TD 1
Les fichiers

EXERCICE 1 ► Allocation par blocs contigus

Dans l'allocation par blocs contigus, un fichier est stocké sur le disque sous la forme d'un ensemble de blocs qui sont forcément voisins les uns des autres (contigus!), et restent dans un ordre fixe (les premiers octets du fichier sont dans le bloc d'adresse la plus faible, les derniers octets dans le bloc d'adresse la plus haute). La figure 1.1 donne un exemple dans lequel 3 fichiers A, B et C ont été placés sur un disque divisé en 20 blocs, d'adresses 1 à 20. La taille des blocs n'a pas d'importance dans cet exercice.

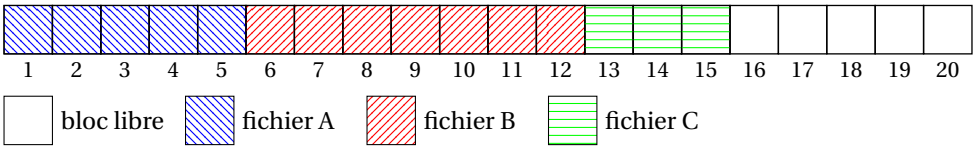

FIGURE 1.1 – Un exemple d'allocation de blocs contigus.

1) Donnez l'évolution de l'allocation du disque après chacune des opérations suivantes :
- — création d'un fichier D de 2 blocs,
- — suppression du fichier B,
- — ajout d'un fichier E de 5 blocs,

2) Donnez la représentation, sous la forme d'un vecteur de bits (une *bitmap*), des blocs libres ou occupés du disque. Imaginons qu'un disque de 1 Tio soit divisé en blocs de 1 Mio : quelle serait en kio la taille du vecteur de bits nécessaire pour représenter l'état (libre ou occupé) des blocs du disque ?

3) Dans l'état actuel de l'allocation sur le disque, quel est le plus gros fichier (en nombre de blocs) que l'on peut encore ajouter ? Si on voulait ajouter un fichier composé de 4 blocs, que faudrait-il faire au préalable ?

4) Imaginons que l'on a pas ajouté de nouveau fichier, mais que l'on veuille maintenant ajouter un bloc au fichier A : que faudrait-il faire ?

EXERCICE 2 ► Allocation par blocs chaînés

Dans l'allocation par blocs chaînés, un fichier est stocké sous la forme d'une liste chaînée de blocs qui peuvent être dispersés sur le périphérique. Une manière de procéder est de réserver quelques octets en fin de chaque bloc afin de stocker l'adresse du bloc suivant. Quand il s'agit du dernier bloc d'un fichier, on stocke une valeur particulière pour indiquer qu'il n'y a pas de bloc suivant.

Un exemple d'allocation par blocs chaînés est représenté sur la figure 1.2 ; dans cet exemple, 3 fichiers sont présents sur le disque, dont l'espace est divisé en 20 blocs d'adresses 1 à 20.

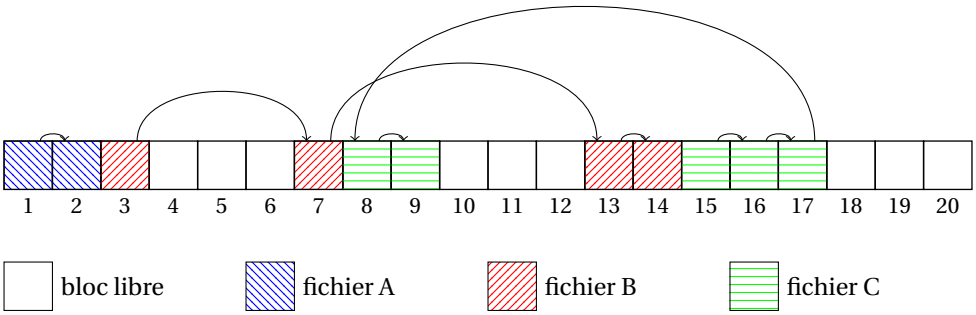

FIGURE 1.2 – Un exemple d'allocation de blocs chaînés.

Une variante du chaînage par bloc consiste à utiliser une table d'allocation des fichiers (table FAT, pour *File Allocation Table*). La table suivante représente la même allocation des blocs que dans l'exemple précédent, mais sous la forme d'une table d'allocation. À chaque adresse de la table, on a

— soit l'adresse du bloc suivant dans le fichier,

— soit une valeur particulière indiquant que l'on est en fin de fichier (notée F ci-dessous),

— soit une valeur particulière indiquant que le bloc est libre (notée V ci-dessous).

Voici par exemple la partie de la table d'allocation qui correspond à l'allocation représentée sur la figure 1.2 :

adresse	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
suivant	02	F	07	V	V	V	13	09	F	V	V	V	14	F	16	17	08	V	V	V

1) Dans le système de fichier FAT16, les adresses de blocs sont codées sur 16 bits : combien de blocs peuvent être gérés au plus dans ce système ? Quel espace maximal peut être géré, avec des blocs de 8, 16 ou 32 kio ?

2) Donnez une représentation du chaînage des blocs pour la table d'allocation suivante :

adresse	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
suivant	02	04	06	05	F	07	08	F	F	11	12	13	F	V	V	V	V	V	V	V

Vous donnerez les noms A, B, C, D aux fichiers présents, en commençant par A à l'adresse la plus basse.

3) Donnez l'évolution de la table d'allocation précédente, après chacune des opérations suivantes :

— ajout de 2 blocs au fichier C,

— suppression du fichier B,

— ajout de 4 blocs au fichier C.

On suppose que les demandes sont traitées bloc par bloc, et que c'est le bloc libre d'adresse la plus basse qui est alloué en premier lors d'une demande d'allocation.

4) On dit parfois que « le système de fichier FAT a tendance à beaucoup fragmenter les fichiers » : commenter cette affirmation en vous appuyant sur votre réponse à la question précédente.

EXERCICE 3 ► **Inœuds (d'après un exercice de T. Lavergne, Univ. Paris Saclay.)**

On considère un système de fichiers dans lequel l'information concernant les blocs de données de chaque fichier est accessible à partir de son inœud (c'est une variante de l'ext2 de Linux).

On suppose dans cet exercice que :

— le système de fichiers utilise des blocs de données de taille fixe de 1 kio (1024 octets).

— l'inœud de chaque fichier (ou répertoire) contient 12 pointeurs directs sur des blocs de données, 1 pointeur indirect simple, 1 pointeur indirect double et 1 pointeur indirect triple.

— chaque pointeur (adresse de bloc) est représenté sur 4 octets.

La situation est représentée sur la figure 1.3 (extraite de `https://en.wikipedia.org/wiki/Ext2`).

1) Avec les données dont vous disposez dans l'énoncé, estimez la plus grande taille de données que ce système peut stocker en un seul fichier.

2) On considère un fichier contenant 100 000 octets : combien de blocs faut-il (au total) pour représenter ce fichier sur disque ?

3) Même question avec un fichier de 1 Mo de données.

1.1 Primitives `read` et `write`

EXERCICE 4 ► **Écritures sur le disque avec write**

On reproduit ci-dessous un extrait de la page de manuel (release 4.15 of the Linux man-pages project, 2018-02-02) de la fonction C `write()`. Vous vous référerez à cet extrait pour la suite de l'exercice.

```
NAME
       write - write to a file descriptor
SYNOPSIS
       #include <unistd.h>
       ssize_t write(int fd, const void *buf, size_t count);
DESCRIPTION
       write()  writes  up  to count bytes from the buffer starting at buf to the file referred to by the
       file descriptor fd.
```

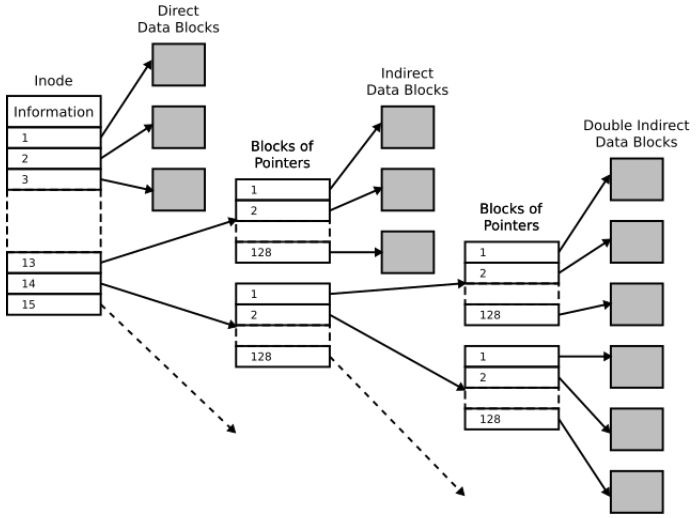

FIGURE 1.3 – Organisation d'un fichier à partir d'un inœud ext2 inodes (Timtjtim, CC by SA). Attention, la taille des blocs choisie sur ce schéma n'est pas la même que dans l'énoncé!

```
       The number of bytes written may be less than count if, for example, there is insufficient space on
       the underlying physical medium, or the RLIMIT_FSIZE resource limit is encountered (see setr-
       limit(2)), or the call was interrupted by a signal handler after having written less than count
       bytes. (See also pipe(7).)

       For a seekable file (i.e., one to which lseek(2) may be applied, for example, a regular file)
       writing takes place at the file offset, and the file offset is incremented by the number of bytes
       actually written. If the file was open(2)ed with O_APPEND, the file offset is first set to the
       end of the file before writing. The adjustment of the file offset and the write operation are
       performed as an atomic step.
       [...]
RETURN VALUE
       On success, the number of bytes written is returned (zero indicates nothing was written). It is
       not an error if this number is smaller than the number of bytes requested; this may happen for
       example because the disk device was filled. See also NOTES.

       On error, -1 is returned, and errno is set appropriately.

       If count is zero and fd refers to a regular file, then write() may return a failure status if one
       of the errors below is detected. If no errors are detected, or error detection is not performed,
       0 will be returned without causing any other effect. If count is zero and fd refers to a file
       other than a regular file, the results are not specified.
NOTES
       The types size_t and ssize_t are, respectively, unsigned and signed integer data types specified
       by POSIX.1.
       [...]

       If a write() is interrupted by a signal handler before any bytes are written, then the call fails
       with the error EINTR; if it is interrupted after at least one byte has been written, the call suc-
       ceeds, and returns the number of bytes written.
       [...]
```

1) Ré-expliquez, avec vos propres mots ce que permet la fonction `write()` ? (détaillez les paramètres et la première phrase de la `DESCRIPTION`).

2) On suppose qu'un programme initialise un tableau de caractères déclaré par `char tab[64]`. Comment appeler `write` pour écrire (au plus) tout le contenu de `tab` sur le descripteur de fichier `fd`? Même question si le tableau a été déclaré par `long tab[64]`. Proposez des solutions avec ou sans `sizeof`.

3) Comment écrire une (potentiellement longue) chaîne de caractères de type `char*` sur le disque, via un descripteur de fichier `fd` **caractère par caractère**? Donnez une fonction `writech` pour cela (prototype et définition). Donnez un exemple d'appel de votre fonction.

EXERCICE 5 ► Lectures sur le disque avec read

On reproduit ci-dessous un extrait de la page de manuel de la fonction C `read()`. Vous vous référerez à cet extrait pour la suite de l'exercice.

```
NAME
       read - read from a file descriptor
SYNOPSIS
       #include <unistd.h>
       ssize_t read(int fd, void *buf, size_t count);
DESCRIPTION
       read()  attempts to read up to count bytes from file descriptor fd  into the buffer
       starting at buf.

       On files that support seeking, the read operation commences at the file offset, and
       the file offset is incremented  by the number of bytes read.  If the file offset is
       at or past the end of file, no  bytes  are  read, and read() returns zero.

       If count is zero, read() may detect the errors described below.  In the  absence of
       any errors,  or if  read() does not check for errors,  a read() with a count  of  0
       returns zero and has no other effects.
       [...]
RETURN VALUE
       On success, the number of bytes read  is returned (zero indicates end of file), and
       the file position is advanced by this number. It is not an  error if this number is
       smaller than the  number  of  bytes requested; this  may happen for example because
       [...]

       On error, -1 is returned, and  errno  is set appropriately.  In  this case,  it  is
       left  unspecified  whether  the  file  position (if any) changes.
       [...]
NOTES
       The types  size_t  and  ssize_t  are, respectively, unsigned and signed integer data
       types specified by POSIX.1.
```

1) Ré-expliquez, avec vos propres mots ce que permet la fonction `read()` ? (détaillez les paramètres et la première phrase de la `DESCRIPTION`).
2) Utiliser cette fonction pour écrire une fonction d'en-tête `void readch_stdin(char *buf, size_t count)` pour lire exactement `count` caractères sur sur l'entrée standard ; votre fonction tentera de lire en une seule fois `count` caractères, puis achèvera la lecture caractère par caractère.

1.2 Fichiers en ligne de commande

EXERCICE 6 ▶ **Arborescence — Droits sous unix**
Sur le système considéré, il y a 4 utilisateurs :

— `fontaine` qui fait partie du groupe `prof` et `user` ;
— `elise` qui fait partie des groupes `etu` et `user` ;
— `hippolyte` qui fait partie du groupe `prof`.
— `root` qui est l'administrateur et fait partie du groupe `root`

```
drwxr-x--x 27      root  user   /bin/
-rwsr-xr-x  1      root   etu   /bin/visioneurPDF
drwxr-xr-x 80      root  user   /home/
drwxr-x--x 10  fontaine  prof   /home/fontaine/
drwx--x---  4  fontaine  prof   /home/fontaine/prive/
-rw-r-x---  1  fontaine  prof   /home/fontaine/sujet.pdf
-rw-r--r--  1 hippolyte  prof   /home/fontaine/prive/correction.pdf
-rw-rw----  1  fontaine  prof   /home/fontaine/prive/notes.ods
```

Attention, pour les fichiers vous devez tenir compte des droits des répertoires et sous répertoires. Comme son nom l'indique, `visioneurPDF` est un logiciel pour visualiser les PDF...

1) Représenter l'arborescence.
2) Quelle commande peut faire `hippolyte` pour avoir une copie du fichier `correction.pdf` la racine de son compte ? En a-t-il le droit ?
3) Représenter les possibilités d'accès des 4 utilisateurs aux fichiers `visioneurPDF`, `sujet.pdf`, `correction.pdf` et `notes.ods`, par un tableau à double entrée.
4) Élise peut-elle copier le fichier `correction.pdf` ? Peut-elle visualiser ce fichier ?