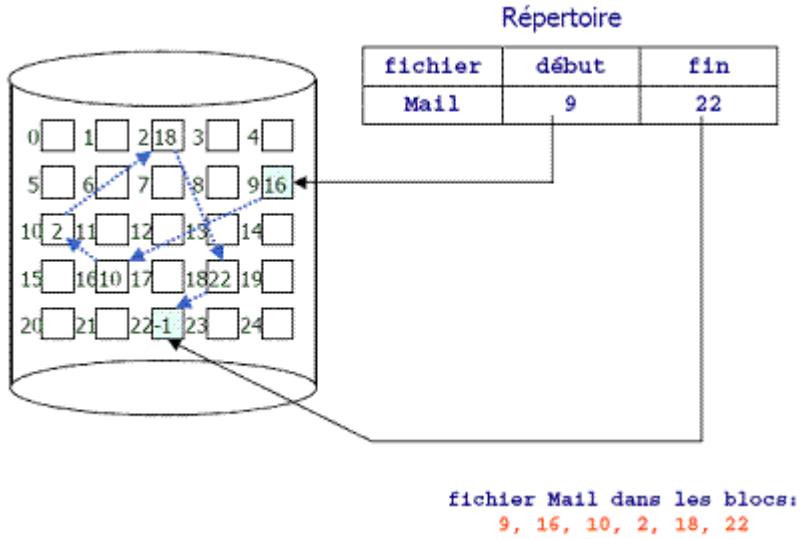
Systèmes d'Exploitation

Gestion de fichiers



Y. Challal, H. Bettahar, M. Vayssade

Table des matières

Objectifs

- Analyser le fonctionnement du sous-système de gestion de fichiers

Système de Gestion de Fichiers

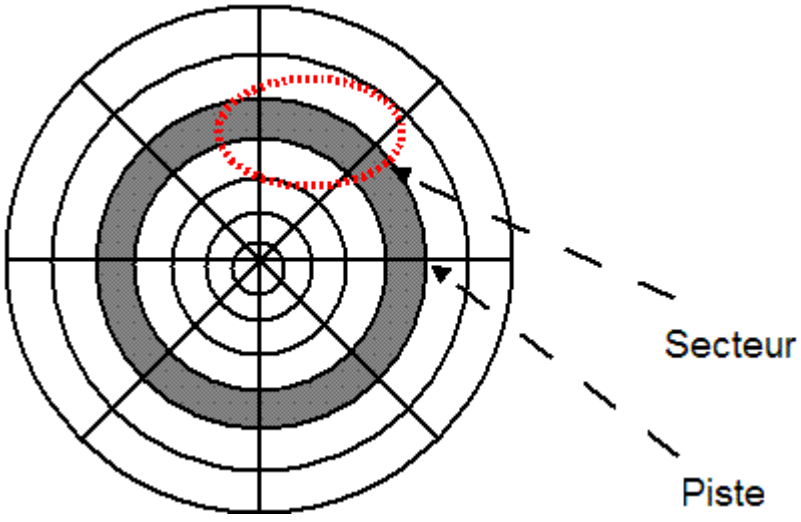
I

A. Gestion des entrées sorties sur disque [Beauquier]

1. Caractéristiques physiques d'un disque dur

Structure

Un disque est composé d'un ou plusieurs plateaux magnétiques ou optiques. Chaque face du plateau est divisée en pistes, qui sont des cercles concentriques, et en "quartier", d'angles fixe. L'intersection d'une piste et d'un quartier s'appelle un secteur.



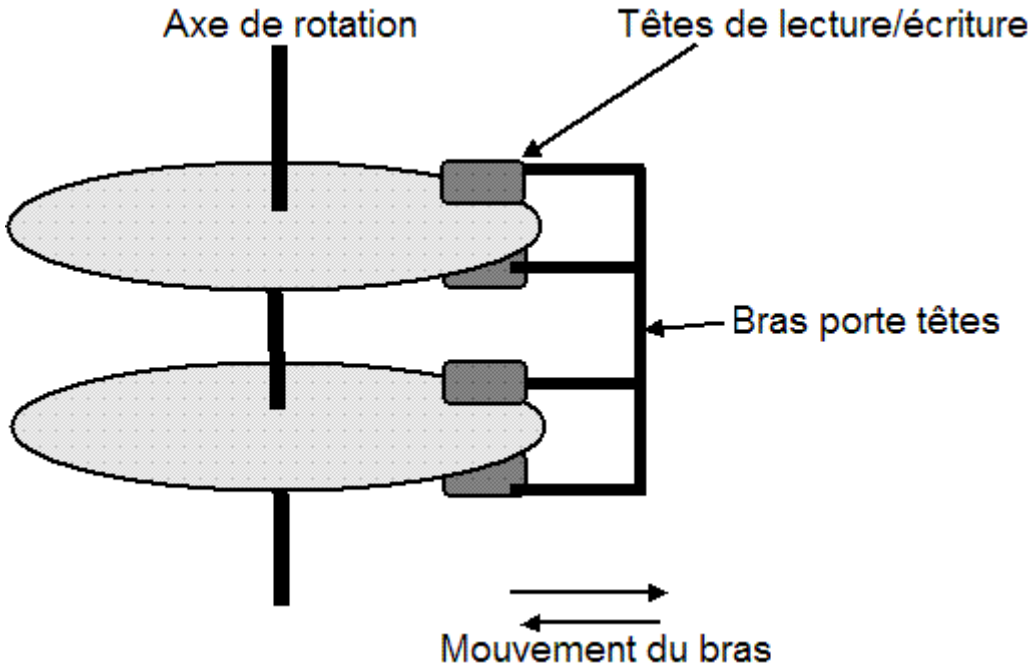
Division d'un disque en secteurs

Les secteurs sont organisés en deux parties : des octets "utiles", et des octets

contenant diverses informations de contrôle, comme le numéro de secteur et bits de parité. La plupart du temps, le secteur correspond à ce que nous avons appelé bloc, c'est-à-dire à l'unité de transfert entre le périphérique et la mémoire centrale. Cependant un bloc peut dans certains cas être composé de plusieurs secteurs.

Accès

L'accès à une face d'un disque s'effectue au moyen d'un bras, supportant une tête de lecture-écriture. Le bras avance ou recule pour atteindre les différentes pistes et le disque est entraîné par un moteur dans un mouvement de rotation de façon que la tête puisse atteindre tous les secteurs.



Disque dur à deux plateaux

La réalisation d'une opération d'entrée-sortie comprend plusieurs étapes. La première est l'attente dans une file du dispositif de lecture-écriture, si celui-ci n'est pas disponible. Ensuite, le bras est déplacé de sorte que les têtes se trouvent positionnées sur le bon cylindre. La durée correspondant à ce déplacement s'appelle **temps de recherche**.

Puis la rotation du disque amène ce secteur sous la tête de lecture-écriture, avec une durée appelée **temps de latence**. Enfin, le transfert depuis ou vers la mémoire centrale est effectué, avec un **temps de transfert**.

Le temps de recherche est généralement le plus élevé des deux et l'objectif de la plupart des algorithmes d'ordonnancement est de le minimiser.

2. Ordonnancement des requêtes du disque

a) Ordonnancement dans l'ordre d'arrivée (FIFO)

Les requêtes sont traitées dans leur ordre d'arrivée. Bien qu'il s'agisse d'une stratégie équitable, les performances sont parfois très médiocres.

Exemple

Supposons qu'un disque contienne 20 pistes, numérotées de 0 à 19, et que la tête de lecture-écriture soit positionnée à la piste numéro 14. Une requête étant

représentée par son numéro de piste, la file d'attente contient les requêtes suivantes dans l'ordre d'arrivée : 17, 18, 4, 11, 2, 12. En appliquant l'algorithme FIFO, le nombre total de piste parcourues est 44 pistes.

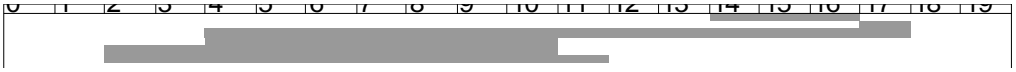


Tableau 1 : Mouvements de la tête avec FIFO

b) Ordonnancement suivant le plus court temps de recherche

Dans cet algorithme, la prochaine requête traitée est celle pour laquelle le déplacement de la tête est minimal, à partir de la position courante.

Exemple

En considérant la liste de requêtes de l'exemple précédent et la stratégie PCTR, le nombre total de pistes parcourues devient égal à 26.

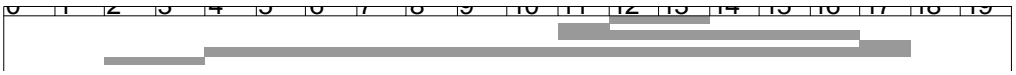


Tableau 2 : Mouvements de la tête avec PCTR

Si de nouvelles requêtes arrivent en cours de traitement, un phénomène de famine peut se produire. Supposons par exemple qu'il arrive un flot continu de requêtes pour des pistes proches de 14. Ces requêtes sont traitées par PCTR avant celles qui concernent les pistes 2 et 4, obligeant ces dernières à attendre indéfiniment.

c) Ordonnancement par balayage

Cet algorithme parcourt toutes les pistes dans une direction donnée, par exemple vers l'intérieur, et traite au fur et à mesure les requêtes qu'il rencontre. Ensuite, la tête change de direction et balaie toutes les pistes vers l'extérieur. Cette version de balayage est appelée SCAN. Dans la version connue sous le nom LOOK, la tête ne va pas jusqu'au bout des pistes mais repart dans l'autre sens dès qu'il n'y plus de requête en attente.

Exemple

En considérant la même liste de requête des exemple précédent, le nombre total de piste parcourues avec l'algorithme LOOK est 20 pistes.

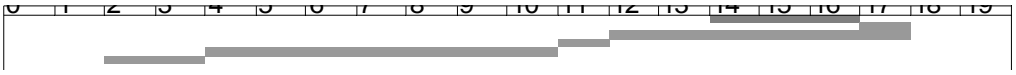


Tableau 3 : Mouvements de la tête avec LOOK

B. Concept de fichier

Définition : Fichier

On désigne sous le nom de **fichier** un ensemble d'informations regroupées en vue de leur conservation et de leur utilisation dans un système informatique. Les fichiers ont le plus souvent une durée de vie supérieure à la durée de l'exécution d'un programme ou à celle d'une séance de travail: leur support permanent est donc la mémoire secondaire *[krakowiak87]*.

Définition : Système de gestion de fichiers

Un fichier est un objet qui possède un nom qui permet de le désigner et il est muni de fonctions d'accès, c'est-à-dire d'opérations qui permettent de consulter ou de modifier les informations qu'il contient. La partie d'un système d'exploitation qui assure la conservation des fichiers et réalise les fonctions d'accès est appelée **système de gestion de fichiers** (en abrégé SGF) .

Définition : Bloc

L'unité d'allocation sur un disque est le **bloc**, qui est une unité logique qui peut contenir un nombre variable de secteurs physiques; ce nombre étant fonction de la taille du bloc qui est fixé par l'administrateur du système à la création de la partition.

Remarque : Bloc : unité élémentaire d'allocation

Selon sa taille, un fichier occupe sur le disque un certain nombre de blocs. Les blocs sont alloués entièrement au fichier; autrement dit, si la taille du bloc est de 2 Ko, un fichier de 2100 octets occupera 2 blocs, soit 4096 octets *[bouzefrane03]*.

Il existe plusieurs techniques d'allocation de blocs aux fichiers, que nous détaillons dans ce qui suit.

C. Allocation contiguë

Définition : Allocation contiguë

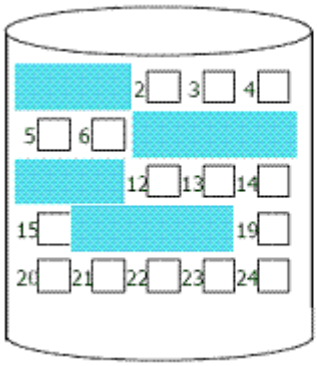
Elle consiste à trouver suffisamment de place sur l'unité de stockage, pour stocker le fichier de manière contiguë.

Remarque : évolution du fichier et fragmentation

La première contrainte de cette technique d'allocation est évidemment la connaissance de la taille du fichier au moment de sa création car il faut lui réserver dès le départ suffisamment de place. La taille du fichier peut évoluer durant son existence. Autrement dit, l'espace réservé peut devenir insuffisant ou, au contraire, partiellement inutilisé: on parle de **fragmentation** de l'unité de stockage *[bouzefrane03]*.

Structure du répertoire

Dans cette technique d'allocation, la structure répertoire contient le nom du fichier, son bloc disque de départ et le nombre de blocs faisant partie du fichier. La figure (cf. 'Allocation contiguë' p 11) illustre cette technique de stockage.



Répertoire

fichier	début	longueur
mail	0	2
rapport	7	5
temp	16	3

Allocation contiguë

D. Allocation avec une liste chaînée

Méthode : Liste chaînée des blocs

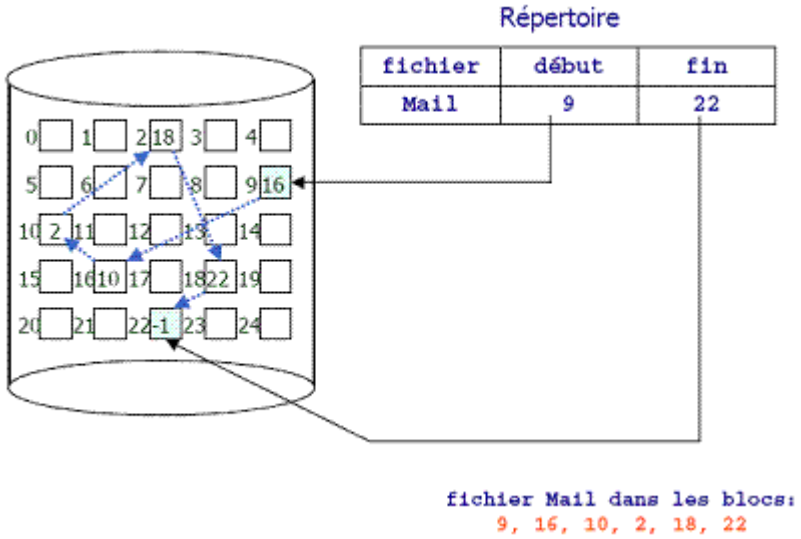
Pour éviter le problème de fragmentation, cette technique place les blocs du fichier à des endroits différents sur l'unité de stockage ; ces blocs sont gérés à l'aide d'une liste chaînée *[bouzefrane03]*.

Remarque : Lenteur d'accès

L'accès à un bloc du fichier reste toutefois lent car il faut accéder aux différents éléments de la liste à différents endroits de l'unité de stockage .

Structure d'un répertoire

Dans cette technique d'allocation, la structure répertoire contient le nom du fichier, le bloc de départ et le bloc de fin. La figure (cf. 'Allocation avec une liste chaînée' p 11) illustre cette technique de stockage.

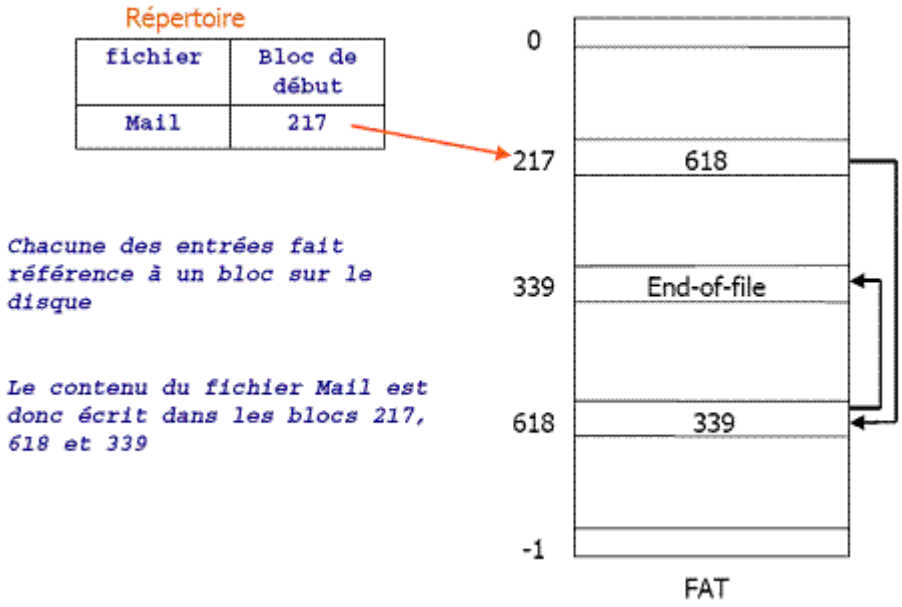


Allocation avec une liste chaînée

E. Allocation avec une liste chaînée indexée

Méthode : File Allocation Table (FAT)

Cette technique d'allocation gère aussi une liste chaînée de blocs. Mais pour éviter les accès, qui peuvent être nombreux, aux éléments de la liste chaînée sur l'unité de disque, elle implémente la liste dans une table en mémoire (dite FAT: File Allocation Table). Chaque entrée de la table possède un numéro du bloc physique ainsi que celui qui le suit dans la liste *[bouzefrane03]* (voir figure (cf. 'FAT: File Allocation Table' p 12)).



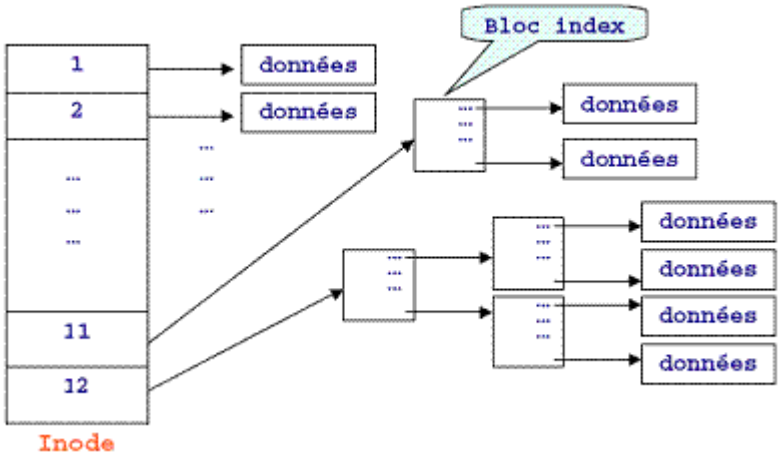
FAT: File Allocation Table

F. Allocation avec des i-noeuds

Définition : i-noeud

Les systèmes de fichier d'Unix disposent d'une table à n entrées, n est fixé au moment de la création du système de fichier. Chaque entrée est appelée i-noeud et est associée à un seul fichier. Un i-noeud contient des informations de description du fichier ainsi que 12 adresses physiques des blocs du fichier. Les 10 premières adresses pointent sur les 10 premiers blocs du fichier. La 11ème adresse pointe sur un bloc d'index pouvant contenir 256 adresses de blocs; on a une simple indexation. La 12 ème adresse pointe sur un bloc d'index pouvant contenir 256 adresses de blocs, où chaque bloc contient 256 adresses de blocs de données; on a une double indexation.

La figure (cf. 'Allocation indexée avec i-noeuds' p 13) (cf. 'Allocation indexée avec i-noeuds' p 13)illustre le concept de i-noeud.



Allocation indexée avec i-noeuds

Exemple

Si la taille du bloc est de 4 Ko, un fichier de taille inférieure ou égale à 40 Ko n'utilisera pas d'indexation. Un fichier de taille comprise ente 40 Ko et 1 Mo utilise une indexation simple. Un fichier de taille supérieure à 1 Mo utilise une indexation double.

G. Opérations sur les fichiers

Structure logique d'un fichier

La vision logique d'un fichier consiste à le considérer comme une succession d'éléments structurés (enregistrements) ou non structurés (caractères).

Accès séquentiel vs. accès direct

L'accès aux éléments d'un fichier peut se faire de manière séquentielle, c'est-à-dire que l'accès à une donnée particulière nécessite d'abord la lecture des données qui la précèdent dans le fichier *[bouzefrane03]*. A tout instant, un curseur indique l'élément suivant à lire dans le fichier. Il est aussi possible d'accéder directement à une information lorsque celle-ci est contenue dans un fichier stocké sur disque. On parle dans ce cas d'accès direct.

1. Opérations sur les fichiers

Plusieurs opérations

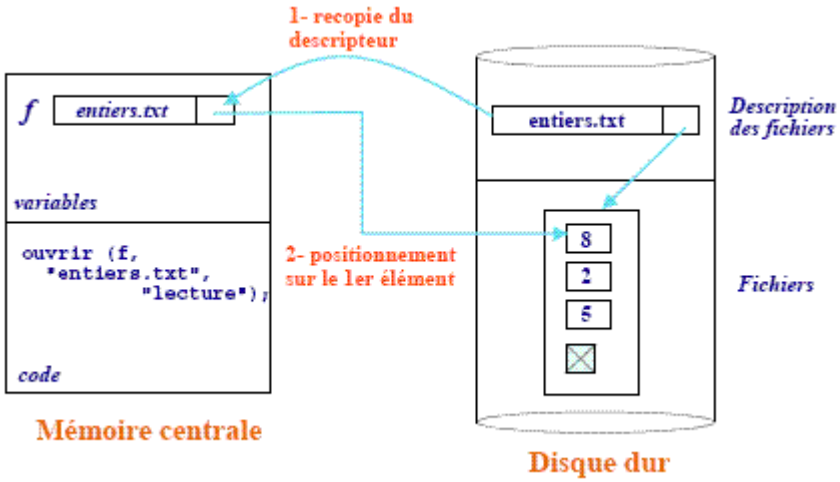
Différentes opérations peuvent être effectuées sur un fichier:

- Création d'un fichier;
- Lecture à partir d'un fichier;
- Ecriture dans un fichier;
- Rajout de données en fin de fichier;
- Accès direct à un endroit d'un fichier;
- Suppression d'un fichier, etc.

Méthode : Ouverture de fichiers

Tout fichier doit être ouvert avant son utilisation en lecture et/ou écriture. Lorsque le fichier est ouvert en lecture et/ou écriture, son curseur est positionné sur son

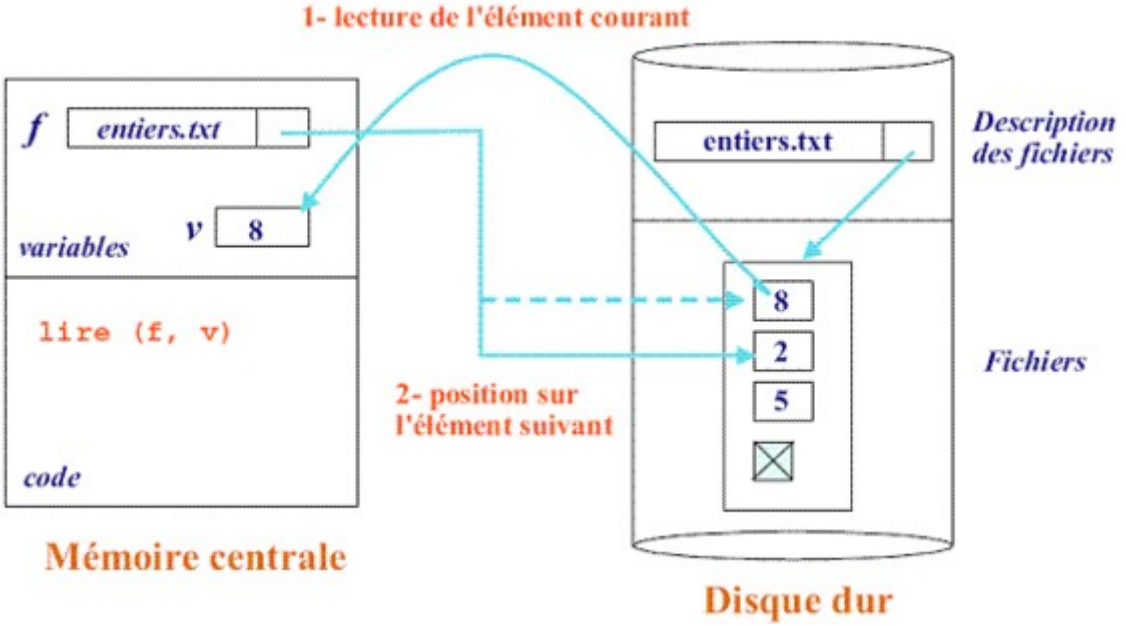
premier élément. La figure (cf. 'Ouverture d'un fichier' p 14) illustre l'opération d'ouverture.



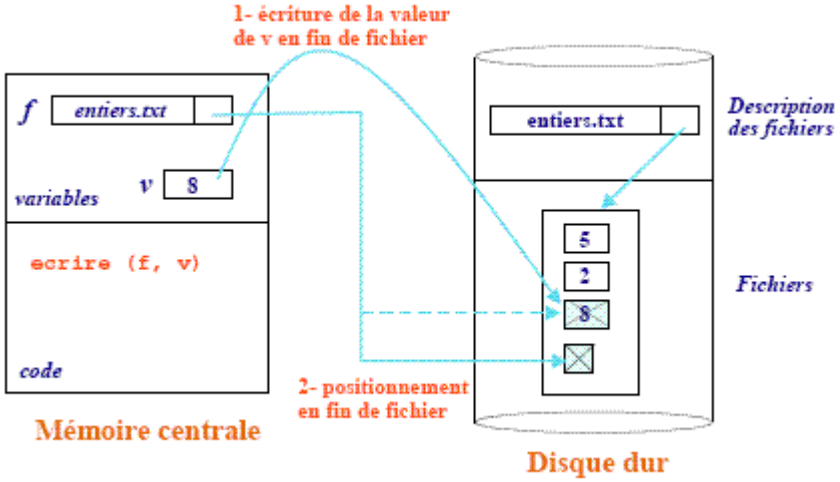
Ouverture d'un fichier

Méthode : Lecture/écriture dans un fichier

Les opérations de lecture et d'écriture permettent de lire ou d'écrire un certain nombre d'éléments séquentiellement. Lorsque un élément est lus ou écrit dans le fichier, le curseur avance vers l'élément suivants. Les figures (lecture (cf. 'Lecture à partir d'un fichier' p 14), écriture (cf. 'Écriture dans un fichier' p 15)) (cf. 'Lecture à partir d'un fichier' p 14) illustrent le déroulement des opérations de lecture et d'écriture d'un élément dans un fichier.



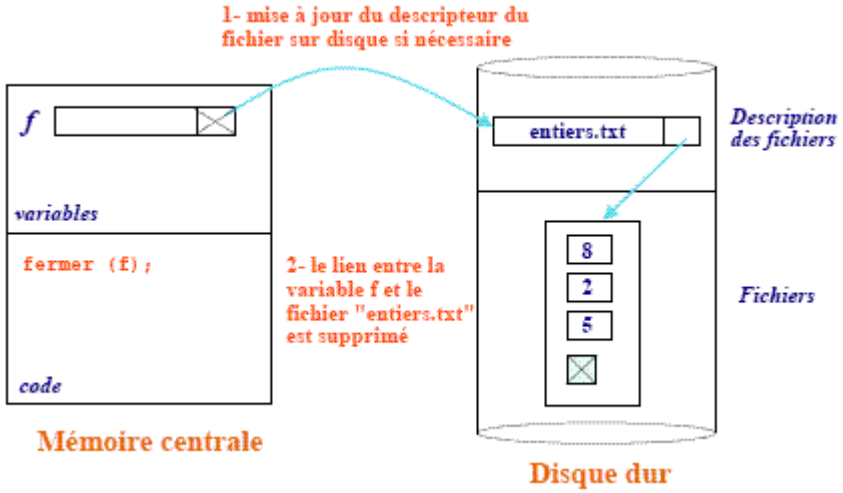
Lecture à partir d'un fichier



Écriture dans un fichier

Méthode : Fermeture d'un fichier

La fermeture d'un fichier entraîne la suppression du lien entre le descripteur de ce dernier en mémoire et le fichier physique. La figure (cf. 'Fermeture d'un fichier' p 15) (cf. 'Fermeture d'un fichier' p 15)illustre l'opération de fermeture.



Fermeture d'un fichier

H. Opérations sur les fichiers sous UNIX [bouzefrane03]

Méthode : Ouverture de fichiers

Sous UNIX, la primitive open est utilisée pour l'ouverture d'un fichier:

```
int open(const char *nom_fichier, int flags);
```

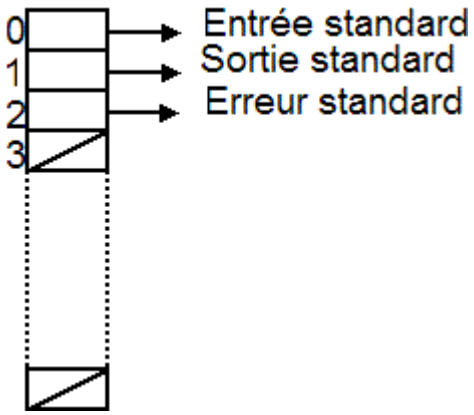
flags définit le mode d'ouverture du fichier "nom_fichier". Les constantes utilisées par flags sont définies dans le fichier <fcntl.h>:

- O_RDONLY pour une ouverture en lecture seule;
- O_WRONLY pour une ouverture en écriture seule;
- O_RDWR pour une ouverture en lecture/écriture;

- O_CREAT pour créer le fichier s'il n'exsite pas;
- O_TRUNC pour écraser le contenu du fichier;
- O_APPEND pour ouvrir le fichier en mode ajout (écriture à la fin du fichier);
- O_SYNC pour ouvrir le fichier en mode synchrone (toute mise à jour est directement écrite sur disque).

Table de descripteurs de fichiers d'un processus

La valeur entière retournée par la primitive open est l'indice de la nouvelle entrée créée dans la table des descripteurs de fichiers. En effet, à tout processus du système est associé une table des descripteurs (cf. 'Descripteurs de fichiers' p 18) qui contient au départ 3 entrées:



Descripteurs de fichiers

Table des fichiers ouverts

Tout fichier ouvert par un processus possède une entrée dans une table système dite table des fichiers ouverts. Son descripteur pointe sur son entrée dans cette table. Alors qu'une table des descripteurs existe pour chaque processus créé, la table des fichiers ouverts (cf. 'Table des fichiers ouverts' p 18) est commune à tous les processus. Une entrée dans la table des fichiers ouverts contient des informations telles que le mode d'accès au fichier, le pointeur du fichier, le numéro de l'i-noeud associé au fichier, etc.

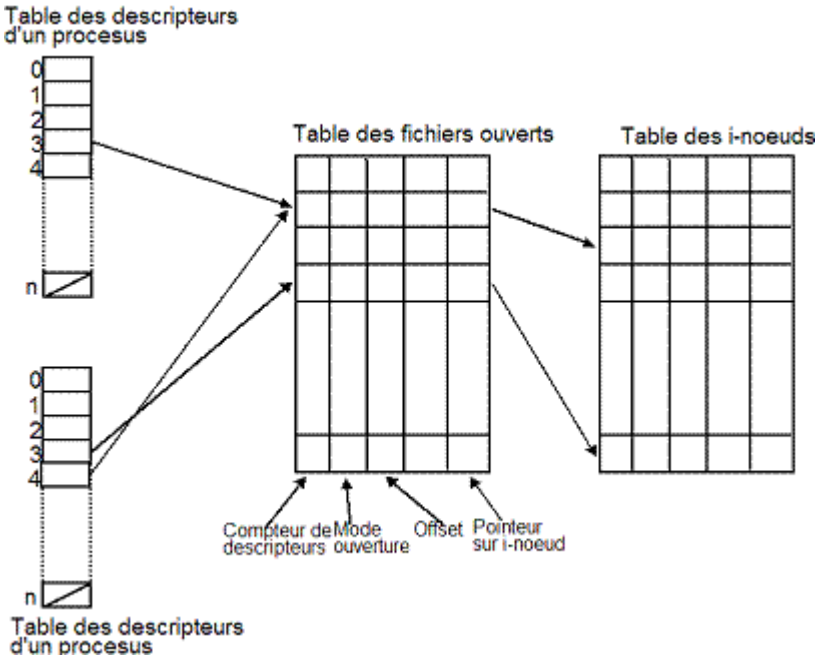


Table des fichiers ouverts

Méthode : Opérations de lecture / écriture

Sous UNIX, ces opérations manipulent des octets; tout fichier Unix est considéré comme une suite d'octets.

```
size_t read(int fd, void *buf, size_t count);
```

permet de lire count caractères dans le tampon mémoire buf à partir du fichier pointé par le descripteur fd.

```
size_t write(int fd, const char *buf, size_t count);
```

permet l'écriture de count caractères se trouvant dans le tampon mémoire pointé par buf vers le fichier de descripteur fd.

Méthode : Accès direct à un fichier

La primitive suivante permet de positionner le pointeur de lecture/écriture à un endroit précis du fichier:

```
off_t lseek(int fd, off_t offset, int whence);
```

offset est un déplacement dans le fichier de descripteur fd par rapport à une base spécifiée par whence:

- SEEK_SET : Positionnement par rapport au début du fichier
- SEEK_CUR : Positionnement par rapport à la position courante du fichier
- SEEK_END : Positionnement par rapport à la fin du fichier

Méthode : Fermeture de fichiers

Tout fichier ouvert doit être fermé en fin d'utilisation.

```
int close(int fd);
```

fd est le descripteur d'entrée/sortie retourné par la primitive open.

Méthode : Duplication de descripteurs

int dup(int desc);

Duplique le descripteur "desc" sans ouvrir un nouveau fichier.

```
int dup2(int desc1, int desc2);
```

Force la duplication de "desc1" vers "desc2".

Entrées / Sorties sous UNIX

Les entrées / sorties sous UNIX passent par des fichiers. On retrouve alors :

- les fichiers de données réguliers, dont la structure est laissée à l'application,
- les fichiers répertoires,
- les fichiers associés aux ressources du système (/dev),
- les fichiers associés aux supports magnétiques (/dev/dsk) (blocs), et (/dev/rdsk) (caractères),
- les fichiers associés aux terminaux (/dev/tty),
- les tubes nommés (pipes) qui servent pour la communication inter-processus,
- les sockets qui permettent la communication distantes à travers un réseau,
- les liens symboliques, etc.