F.S

Questions principales

Décrivez la structure d'une partition formatée en F.A.T Le schéma ci-dessous représente une partition formatée en F.A.T

```
-----------------------------------------------
| BA | FAT | Copie FAT | Tableau de clusters |
-----------------------------------------------
```

- BA = Boot Area[1]
- FAT = Index

Le tableau de clusters est constitué de clusters de même taille. Un cluster ne peut contenir qu'un seul fichier.

1 cluster == n * secteur défini[2]

La FAT est l'index du tableau de clusters, il est chargé en RAM au démarrage pour des raisons de performance. La FAT contient autant d'entrée que de clusters présent dans le tableau de clusters. Cependant une entrée n'occupe pas nécessairement un cluster. La FAT contient l'adresse des clusters. Elle fait le lien entre les différents clusters d'un fichier.

Une entrée dans la fat peut avoir plusieurs valeurs :

- 0 si le cluster est libre [3]
- Une valeur positive pour désigner l'adresse du cluster suivant.
- -1 si c'est le dernier cluster.
- -2 si le cluster est défectueux. Comment savoir si un cluster est défectueux ? On écrit dedans, on lit ce qu'on a écrit, si les valeurs sont différentes, alors le cluster est défectueux.

Le répertoire racine se trouve trouve toujours au cluster 0 dans une partition formatée en FAT16. En FAT32, le cluster est donné par le MBR.

Les **répertoires en FAT sont des fichiers** particuliers contenant les métadonnées des fichiers présent dans le répertoire. Pour chaque fichier, le répertoire possède un descripteur de 32 bytes.

Les champs d'un descripteur :

[1] BA == BR == Boot Record
[2] cluster == bloc
[3] ne veut pas dire que le cluster est vide, seulement qu'il est libre

- Nom : permet de connaître le nom du fichier.
- Cluster : l'adresse du premier cluster du fichier.
- Dates : dates de modification, création, dernier accès.
- Extension : l'extension du fichier
- Taille : la taille du fichier.[4]

Un répertoire a donc toujours au moins deux descripteurs : le descripteur du répertoire courant et le parent (sauf la racine qui n'a pas de parent).

Avantages de la FAT :

- Très simple à intégrer.

Inconvénients de la FAT :

- Lourd en mémoire si la partition est trop grande.[5]
- Si le nom est trop long, il peut être étalé sur plusieurs entrées.
- Pas de droits d'accès.
- Fragmentation externe, les clusters ne sont pas toujours contigus lorsqu'on écrit et efface beaucoup.
- Fragmentation interne, un cluster ne peut contenir qu'un seul fichier, si celui-ci n'est pas remplit, on perd de la place.

Voir les questions secondaires FAT.

Décrivez la structure d'une partition formatée en EXT/EXT2 EXT :

En EXT, un mini-disque (une partition) est composée de quatres zones :

- Boot Area : Contient les informations sur le démarrage
- Super-bloc : Contient les informations sur le mini-disque
- Tableau d'inodes : contient les inodes (métadonnées des fichiers)
- Tableau de blocs : contient les blocs (les données)

[4] 32Bits !, raison de la taille max d'un fichier de 2^{32}
[5] #bloc * taille entrée FAT

```
-----------------------------------------------
| BA | SB | Tableau d'inodes | Tableau de blocs |
-----------------------------------------------
```

Le tableau d'inode contient un inode par fichier. Pour accéder à un fichier, il faut connaître l'inode, cet inode comprend :

- Nom du propriétaire
- Droits d'accès
- Dates
- Taille
- Type (fichier / dossier)
- Liste de tous les blocs du fichiers

Le chaînage est donc fait dans l'inode lui même. La liste de tous les blocs contient 10 pointeurs de blocs direct, et 3 pointeurs avec indirection.

- Le 11ème pointeur contient un seul niveau d'indirection, il pointe vers un tableau de blocs qui pointe vers les blocs de données.
- Le 12ème pointeur contient deux niveaux d'indirection, il pointe vers un tableau de blocs qui pointent eux même sur un tableau de blocs qui pointent sur les blocs de données.
- Le 13ème pointeur contient trois niveaux d'indirection. il pointe vers un tableau de blocs qui pointent eux même sur un tableau de blocs qui pointent eux même sur un tableau de blocs qui pointent vres les blocs de données.

En EXT, le répertoire racine correspond toujours à l'inode numéro 2.

Voir questions liens soft/hard pour les liens.

Détaillez comment l'OS permet de découper un disque en plusieurs partitions (primaire, logique, étendue). Quelles sont les limites de cette technique ? Comment évolue-t-elle ? Quels sont les outils utilisés pour gérer ces partition (mkfs, fdisk, mount, unmount, df, /dev, . . .) `JE SUIS PAS TROP SUR` Le MBR contient une table des partitions permettant d'avoir 4 partitions. Les partitions logique peuvent être divisées en partitions étendues.

Questions secondaires

(FAT) Détaillez comment l'OS retrouve un fichier, ajoute des données à ce fichier, efface ce fichier Exemple pour retrouver un fichier : int h = open("/home/user/f1", O_RDONLY);

- En FAT16 : Le cluster 0 contient la racine. Dans la liste des descripteurs présent dans le répertoire racine, l'OS cherche un descripteur avec le nom "user". Si il le trouve, il cherche le descripteur avec le nom "f1" dans user. Dans le descripteur de f1, on a le premier cluster du fichier ainsi que sa taille. Les clusters suivant sont chainés et sont retrouvés grâce à la FAT.
- En FAT32 : Même principe qu'en FAT16, sauf que le répertoire racine est retrouvé grâce au MBR.

Supprimer un fichier : On met 0 dans la FAT aux clusters correspondant au fichier.

Ecrire dans un fichier : Il est très simple d'écrire dans un fichier, il suffit de changer le chainage en fat si on utilise de nouveaux clusters.

(FAT) Détaillez les situations d'incohérence et montrer comment l'OS récupère cette situation. Lors d'un arrêt brutal du système, la FAT peut comporter des erreurs par rapport au contenu des clusters. Le système analyse le disque :

- Si la suite d'un fichier est un cluster vide, on remplace l'adresse par -1.
- Si on trouve un cluster qui contient des données mais ne semble appartenir à aucun fichier, et qu'il n'est pas libre. L'OS dépose alors le fragment de fichier dans le dossier lost+found (il y crée un fichier avec l'adresse du premier cluster).

(EXT2) Détaillez comment l'OS retrouve un fichier, ajoute des données à ce fichier, efface ce fichier. handle = open("/usr/home/d", O_RDONLY);

- / (root) == inode 2 => Numero de bloc 28
- bloc 28 => contient l'entrée "usr, inode 35"
- usr/ == inode 35 => Numero de bloc 64
- bloc 64 => contient l'entrée "home, inode 11"
- home/ == inode 11 => Numero de bloc 77
- bloc 77 => contient l'entrée "d, inode 45"
- d == inode 45, l'ensemble des blocs appartiennent au fichier "d".

(EXT2) Détaillez les situations d'incohérence et montrer comment l'OS récupère cette situation

(EXT2) Détaillez le contenu d'un super-bloc et l'utilité des champs qui s'y trouvent

(EXT2) Détaillez le contenu d'un inode et l'utilité des champs qui s'y trouvent

(EXT2) Détaillez les appels système qui permettent d'utiliser le système de fichier (open, read, write, close, dir, dup, lseek, stat, ...) et comment l'OS implémente ces appels système (handle, TDFO, ...)

```
int creat(const char * pathname, mode_t mode);
int open(const char * pathname, int flags);
int open(const char * pathname, int flags, mode_t mode);
int close(int fd);
```

`open` retourne un descripteur de fichier, c'est à dire une entrée dans la table des handle du process pointant vers la table des fichiers ouvert (TDFO). Un descripteur de fichier reste ouvert par défaut après une fonction `exec`car elle n'écrase pas la table des handle du process.

`creat` retourne également un descripteur de fichier, `open` peut également créer des fichiers si le flag `O_CREAT` est spécifié.

`close` ferme un descripteur de fichier, son entrée dans la table des handle. Si le descripteur spécifié en paramètre est le dernier descripteur correspondant à un fichier ouvert dans la TDFO, alors les ressources sont libérées dans cette dernière. close retourne 0 si il réussit, -1 sinon.

`open` créé une nouvelle entrée dans la table des descripteur des fichiers ouverts (`TDFO`) si il n'est pas déjà ouvert, sinon il partage cette entrée. Cette entrée comprend l'offset et le statut du fichier

Flags de open :

- `O_RDONLY` : lecture seulement
- `O_WRONLY` : écriture seulement
- `O_CREAT` : création
- `O_RDWR` : lecture et écriture

- `O_APPEND` : append : Avant chaque écriture : l'offset est positionné à la fin du fichier comme si on utilisait un lseek.
- `O_TRUNC` : Le fichier est "tronqué" (le contenu est supprimé).

Il est évident que pour `O_APPEND` et `O_TRUNC`, il faut un flag en écriture. L'effet de `O_RDONLY | O_TRUNC` est indéfini et varie selon les implémentations, mais a de grande chance d'être tronqué.

On peut utiliser plusieurs flags en les séparants avec le séparateur de flags "|", le `ou binaire`.

Dans le cas d'une création, il faut spécifier un mode. mode est ignoré si `O_CREAT` n'est pas présent. Le mode correspond aux droits du fichier une fois créé.

```
#include <unistd.h>
ssize_t read(int fd, void * buffer, size_t buffer_size);
```

- `fd` correspond au descripteur de fichier retourné par open/creat, dans la table des handles.
- `buffer` correspond à la zone mémoire ou iront les données lues.
- `buffer_size` correspond au nombre d'octet qui doivent être lus à partir de l'offset du fichier spécifié dans la TDFO. Si l'offset vaut `E.O.F.`, rien n'est lu et read() retourne 0.
- read retourne le nombre de bytes lus ou 0 si la fin du fichier est atteinte. L'offset du fichier est avancé par ce nombre.

```
#include <unistd.h>
ssize_t write(int fd, const void * buffer, size_t buffer_size);
```

- `fd` correspond au descripteur retourné par open/creat.
- `buffer` correspond à la zone mémoire contenant les données à écrire dans le fichier.
- `buffer_size` correspond au nombre d'octet à écrire.
- write retourne le nombre de bytes écrit dans le fichier. Ce nombre peut être plus petit que `buffer_size` si la place est insuffisante sur le disque dur. write retourne -1 si erreur.
- write écrit à l'offset du fichier. Cet offset est incrémenté par le nombre de bytes écrit.

```c
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

`lseek` a pour but de repositionner l'offset du fichier ouvert associé au descripteur de fichier fd spécifié en paramètre.

- offset correspond au décalage par rapport à l'offset courant.
- whence correspond à la position de départ du décalage.
  - `SEEK_SET` : correspond au début du fichier.
  - `SEEK_CUR` : correspond à la position actuelle de l'offset.
  - `SEEK_END` : corresond à la fin du fichier.
- lseek retourne la nouvelle position de l'offset par rapport au début du fichier. Si il y a erreur, -1 est retourné.

```c
int position_actuelle = lseek(fd, 0, SEEK_CUR); // Connaitre l'offset de la TDFO
```

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int stat(const char * path, struct stat * buf);
int fstat(int fd, struct stat * buf);
int lstat(const char * path, struct stat * buf);
```

```c
struct stat
{
    dev_t st_dev; // ID of device containing file
    ino_t st_ino; // numéro de l'inodeA
    mode_t st_mode; // les permissions
    nlink_t st_nlink; // Nombre de liens hard
    uid_t st_uid; // user ID du propriétaire
    gid_t st_gid; // group ID du propriétaire
    // PAS FINI
}
```

(EXT2) Détaillez comment l'OS mémorise les liens à l'aide d'exemple (soft, hard)

(EXT2) Détaillez la notion de fichier creux à l'aide d'un exemple (création, taille, occupation du disque) Exemple : On écrit dans un fichier "1" à la position 1 et "2" à la position 100000. Lors d'une lecture, le système retournera des 0 binaires, mais il ne stocke pas sur le disque. Les blocs qui contiennent des 0 ont l'adresse 0 dans l'inode (aucun bloc réservé). Dans ce cas le fichier n'occuperait que 3 blocs sur le disque : celui qui contient le "1", celui qui contient le "2" et le bloc pointé par la 11ème adresse. Alors que normalement un tel fichier devrait occuper 100 blocs. Mais sa taille est bien de 100000 * 4 = 400000 Kb. Mais son disk usage est de 3.

PS : On se déplace dans un fichier avec un lseek(), expliqué dans une autre question.

(EXT-EXT2) Détaillez la structure d'une partition formatée en EXT et EXT2. Détaillez les avantages d'EXT2 et l'implémentation de ces avantages. Commen EXT2 a t-il évolué ensuite ?