Chapitre 5 : Systèmes de gestion de fichiers

1. Introduction

2. Interface des systèmes de fichiers

2.1 Concept de fichier
2.2 Rôle du SGF
2.3 Vue utilisateur sur un fichier
2.4 Opérations sur les fichiers
2.5 Structure interne d'un fichier
2.6 Méthodes d'accès aux fichiers
- Accès séquentiel
- Accès direct

2.7 Les répertoires

3. Implémentation des fichiers

3.1 Organisation physique des fichiers
3.1.1 Implantation séquentielle
3.1.2 Implantation non contiguë
- Blocs chainés
- Allocation par liste chainée dans une table en mémoire
- Les i-nodes :
- Méthodes indexées

3.2 Gestion de l'espace libre d'un disque
3.2.1 Méthode de liste chainée
3.2.2 Méthode de table de bits

3.3 Gestion des fichiers actifs

1. Introduction

Le traitement de l'information nécessite souvent la conservation de l'information. Dans un système informatique, l'unité la plus petite de stockage des données sur supports externes quelque soient leurs natures, disque magnétique, bande magnétique ou disque optique ou flash disque ou autre, est un *fichier*. C'est un moyen qui permet au système d'exploitation de fournir une vue logique uniforme du stockage des informations. En définissant cette unité de stockage qui est le fichier, le système d'exploitation permet de faire une abstraction des propriétés physiques de ces supports de stockage qui sont de nature non volatiles. Par conséquent, il est nécessaire d'introduire des mécanismes qui permettent de concilier la vue logique des fichiers et les dispositifs de stockage. La partie du système d'exploitation qui permet de prendre en charge le stockage et l'accès aux fichiers est le système de gestion de fichiers (SGF). La gestion des fichiers implique aussi l'introduction des méthodes permettant la protection des fichiers lors des accès concurrents. Le SGF permet de réaliser ces opérations et les opérations liées au stockage des fichiers. C'est la partie la plus visible dans un système d'exploitation du point de vue des utilisateurs.

Un fichier est un ensemble d'informations, dépendantes entre elles ou non, fortement formatées d'une manière prédéfinie (enregistrements) ou non (fichier texte). Il représente, du point de vue de l'utilisateur, l'unité d'allocation logique pour le stockage de données sur une mémoire auxiliaire. Un fichier peut stocker de l'information sous forme alphabétique, numérique, alphanumérique ou binaire. Les informations d'un fichier peuvent être de plusieurs types : programme source, programme objet, programme exécutable, données (alphanumériques, texte), enregistrement, image, graphique, son etc. Tous ces formats se présentent sous l'un des deux aspects cités ci-dessus, il revient au programme traitant le fichier d'interpréter son contenu.

Vu le grand nombre de fichiers stockés et la multiplicité des utilisateurs les accédant, il est impératif d'organiser ces fichiers pour ces utilisateurs, d'où l'introduction des *répertoires*. Un répertoire est un moyen logique d'organisation des fichiers pour faciliter leur utilisation, il cloisonne un groupe de fichiers. Des répertoires peuvent aussi être inclus dans d'autres répertoires, ce qui définie des structures (arborescentes, acycliques etc …) de répertoires. La notion de répertoire permet de créer une indépendance entre les différents utilisateurs. Par exemple, deux utilisateurs différents peuvent créer des fichiers avec le même nom ! D'autres avantages sont, la facilitation et l'accélération de la recherche des fichiers. Les répertoires résident au même titre que les fichiers sur un support de stockage externe.

Il est devenu clair que la gestion de fichiers peut être à deux niveaux, le niveau utilisateur (logique) et le niveau machine (physique). Ce chapitre est dédié à la présentation de l'interface des systèmes de gestion de fichiers et l'implémentation des fichiers dans un système Informatique.

2. Interface des systèmes de fichiers

2.1 Concept de fichier

Un fichier est une collection d'informations reliées ou non qui sont enregistrées sur un support externe. Le fichier est une abstraction des propriétés physiques de ce support. Un fichier peut être structuré (fichier à enregistrement) ou non structuré (fichier texte). En général, il peut être une suite de bits, d'octets, de lignes de textes ou d'enregistrements. Il peut contenir des programme sources ou objets ou exécutables, des données numériques, des images etc. Un fichier est désigné par un ensemble d'attributs qui sont : le nom, l'emplacement de stockage, la taille, informations de protections, heures et dates de création

et de dernière modification, identité du propriétaire, les droits d'accès etc. Ces informations définissent le descripteur de fichier et sont stockées dans le répertoire où réside ce fichier.

2.2 Rôle du SGF

Rappelons qu'un système de fichiers possède deux rôles fondamentaux, il fournit des mécanismes de stockage et d'accès aux fichiers. Autrement dit, il définit deux interfaces l'une avec les utilisateurs quand il s'agit de manipuler les fichiers et l'autre avec le système d'exploitation quand il s'agit de leurs stockages. La première relève de l'aspect logique et la seconde du physique.

Un fichier est classé comme un type de donnée abstrait. Des opérations sont alors définies sur un fichier, elles sont entre autres la création, l'écriture, la lecture, le positionnement à l'intérieur du fichier, la destruction, la troncation d'un fichier. Se sont des opérations de base qu'on peut faire sur un fichier. D'autres opérations sont l'ajout à la fin du fichier, la copie de fichier, le changement de nom de fichier, la connaissance de l'état d'un fichier etc… Celles-ci peuvent être réalisées en combinant les opérations de base.

2.3 Vue utilisateur sur un fichier

L'utilisateur d'un système informatique organise l'information de ses fichiers suivant ses besoins en leur définissant une structure dite logique. La représentation des informations en mémoire secondaire (adresse d'implantation, codage de l'information, localisation, chainage des différentes parties du fichier et leur indexation éventuelle) définie leur organisation physique. La correspondance entre l'organisation logique et l'organisation physique est à la charge du SGF. L'accès à un fichier est réalisé à l'aide de fonctions, dites d'accès, exprimées à l'aide de la structure logique. Pour faciliter la présentation et sans perdre de généralité, on considère dans la suite du Chapitre un fichier structuré, celui-ci est vu comme étant une séquence contiguë d'informations élémentaires de même nature, appelées enregistrements (ou articles) qui sont numérotées de 0 à n-1.

2.4 Opérations sur les fichiers

L'utilisateur n'est pas directement concerné par la façon dont le système d'exploitation représente physiquement les fichiers ni de la manière d'accéder aux éléments de ceux là. Ce qui l'intéresse, c'est de pouvoir les définir, les utiliser et les conserver. Pour cela, le SGF produit un ensemble de primitives permettant la manipulation des fichiers dont principalement, on distingue :

- La création : Elle implique deux étapes : la première est le test de la disponibilité de l'espace et l'allocation de celui-ci au fichier. La seconde est la création d'une entrée (descripteur) dans le répertoire et son remplissage avec les informations liées au fichier.
- Destruction : Il s'agit de la libération de l'espace occupé par le fichier et celui de son descripteur.
- Ouverture : Cette opération déclare l'intention de l'utilisation du fichier. Elle consiste à ramener le descripteur du fichier en mémoire principale pour rendre le fichier actif.
- Fermeture. Elle rend le fichier inaccessible à l'utilisateur concerné. Elle peut engendrer la libération de l'entrée du descripteur de ce fichier en mémoire principale.
- Ecriture, lecture : Ces opérations permettent l'accès au fichier respectivement en écriture et en lecture.

D'autres fonctions existent, citons par exemple : le changement de nom du fichier, l'ajout en fin du fichier, copie de fichier etc.

2.5 Structure interne d'un fichier

Les fichiers sont conservés sur mémoire secondaire dans des blocs dont la taille est fixe, dont un bloc est l'unité de transfert entre la mémoire principale et la mémoire secondaire. Elle peut être de 256, 512 ou de 1024 octets ou plus, de manière générale en puissance de deux. Les enregistrements logiques sont regroupés dans ces blocs selon un nombre dépendant de leurs tailles. Ce nombre est appelé facteur de blocage. Par exemple, si la taille d'un enregistrement est de 64 octets et la taille d'un bloc est de 256, alors le facteur de blocage est 4. Les enregistrements logiques peuvent être de longueurs variables, donc le facteur de blocage devient aussi variable. Un fichier peut donc être vu comme étant une séquence continue de blocs logiques numérotées de 0 à n-1. Toutes les fonctions d'entrées/sorties sur le fichier travaillent sur des blocs. Un bloc physique est l'espace mémoire de la taille d'un bloc logique localisé à une adresse physique qui est l'adresse du début du bloc où est mémorisé un bloc logique. La localisation d'un bloc physique sur support externe dépend de l'organisation physique du fichier correspondant. La conversion des enregistrements logiques en blocs physiques est alors nécessaire. Cette opération peut se faire en passant par le bloc logique. La séquence de conversions (i.e de correspondances) est donc : enregistrement logique vers bloc logique vers bloc physique. L'exemple de la figure 6.1 illustre ce processus.

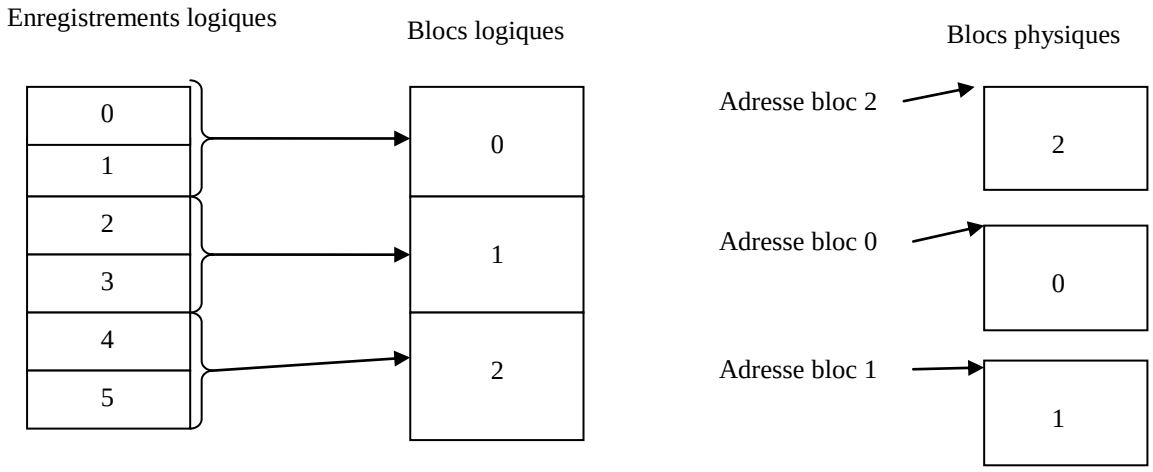
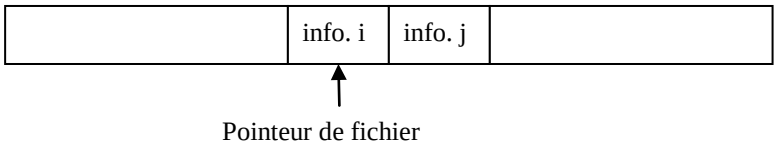


Figure 6.1 : *Correspondance entre enregistrement, bloc logique et bloc physique.*

2.6 Méthodes d'accès aux fichiers

Les primitives permettant d'accéder aux informations d'un fichier définissent la structure logique dont les utilisateurs disposent pour organiser les fichiers. Cette structure est appelée le mode d'accès au fichier. A chaque fichier ouvert est associé un pointeur logique du fichier (appelé pointeur courant) qui référencie l'enregistrement courant concerné par l'opération de lecture/écriture.

- **Accès séquentiel :** Dans ce mode, les numéros d'ordre ne peuvent pas être utilisés par les fonctions d'accès, qui permettent seulement :
    - de lire ou d'écrire le prochain enregistrement (ou l'enregistrement pointé), (voir Figure 6.2), ou
    - de se positionner au début ou à la fin du fichier.



Après exécution de *lire (art), art* est affecté de *info. i* et le pointeur est déplacé vers *info. j*

Figure 6.2 : *Effet d'une opération d'accès à un fichier.*

L'adresse logique de l'article *i* est donnée par l'adresse logique de l'article *i-1* + taille d'un article. Pour accéder à un élément de numéro *k*, il faut passer par tous les éléments de numéros *0* à k-1. L'accès séquentiel est un mode hérité de l'époque où l'implémentation physique des fichiers se faisait sur bande magnétique.

- **Accès direct :** Dans ce mode, les fonctions d'accès s'expriment en *fonction des attributs des articles* ; ces attributs sont généralement les valeurs des différents champs. Rappelons qu'un article est structuré en plusieurs champs dont chacun est d'un type de données qui peut être différent. On appelle *clé*, un champ d'un article dont la valeur peut servir pour désigner l'article (voir Figure 6.3). La clé peut être unique, dans ce cas, elle désigne sans ambiguïté un article. Elle peut être aussi multiple dans le cas contraire. Pour simplifier, on considère le cas où la clé est unique, donc deux articles différents possèdent deux clés différentes.

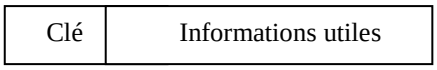


Figure 6.3 : *Structure d'un article.*

La recherche d'un article dans le fichier est la base de la plupart des fonctions élémentaires d'accès direct : lire, écrire, supprimer… Deux classes de méthodes sont utilisées : l'adressage dispersé et l'adressage indexé.

- L'adressage dispersé avec indexe : Dans cette méthode, on essaye de réaliser directement la fonction de recherche en définissant une fonction *f*, dite fonction de hachage (ou de dispersion), telle que *f(clé)* donne une entrée dans une table qui donnera le numéro du bloc contenant l'article dont la clé est le paramètre de cette fonction (voir figure 6.4). La fonction f appliquée à deux clés différentes peut donner le même numéro du bloc, cela est possible si le facteur de blocage est supérieur à 1.

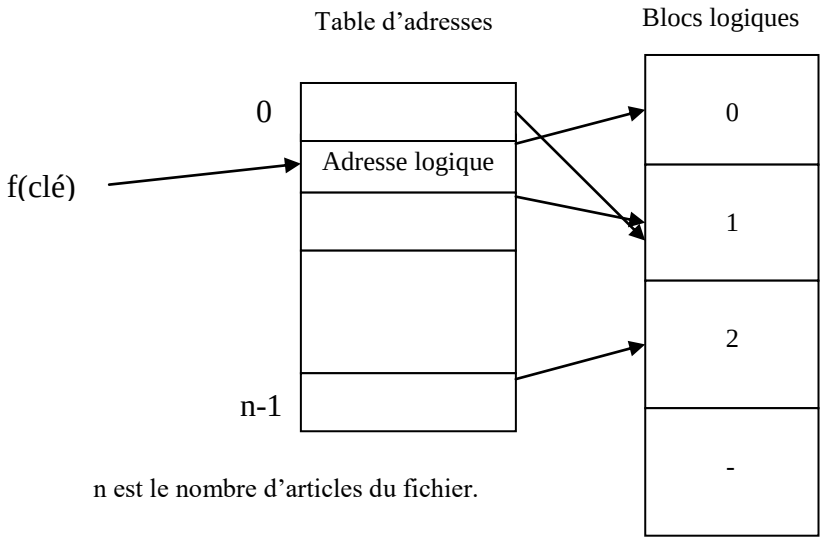


Figure 6.4 : *Adressage dispersé.*

Un exemple de fonction de hachage est donné comme suit : $f(cle_i) = \Sigma(codes\ de\ caractères\ de\ la\ clé_i) / n$. Une fonction de hachage doit respecter les règles suivantes :

- $\forall c$ *une clé,* $0<=f(c)<n$ et,
- $\forall c1, c2, f(c1) \neq f(c2)$, *si* $c1 \neq c2$.

Cette deuxième condition n'est pas toujours respectée, ce qui implique des collisions. Dans ce cas, les entrées donnant la même valeur sont chainées entre elles en utilisant une zone de débordement. Par conséquent, *n* est inférieur au nombre d'articles. La structure de la nouvelle table est donnée par la figure 6.5.

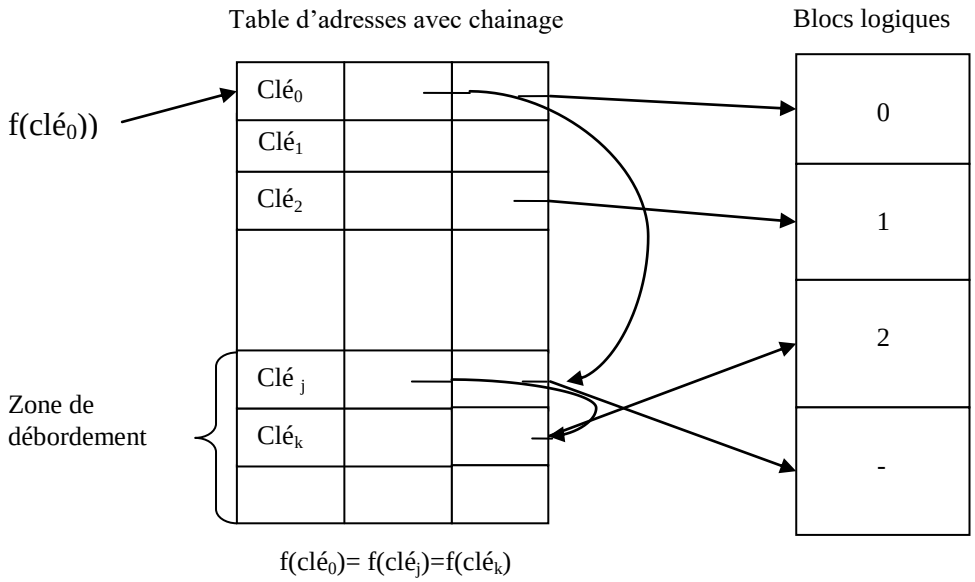


Figure 6.5 : *Adressage dispersé avec table de chainage.*

Pour rechercher une adresse logique, on applique la fonction de hachage à la clé donnée. On compare ensuite, la clé donnée aux différentes clés des entrées obtenues en

suivant le chainage à partir de l'entrée désignée par la fonction de hachage, jusqu'à ce qu'on trouve la clé recherchée. Le champ adresse de cette entrée donne alors le numéro du bloc correspondant à la clé.

Remarques

- La fonction de hachage n'assure pas la relation d'ordre des clés avec l'ordre des adresses logiques.
- La fonction de hachage peut être utilisée sans index. Dans ce cas, elle donne directement l'adresse logique du bloc correspondant à la clé.

o Index : L'accès direct peut être réalisé sans utilisation de la fonction de hachage. Dans ce cas, la table contient les clés triées de manière croissante ou décroissante de valeurs (voir figure 6.6). Pour rechercher l'entrée correspondant à une clé donnée et donc l'adresse du bloc logique correspondant, on emploie une recherche moins couteuse, par exemple la recherche dichotomique.

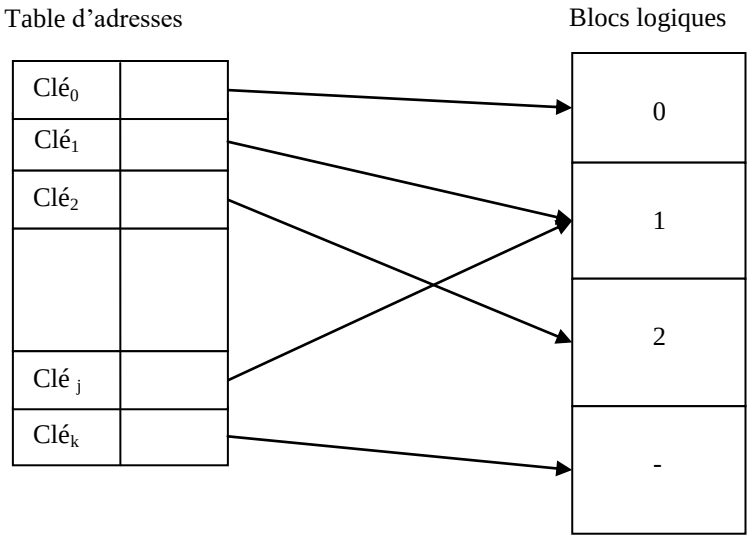


Figure 6.6 : *Table d'adressage sans hachage.*

2.7 Les répertoires

Pour organiser les fichiers, les systèmes de fichiers font recours aux répertoires. Logiquement, un répertoire peut être vu comme étant un ensemble qui regroupe un groupe de fichiers mais aussi de répertoire, appelés sous répertoires. Ainsi, on observe une structure hiérarchique de répertoires. Pour retrouver un fichier dans un répertoire donné, la notion de chemins d'accès est employée. Il s'agit de la suite de noms de répertoires partant de la racine (répertoire de départ) en passant par les répertoires intermédiaires dans la hiérarchie sur le chemin qui mène vers le répertoire contenant le fichier en question. Les opérations portantes sur un fichier, notamment sa création doit référencer le répertoire ciblé, donc le chemin d'accès vers ce fichier.

Les répertoires sont manipulés par différentes opérations qui sont entre autres, la création, la suppression, listing du contenu du répertoire, etc.

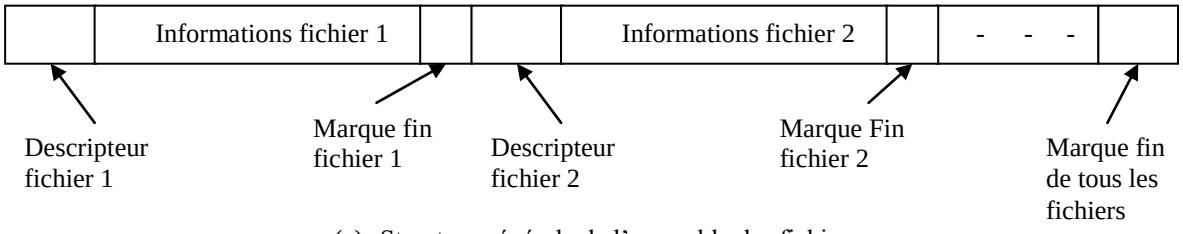
3. Implémentation des fichiers

3.1 Organisation physique des fichiers

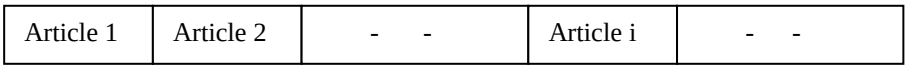
L'organisation physique d'un fichier définie la manière dont les informations de chaque fichier sont implantées en mémoire secondaire. Cette mémoire est supposée être un disque qui est organisé comme une séquence de blocs dits blocs physiques de tailles fixes et égales. On admet qu'un bloc est l'unité de transfert entre la mémoire auxiliaire et la mémoire principale. Deux classes de méthodes sont utilisées pour l'implantation physique des fichiers : l'implantation séquentielle (contiguë) et l'implantation non contiguë. Le problème à résoudre est comment retrouver l'adresse physique sur disque d'un bloc à partir de son numéro logique où à partir d'une clé d'enregistrement.

3.1.1 Implantation séquentielle

Dans cette organisation, chaque fichier occupe en mémoire secondaire un ensemble de blocs consécutifs. Elle est la seule possible sur une bande magnétique : Les fichiers sont rangés consécutivement sur la bande (voir figure 6.7).



(a) Structure générale de l'ensemble des fichiers



(b) Structure interne d'un fichier à accès séquentiel

Figure 6.7 : *Organisation physique séquentielle de fichiers.*

Cette organisation peut être utilisée aussi sur disque, elle permet un accès séquentiel efficace. Le calcul d'adresse d'un bloc est simple, il ne nécessite pas d'accès au disque. Etant donné un bloc logique *i* (qui commence à partir de *0*) à rechercher sur disque, son adresse est donnée comme suit :

$$Adresse\ disque = adresse\ de\ base + (i - 1) * taille$$

où *adresse de base* est l'adresse physique du premier bloc sur support externe et *taille* est la taille d'un bloc physique.

L'organisation séquentielle a l'avantage d'être simple d'implémentation et efficace en temps d'accès. En effet, il suffit de connaître l'adresse de début du fichier et sa taille pour le mettre sur support externe. Quant au coût d'accès au fichier, il est limité juste au chargement des blocs du fichier qui s'effectuent séquentiellement sans déplacement du bras du disque vers d'autres adresses.

En échange, cette organisation présente des inconvénients liés à la création, à la destruction et au changement de taille, fréquentes d'un fichier. Il s'agit des problèmes de fragmentations dites *externes* et des coûts liés aux décalages des fichiers. A la création des premiers fichiers, aucune fragmentation n'est à signaler. Par la suite, quand des fichiers sont détruits, la création de nouveaux est conditionnée par l'espace des fragments libérés. La figure 6.8 illustre ce phénomène. Dans cette figure, *5* fichiers sont stockés avec les tailles indiquées.

Supposons que le fichier *2* et le fichier *4* sont détruits, leurs espaces respectifs deviennent libres. Si une demande de création du fichier 6 avec une taille de *4* blocs arrive, celui-ci occupera l'espace libéré du fichier *4* et *1* bloc reste libre. Si une autre demande pour la création du fichier *7* avec une taille de *4* blocs est demandée, aucun espace contiguë ne peut le contenir bien que l'espace total libre, qui est de *4* blocs, le permet. Une opération de décalage des fichiers existants (dite aussi de défragmentation) après les premiers blocs libres est alors nécessaire pour le contenir, ce qui est couteux.

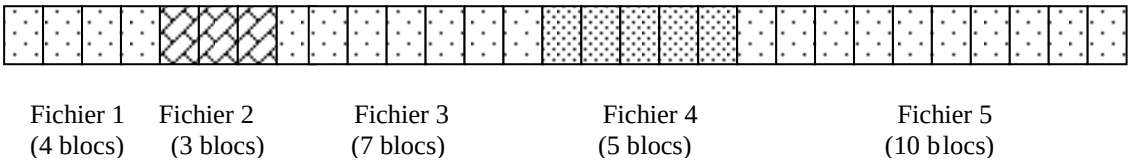


Figure 6.8 : *Problème de fragmentation de fichiers.*

3.1.2 Implantation non contiguë

Dans ce cas, les blocs physiques d'un fichier sont dispersés sur le support externe de manière non forcément contiguë. Différentes solutions d'organisations existent et chacune possède sa propre méthode qui permet de retrouver chaque bloc du fichier.

- **Blocs chainés**

Dans cette organisation (voir figure 6.9), les blocs d'un fichier sont dispersés sur le support externe de manière non contiguë selon l'espace disponible alloué par le système de fichiers. Les blocs sont chainés entre eux, le bloc de numéro *i* contient l'adresse physique du bloc *i+1*. Pour retrouver l'adresse du fichier, on utilise un descripteur qui pointe sur l'adresse du premier et du dernier, blocs du fichier.

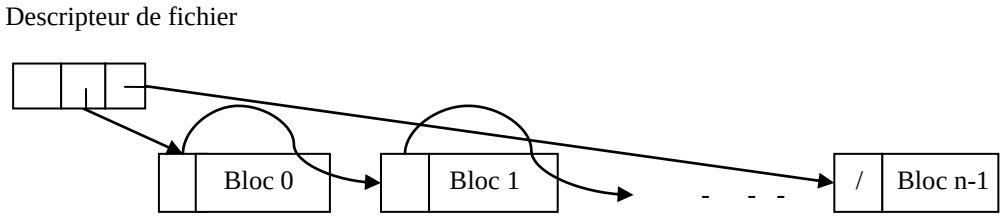


Figure 6.9 : *Organisation physique séquentielle de fichiers.*

Cette organisation s'adapte bien à l'accès séquentiel. En effet, pour retrouver le bloc de numéro *i*, il faut accéder d'abord à tous les blocs du début jusqu'au bloc *i-1*. Ce qui implique *i* accès au support externe. Aussi, le problème de la fragmentation externe est éliminé complètement, mais la fragmentation dite *interne* est toujours d'actualité si la taille du fichier n'implique pas l'occupation complète du dernier bloc physique de ce fichier. L'inconvénient de cette méthode est que la taille du bloc de données est diminuée à cause de l'espace pris par l'adresse, la taille du bloc n'est plus une puissance de deux. Si la taille en puissance de deux est requise par un programme, la lecture d'un bloc entier nécessite l'accès à deux blocs.

- **Allocation par liste chainée dans une table en mémoire**

Pour éliminer ce dernier problème, on emploie une table en mémoire pour contenir les chainages nécessaires aux différents blocs de chaque fichier. Dans cette table, appelée FAT

(*File Allocation Table*), le numéro de chaque entrée indique le numéro du bloc physique du fichier (s'il est alloué) et le contenu de cette entrée indique le numéro du prochain bloc physique s'il y a lieu sinon une marque de fin des blocs du fichier. L'adresse réelle est obtenue par simple calcul : <*numéro bloc*> . <*taille d'un bloc*>. La figure 6.10 donne un exemple d'implantation de deux fichiers, le premier est de taille 3 blocs physiques numérotés : 1, 5 et 2 ; le second est de taille 4 blocs physiques numérotés : 3, 7, 4 et 8. Les blocs de numéros : 0, 6 et 9 sont libres.

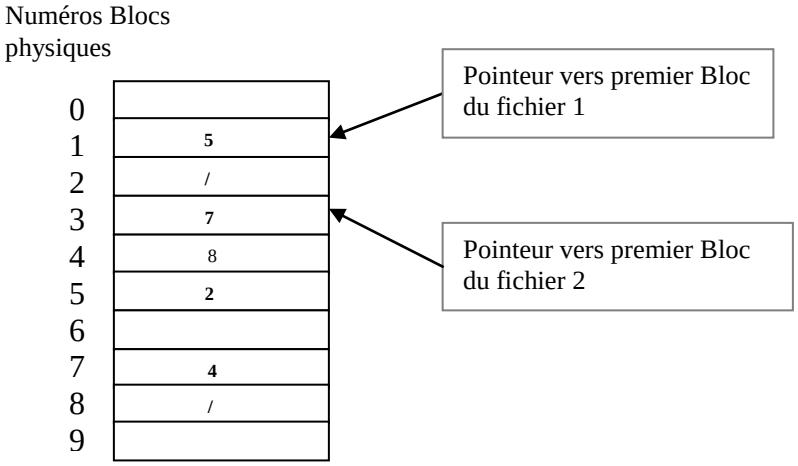


Figure 6.10 : *Utilisation d'une table en mémoire pour l'allocation par liste chainée.*

Cette organisation permet d'une part de réserver le bloc en entier pour les données et d'autres part, de limiter le nombre d'accès au disque à un seul si on admet que cette table est chargée en entier en mémoire principale. En effet, le parcours de la table pour rechercher un bloc donné ne nécessite pas de nouveaux accès au disque, donc l'accès à un bloc est presque similaire à un accès direct. Le seul inconvénient est que la table doit être en entier en mémoire principale, ce qui n'est pas très pratique pour des disques trop volumineux.

- **Les i-nodes :**

Une autre solution permettant d'éviter l'inconvénient de la table en mémoire pour l'allocation chainée est de regrouper ensemble les adresses des blocs d'un fichier avec les autres attributs du fichier dans le descripteur du fichier nommé pour la circonstance *nœud index* ou *i-node*. L'apport de cette structuration est que le chargement de l'i-node n'est requis que si le fichier est ouvert. Par conséquent, la taille des indexes des fichiers requis est limitée à celle des i-nodes des fichiers ouverts. Selon la structure de l'i-node, on peut retrouver l'ensemble des blocs du fichier. La figure 6.11 donne un exemple simple de la structure d'un i-node. La dernière entrée de l'i-node est utilisée pour pointer un bloc d'adresse disques si la taille du fichier est grande. On peut même imaginer une structure arborescente de blocs d'adresses si la taille du fichier est encore plus grande.

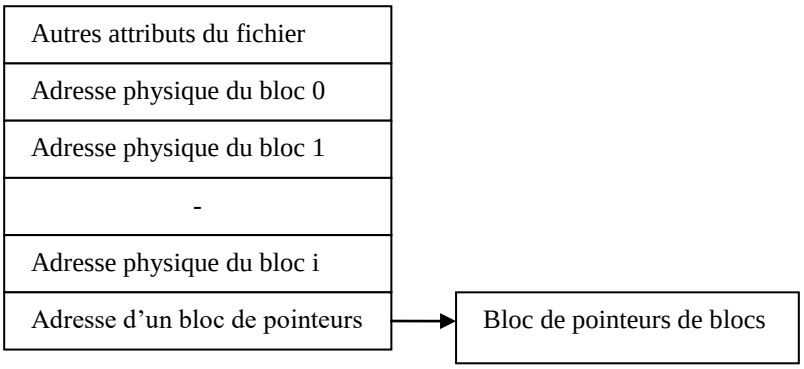
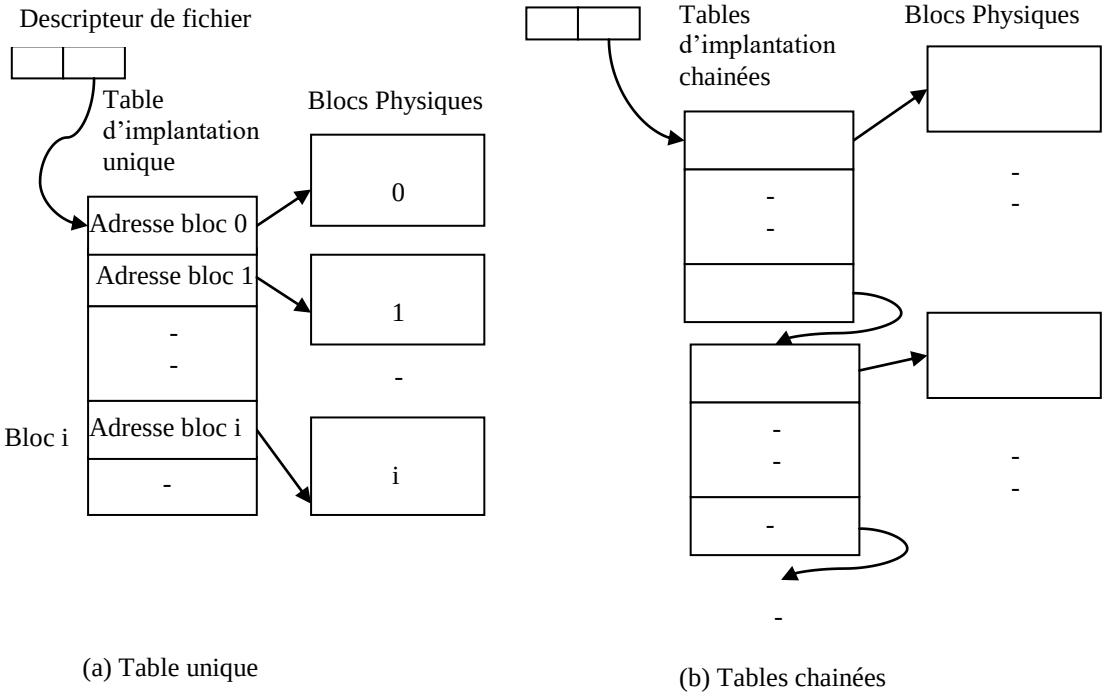


Figure 6.11 : *Structure simple d'un i-node.*

- **Méthodes indexées**

Pour réaliser des méthodes d'accès efficaces, il faut réduire le nombre d'accès au support externe pour retrouver un bloc. La meilleure méthode est celle qui donne un temps d'accès à un bloc indépendamment de son adresse. Dans ce sens, on emploie les méthodes indexées. On distingue deux types d'indexations, à un niveau et à plusieurs niveaux.

Table d'implantation à un niveau : Pour réaliser cette méthode, on regroupe les adresses des blocs physiques dans une table d'implantation unique. L'entrée *i* de la table donne l'adresse physique du bloc logique de numéro *i* (voir figure 6.12(a)). Dans le cas où le fichier est de grande taille, on peut employer plusieurs tables chainées entre elles (voir figure 6.12(b)).



(a) Table unique

(b) Tables chainées

Figure 6.12 : Table d'indexation à un niveau.

Table d'implantation à plusieurs niveaux : Ce type d'indexation emploie différents niveaux d'indexation pour retrouver l'adresse d'un bloc physique à partir d'une clé donnée. La Figure 6.13 donne un exemple de tables à trois niveaux. Dans cette figure, la table de niveau 1 indexe 256 tables dont chacune indexe 256 tables dont chacune indexe 256 blocs. Donc, la taille maximale du fichier à indexer est de 256*256*256 blocs. Une manière qui permet de

retrouver un bloc est comme suit : La fonction de hachage va donner le numéro du bloc logique correspondant à la clé donnée, soit x. On transforme x comme suit :

X= (x2*256 + r3) : où r3 est le numéro de l'entrée dans la table de niveau 3, qui donne l'adresse physique du bloc recherché, dont le numéro au niveau 2 est x2.

x2= (r1*256 + r2) : où r2 est le numéro de l'entrée dans la table de niveau 2, qui donne l'adresse de la table au niveau 3 qui donne l'adresse physique du bloc recherché, dont le numéro de l'entrée dans la table au niveau 1 est r1. r1 est nécessairement inférieure à 256.

$$X= ((r1*256+ r2)*256 + r3) =$$

$$(256*256*r1 +r2*256)+r3=$$

$$X= r1*256^{2} + r2*256 + r3$$

Donc, pour retrouver le bloc recherché, on accède à l'entrée r1 dans la table de niveau 1 pour trouver l'adresse de la table de niveau 2 concernée. A l'entrée r2 de cette dernière table, on trouve l'adresse de la table de niveau 3 concernée. A l'entrée r3 de cette dernière table, on trouve l'adresse du bloc physique recherché.

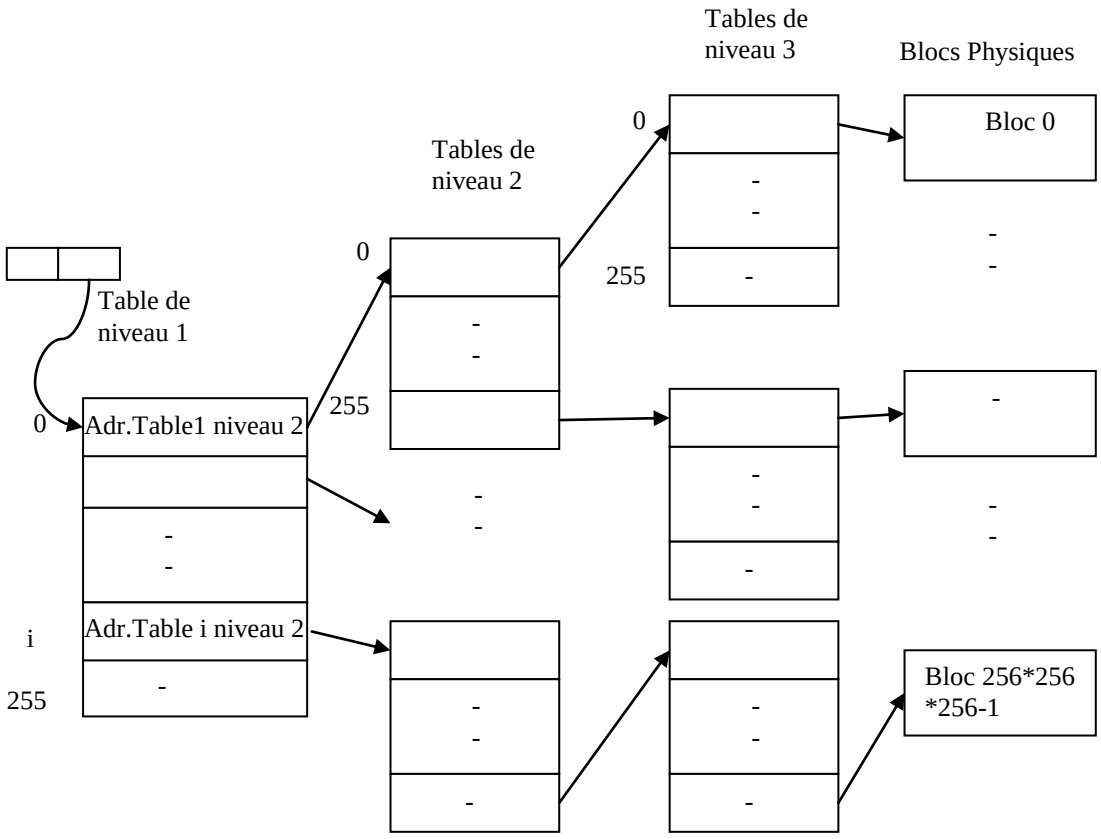


Figure 6.13 : *Table d'indexation à plusieurs niveaux.*

3.2 Gestion de l'espace libre d'un disque

La section précédente a montré comment garder trace des fichiers stockés sur supports externe. A présent, il est question d'examiner les méthodes utilisées pour gérer et retrouver les espaces libres. Deux méthodes principales sont utilisées, la méthode de liste chainée et la méthode de table de bits. Il est à souligner qu'un premier problème à résoudre est celui de la

taille d'un bloc physique. Celle-ci varie d'un système de gestion de fichiers à un autre. Elle est dictée par des considérations de minimisation de la fragmentation interne des fichiers. Par conséquent, elle est déduite en se basant sur des statistiques faites sur des études de cas réelles de tailles de fichiers utilisées dans différents domaines, privés, entreprises ou d'établissements administratives ou d'éducation. Elle peut être de 256 octets, de 512 octets, de 1 Ko ou de 4 Ko ou plus.

3.2.1 Méthode de liste chainée

Dans cette méthode, des blocs chainés entre eux sont réservés pour contenir, chacun et pour chaque entrée le numéro d'un bloc libre (voir figure 6.14). Si on admet un bloc de taille 512 octets et des numéros de blocs de 32 bits, 127 blocs sont alors référencés et une entrée est utilisée pour le chainage des blocs. Si le disque est de taille 100 Go qui représente environ 195 millions de blocs, il faut 1,5 millions de blocs pour conserver toutes ses adresses.

La gestion de l'espace libre implique la mise à jour de cette liste suite aux opérations de créations et de destructions de fichiers en enregistrant ou on enlevant des blocs libres. Donc, au moins un bloc de cette liste est requis en mémoire principale. Il est commode de garder en mémoire le dernier bloc de la liste, et quand il devient plein, il sera enregistré sur disque mais un effet de ping pong peut être observé si un bloc est presque plein et qu'une suite alternative de libérations, acquisitions de blocs est observée avec une taille égale ou supérieure au nombre d'entrées restantes dans le bloc en mémoire.

Pour optimiser l'espace de stockage des blocs libres, on peut regrouper les blocs libres en suites contiguës. Dans ce cas, on aura besoin de deux informations pour garder trace de chaque suite de blocs libre : le numéro du premier bloc et la longueur de cette suite en nombre de blocs. Donc, un compteur de taille donnée (8 octets, par exemple) de plus est alors nécessaire ; Par conséquent, l'espace global pour référencer les blocs libres est alors sensiblement diminué. L'inconvénient majeur est que la gestion de l'espace libre devient couteuse si le disque est trop fragmenté.

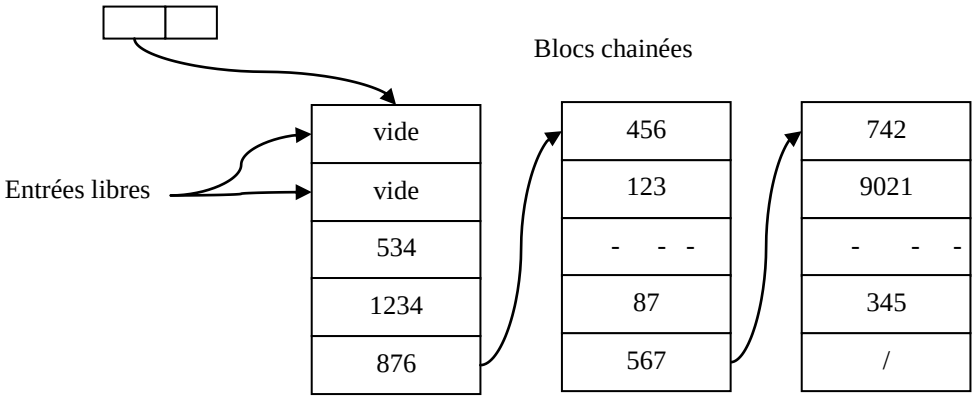


Figure 6.14 : *Gestion de l'espace libre par une liste chainée de blocs.*

3.2.2 Méthode de table de bits

Dans ce cas, on utilise une table de bits et la valeur de chacun donne l'état libre ou occupé du bloc correspondant (par exemple, 1 pour occupé et 0 pour libre). Sa taille est alors *statique* et est calculée en fonction de la taille du disque. Le changement de l'état d'un bloc implique l'inversement de la valeur du bit correspondant dans la table. Un seul bloc de cette table est requis en mémoire principale, et il est copié sur disque une fois devenu plein. Le fait que les

blocs physiques référencées par un même bloc de la table sont proches les uns des autres, cela minimise les mouvements successives du bras relativement à la première méthode.

Il est clair que la taille de cette table est réduite d'un facteur égal à la taille d'une adresse sur disque. Par exemple, si l'adresse disque porte sur 32 bits le facteur est de 32, ce qui très considérable relativement à la première méthode. En échange, la première méthode est intéressante, quand le disque devient presque plein. Dans ce cas, étant donné que la taille de la liste chainée est *dynamique*, elle occupe très peut d'espace.

3.3 Gestion des fichiers actifs

La plupart de ces fonctions impliquent la recherche de l'entrée (descripteur) du fichier dans le répertoire sur le support de stockage. Pour éviter cette recherche continue, l'opération d'ouverture permet de charger et de garder une copie de ce descripteur en mémoire principale dans une table de fichiers ouverts et un indice (qui correspond au numéro de l'entrée du fichier dans cette table) est utilisé à chaque référence à ce fichier. Cette table, appelée table des fichiers actifs, contient les descripteurs de tous les fichiers ouverts dans le système. Elle est donc commune à tous les utilisateurs du système. Les informations du descripteur sont principalement :

- Un compteur d'ouverture du fichier : un même fichier peut être ouvert par plusieurs processus. Ce compteur enregistre le nombre de ces ouvertures simultanées. Il est mis à jour à chaque nouvelle ouverture ou fermeture. Quand le dernier utilisateur ferme le fichier, le descripteur est supprimé de la table.
- L'adresse physique du fichier : La plupart des opérations impliquent la modification de données dans le fichier. Cette adresse permet d'accéder au fichier avec un coût moindre. Elle peut être une simple adresse ou un pointeur vers une structure qui contient les adresses des différents blocs du fichier.

D'autres informations concernant le fichier et propre à chaque utilisateur sont nécessaires tel que le pointeur de fichier, il indique la position suivante dans le fichier où aura lieu soit la lecture ou l'écriture. Une autre table pour chaque processus est alors entretenue telle que décrit sur la figure 6.15.

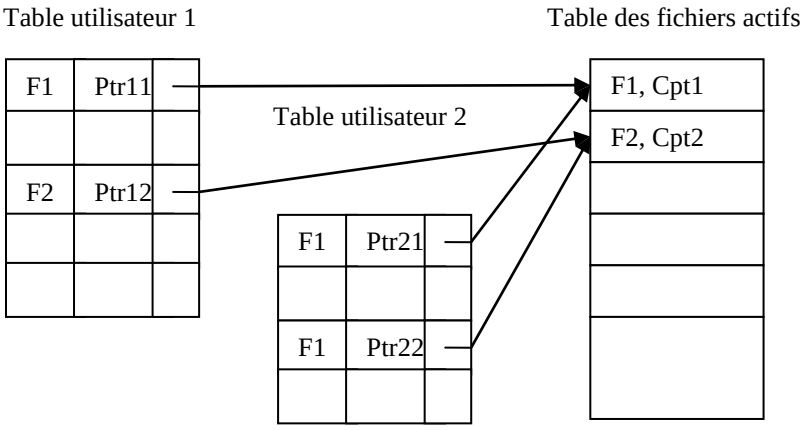


Figure 6.15 : *Tables des fichiers actifs.*