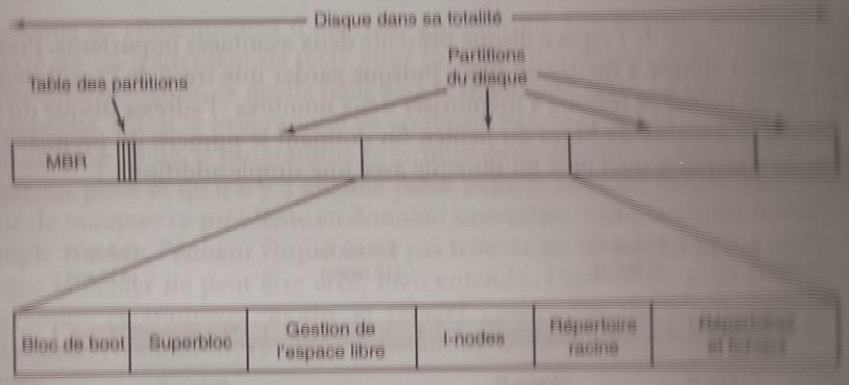

Figure 4.9 • Organisation possible d'un système de fichiers.

Viennent ensuite des informations sur les blocs libres du système de fichiers, par exemple sous la forme de tables de bits ou de listes chaînées. Elles sont suivies des i-nodes, représentés par un tableau, chaque élément du tableau correspondant à un fichier qui contient toutes les informations nécessaires sur le fichier. Enfin, on trouve le répertoire racine, qui contient le sommet de l'arborescence du système de fichiers et, pour terminer, le reste du disque qui renferme tous les répertoires et les fichiers.

4.3.2 Mise en œuvre des fichiers

L'aspect le plus important dans la mise en œuvre des fichiers est probablement la mémorisation des adresses de tous les blocs utilisés par chaque fichier. Différentes méthodes sont employées suivant les systèmes d'exploitation. Nous en examinerons quelques-unes dans cette section.

Allocation contiguë

La méthode d'allocation la plus simple consiste à stocker chaque fichier dans une suite de blocs consécutifs. Ainsi, dans un disque avec des blocs de 1 Ko, un fichier de

50 Ko se verra allouer 50 blocs consécutifs. Dans un disque avec des blocs de 2 Ko, ce fichier aura 25 blocs consécutifs.

Nous donnons un exemple d'allocation contiguë à la figure 4.10. Les 40 premiers blocs du disque y sont indiqués, démarrant avec le bloc 0 sur la gauche. Initialement, le disque est vide. Ensuite, un fichier *A*, d'une longueur de 4 blocs, est écrit sur le disque à partir du début (bloc 0). Puis c'est un fichier *B*, d'une longueur de 3 blocs, qui est écrit juste après la fin du fichier *A*.

Notons que chaque fichier commence au début d'un nouveau bloc, de sorte que si le fichier *A* avait en fait une longueur de 3,5 blocs, on perdrait un peu d'espace à la fin du dernier bloc. La figure présente 7 fichiers, qui débutent chacun dans le bloc suivant la fin du fichier précédent. Les nuances de gris ne sont là que pour mieux distinguer l'appartenance des blocs à un fichier.

L'allocation contiguë de l'espace disque présente deux avantages importants. Premièrement, elle est simple à mettre en place puisque garder une trace de l'emplacement des blocs d'un fichier se résume à mémoriser deux nombres : l'adresse disque du premier bloc et le nombre de blocs du fichier. En donnant le numéro du premier bloc, l'adresse de n'importe quel bloc est obtenue avec une simple addition.

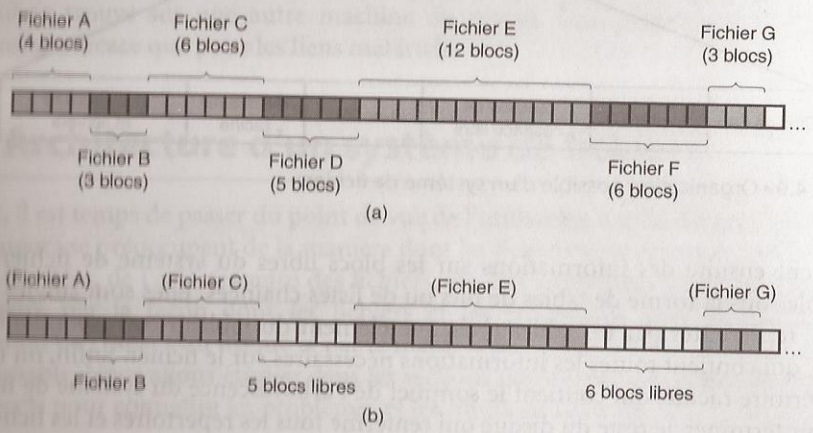

Figure 4.10 • (a) Allocation contiguë de l'espace disque pour 7 fichiers. (b) L'état du disque après la suppression des fichiers *D* et *F*.

Deuxièmement, les performances de lecture sont excellentes parce que le fichier peut être lu dans sa totalité en une seule opération. Seul un déplacement est nécessaire (au premier bloc). Après cela, plus aucun délai de déplacement ou de rotation n'intervient ; ainsi, les données arrivent avec le débit maximum du disque. Une allocation contiguë est donc simple à mettre en œuvre et offre de hautes performances.

Malheureusement, l'allocation contiguë a aussi deux inconvénients majeurs : avec le temps, le disque se fragmente. Pour comprendre ce qui se produit, voyons la figure 4.10(b). Ici, deux fichiers *D* et *F* ont été effacés. Quand un fichier est effacé, ses blocs sont libérés, laissant une série de blocs libres sur le disque. Le disque n'est pas compacté pour détruire les trous, parce qu'il faudrait alors copier tous les blocs suivant le trou, à savoir un nombre potentiel de millions de blocs. Par conséquent le disque consiste en une succession de fichiers et de trous, comme l'illustre la figure.

Au départ, la fragmentation n'est pas un problème, puisque chaque nouveau fichier peut être écrit à la fin du disque, à la suite des précédents. Toutefois, un jour ou l'autre, le disque sera plein et il deviendra nécessaire de le compacter – ce qui demande beaucoup de temps – ou de réutiliser l'espace libre des trous. La réutilisation de cet espace nécessite le maintien d'une liste de trous, ce qui est faisable. Néanmoins, quand un fichier est créé, il est nécessaire de connaître sa taille maximum afin de pouvoir choisir le trou de la bonne taille, à la bonne place.

Imaginons les conséquences d'une telle conception. L'utilisateur lance un éditeur de texte ou un traitement de texte pour saisir un document. Avant toute chose, le programme demande de quelle taille sera le document final. La question doit avoir une réponse, sinon le programme ne continue pas. Si la valeur donnée précédemment est trop petite, le programme se termine prématurément parce que le trou du disque est plein et qu'il n'y a aucune place pour le reste du fichier. Si l'utilisateur essaie de masquer ce problème en donnant une valeur complètement irréaliste, par exemple 100 Mo, l'éditeur risque de ne pas trouver un trou assez grand et d'annoncer que le fichier ne peut être créé. Bien entendu, l'utilisateur peut redémarrer le programme et annoncer 50 Mo, et répéter l'opération jusqu'à trouver un trou convenable. Néanmoins, ce type de fonctionnement ne peut pas satisfaire les utilisateurs.

Il existe toutefois une situation pour laquelle l'allocation contiguë est possible et, de fait, largement utilisée : sur les CD-ROM. En effet, la taille de tous les fichiers y est connue à l'avance et ne changera jamais lors d'utilisations ultérieures du système de fichiers du CD-ROM. Nous étudierons le plus commun des systèmes de fichiers de CD-ROM plus loin dans ce chapitre.

La situation est un peu plus compliquée avec les DVD. En principe, un film de 90 minutes peut être encodé dans un fichier unique de 4,5 Go, mais le système de fichiers utilisé appelé **UDF** (*Universal Disk Format*) utilise un nombre de 30 bits pour exprimer la longueur des fichiers, ce qui limite cette longueur à 1 Go. Par conséquent les films sont généralement rangés en trois ou quatre fichiers contigus de 1 Go. Les morceaux physiques d'un même fichier logique (le film) sont appelés des domaines (*extents*).

Comme nous l'avons vu au chapitre 1, l'histoire se répète souvent dans le monde informatique, à mesure que de nouvelles générations technologiques apparaissent. L'allocation contiguë était mise en place dans les systèmes de fichiers des disques magnétiques il y a quelques années, en raison de sa simplicité et de ses bonnes performances. Puis l'idée a été laissée de côté parce qu'il était fastidieux d'avoir à préciser la taille finale d'un fichier lors de sa création. Mais avec l'arrivée des CD-ROM, DVD et autres médias optiques à écriture unique, il est à nouveau intéressant de tenir compte

des fichiers contigus. Il est ainsi important d'étudier de vieux systèmes qui étaient conceptuellement clairs et simples, parce qu'on peut les appliquer de manière surprenante à de futurs systèmes.

Allocation par liste chaînée

La seconde méthode d'allocation des fichiers consiste à conserver chacun d'eux comme une liste chaînée de blocs de disque, comme illustré à la figure 4.11. Le premier mot de chaque bloc sert de pointeur sur le bloc suivant. Le reste du bloc contient les données du fichier.

Contrairement à l'allocation contiguë, chaque bloc du disque peut être utilisé dans cette méthode. Aucun espace n'est perdu dans une fragmentation du disque (excepté pour la fragmentation interne dans le dernier bloc). De plus, pour l'entrée du répertoire, il suffit de conserver l'adresse disque du premier bloc. Le reste peut être trouvé à partir de là.

D'un autre côté, bien que la lecture séquentielle d'un fichier soit simple, l'accès aléatoire est extrêmement lent. Pour accéder au bloc n, le système d'exploitation doit démarrer au début et lire les $n - 1$ blocs précédents, un par un. À l'évidence, autant de lectures provoqueront un fonctionnement terriblement lent.

Par ailleurs, la quantité de données stockées dans un bloc n'est pas en puissance de deux, parce que le pointeur prend quelques octets. Même si le fait d'avoir une taille particulière n'est pas spécialement néfaste, c'est en tout cas moins efficace puisque de nombreux programmes lisent et écrivent dans des blocs dont la taille est en puissance de deux. Lorsque les premiers octets de chaque bloc sont occupés par le pointeur sur le prochain bloc, la lecture d'un bloc complet nécessite l'acquisition et la concaténation de deux blocs du disque, d'où un temps supplémentaire pour la copie.

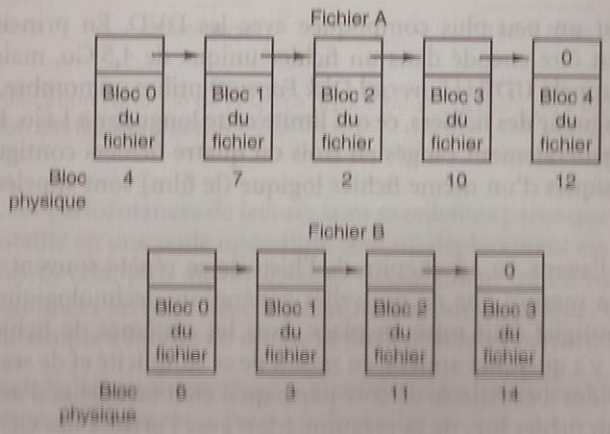

Figure 4.11 • Stockage d'un fichier à l'aide d'une liste chaînée de blocs de disque.

Allocation par liste chaînée utilisant une table en mémoire

Les deux inconvénients d'une liste d'allocation chaînée peuvent être éliminés en prenant le pointeur de chaque bloc du disque (le premier mot) pour le ranger dans une table en mémoire. La figure 4.12 montre à quoi ressemble la table de l'exemple 4.11. Dans les deux figures, nous avons deux fichiers. Le fichier *A* utilise les blocs 4, 7, 2, 10 et 12 dans cet ordre, et le fichier *B* utilise les blocs 6, 3, 11 et 14 dans cet ordre. En nous aidant de la figure 4.12, nous pouvons démarrer avec le bloc 4 et suivre la chaîne jusqu'à la fin. Nous pouvons procéder de même avec le bloc 6. Les deux chaînes se terminent avec un caractère spécial (−1 par exemple) qui n'est pas un numéro de bloc valide. Une telle table en mémoire principale est appelée FAT (*File Allocation Table*, table d'allocation des fichiers).

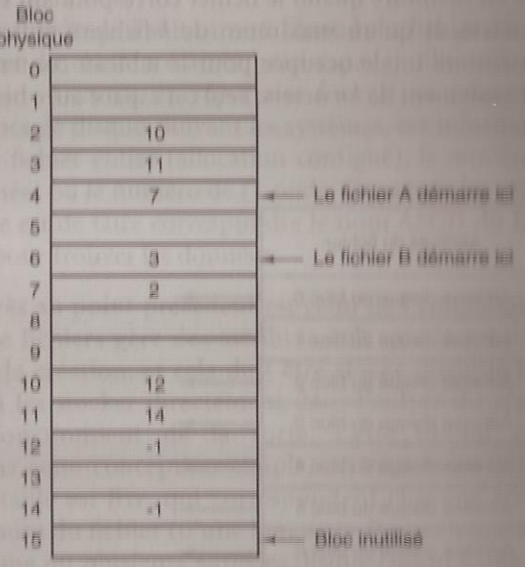

Figure 4.12 • Allocation par liste chaînée utilisant une table en mémoire principale.

Grâce à cette organisation, le bloc est disponible dans sa totalité pour les données. De plus, l'accès aléatoire est facilité. Comme la chaîne est intégralement en mémoire, bien qu'il faille encore la parcourir pour trouver un déplacement donné à l'intérieur du fichier, cela peut être réalisé sans faire de nouveaux accès disque. Comme dans la première méthode, il suffit, pour l'entrée du répertoire, de conserver un entier unique (le numéro du bloc de départ) ; il sera toujours possible de trouver tous les blocs, quelle que soit la taille du fichier.

Le principal inconvénient de cette méthode est que la totalité de la table doit se trouver tout le temps en mémoire. Avec un disque de 200 Go et une taille de bloc de 1 Ko, la table contient 200 millions d'entrées, c'est-à-dire une pour chacun des 200 millions de blocs. Chaque entrée doit avoir un minimum de 3 octets, et même de 4 octets