Système d'Exploitation
Travaux Pratiques (6), Licence 2 Informatique
Gestion de Fichiers

L'objectif de ce TP est d'offrir des fonctions pour un système de fichiers basé sur le mécanisme de FAT (File Allocation Table). Une FAT est une structure de données permettant de gérer à la fois l'espace libre et l'espace alloué. Il s'agit d'un tableau avec une entrée par bloc. Dans un répertoire, on trouve pour chaque fichier le numéro du premier bloc occupé par le fichier. L'entrée correspondante de la FAT à ce bloc contient l'adresse du 2ème bloc occupé, etc… Le dernier contient une valeur spéciale `FIN_FICHIER` (égale à `-1`). Les blocs vides sont à 0.

Nous considérons un disque dont la taille des blocs (secteurs) est de **128** octets. Le répertoire et la FAT sont de taille fixe. Le répertoire constitué de **16 entrées** occupe le bloc **0** et **1** du disque tandis que la FAT possède **128 entrées** (chaque entrée occupe **2** octets) et se trouve dans les blocs **2** et **3** du disque.

Il n'y a qu'un seul répertoire qui ne supporte pas d'arborescence. Chaque entrée du répertoire (**16 octets**) possède les champs suivants :

- **`del_flag`** (1 octet) : 0 indique que l'entrée est libre ; 1 indique que l'entrée est occupée.
- **`name`** (9 octets) : nom du fichier (8 caractères au maximum + '**`\0`**') ;
- **`first_bloc`** (2 octets) : premier bloc du fichier.
- **`last_bloc`** (2 octets) : dernier bloc du fichier.
- **`size`** (2 octets) : taille du fichier.

Le fichier **`fat.h`**, donné en annexe, définie la structure du répertoire ainsi que plusieurs constantes et prototypes de fonctions.

Le pointeur ***`short* pt_FAT`*** pointe vers le début de la FAT et le pointeur ***`struct ent_dir* pt_DIR`*** pointe vers le début du répertoire.

Nous vous offrons les fonctions suivantes :

- ***`void open_FS()`*** : initialise le système de fichier y compris les pointeurs ***`pt_FAT`*** et ***`pt_DIR`*** ci-dessus décrits.
- ***`void close_FS()`*** : libère les ressources allouées par la fonction d'initialisation.
- ***`int read_sector(short num_sect, char*buffer)`*** : rend le contenu du bloc de numéro ***num_sect*** dans le tampon ***buffer***. Renvoi 0 en cas de succès ou -1 en cas d'erreur de lecture depuis le disque.
- ***`int write_sector(short num_sect, char*buffer)`*** : écrit 128 octets du tampon **`buffer`** dans le bloc de numéro **`num_sect`** . Renvoi 0 en cas de succès ou -1 en cas d'écriture du disque.
- ***`int write_DIR_FAT_sectors()`*** : écrit sur le disque les secteurs contenant la FAT et répertoire (secteurs 0 à 3). Renvoi 0 en cas de succès ou -1 en cas d'erreur d'accès au disque.

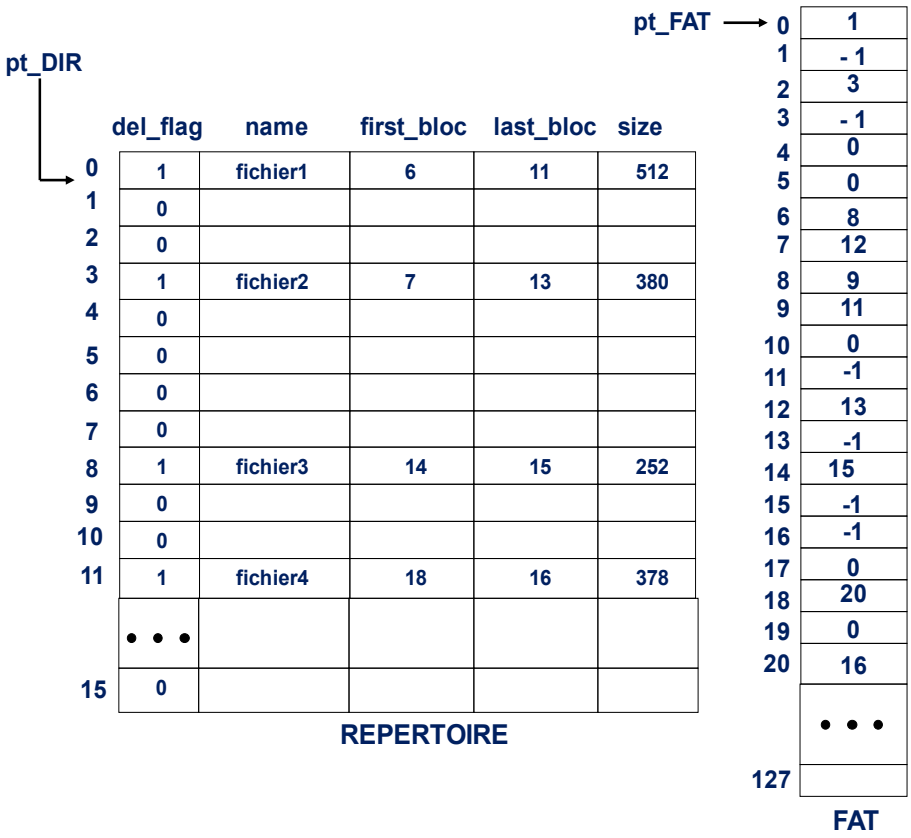
Observation : Le disque est simulé en utilisant le fichier **`disque_image`** qui doit se trouver dans le répertoire courant. Vous pouvez regarder le contenu de ce fichier en utilisant la commande :

```
hexdump -C disque_image
```

Le fichier **`disque_image.bak`** est une copie de sécurité du fichier **`disque_image`** original.

Au début, nous considérons que le disque contient les 4 fichiers suivants :

- **`fichier1`** : size = 512 ; possède 4 blocs : **`6 8 9 11`**
- **`fichier2`** : size = 380 ; possède 3 blocs : **`7 12 13`**
- **`fichier3`** : size = 252 ; possède 2 blocs : **`14 15`**
- **`fichier4`** : size = 378 ; possède 3 blocs : **`18 20 16`**



La figure ci-dessus montre le contenu du répertoire et de la FAT.

Les contenus des fichiers sont:

- **`fichier1`** : bloc 6 = 128 fois ‘A’; bloc 8 = 128 fois ‘B’; bloc 9 = 128 fois ‘C’; bloc 11 = 128 fois ‘D’;
- **`fichier2`** : bloc 7 = 128 fois ‘1’; bloc 12 = 128 fois ‘2’; bloc 13 = 124 fois ‘3’;
- **`fichier3`** : bloc 14 = 128 fois ‘a’; bloc 15= 124 fois ‘b’;
- **`fichier4`** : bloc 18 = 128 fois ‘*’; bloc 20= 128 fois ‘&’; bloc 16 = 122 fois ‘ #’;

Les squelettes des fonctions que vous devez programmer se trouvent dans le fichier **`func_FS_FAT.c`**. Vous y trouverez aussi la fonction ***`int file_found(char *file)`***, qui permet de vérifier si un fichier existe et la fonction ***`void list_fat()`***, qui affiche la liste des blocs alloués de la FAT. Ces deux fonctions donnent un exemple de comment parcourir les entrées du répertoire et de la FAT respectivement.

```
int file_found (char * file ) {
  int i;
  struct ent_dir * pt = pt_DIR;

  for (i=0; i< NB_DIR; i++) {
    if ((pt->del_flag) && (!strcmp (pt->name, file)))
      return 0;
    pt++;
  }
  /* fichier n'existe pas */
  return 1;
}

void list_fat () {
  int i;
  short *pt = pt_FAT;
  for (i=0; i < NB_ENT_FAT; i++) {
    if (*pt)
      printf ("%d ",i);
    pt++;
  }
  printf ("\n");
}
```

1. Création Environnement

Recopiez depuis la page du cours sur Celene le fichier compressé `TME_FS_FAT.zip` sur votre bureau Windows.

À partir de la VM Linux, recopiez depuis la page du cours sur Celene le fichier compressé `TME_FS_FAT.zip` dans votre répertoire personnel :

Décompressez le fichier : **`$ unzip TME_FS_FAT.zip`** pour obtenir les fichiers sources et le `Makefile`.

1.1.

Quelle est la taille maximale d'un fichier ?

2. Listage du répertoire

2.1

Complétez la fonction ***void list_dir()*** du fichier **func_FS_FAT.c** qui rend la liste des fichiers du répertoire. La fonction doit afficher les noms des fichiers et leur taille ainsi que le nombre total de fichiers du répertoire.

Testez la fonction avec:

```
make test_dir
./test_dir
```

2.2

Modifiez la fonction ***void list_dir()*** de la question précédente afin d'afficher aussi les blocs de données de chaque fichier.

3. Changement du nom d'un fichier

3.1

Complétez le code de la fonction ***int mv_file(char * file1, char*file2)*** qui permet de changer le nom du fichier ***file1*** par ***file2***. La fonction renvoi ***0*** en cas de succès et ***-1*** si le fichier ***file1*** n'existe pas ou erreur d'écriture sur disque.

Testez la fonction avec:

```
make test_mv_file
./test_mv_file fichier4  fichier6
./test_mv_file fichier5  fichier7
```

Observations :

1. Après avoir changé le nom fichier dans le répertoire en mémoire, il faut sauvegarder le répertoire sur disque en utilisant la fonction ***write_DIR_FAT_sectors()***.
2. ***fichier4*** existe et doit être renommé vers ***fichier6***, mais le ***fichier5*** n'existe pas.

4. Affichage du contenu d'un fichier

4.1

Complétez la fonction ***int cat_file(char * file)*** qui affiche le contenu du fichier ***file***. La fonction renvoi ***0*** en cas de succès et ***-1*** si le fichier ***file*** n'existe pas ou erreur de lecture.

Testez la fonction avec:

```
make test_cat_file
./test_cat_file fichier2
./test_cat_file fichier5
```

Observations :

1. Utilisez la fonction ***read_sector (short num_sect, char* buffer)*** pour lire un bloc du disque. Cette fonction renvoie un bloc dont la taille est de 128 caractères. Cependant, il se peut que le denier bloc d'un fichier ne soit pas complètement rempli (voir taille du fichier).
2. *fichier2* existe, mais le *fichier5* n'existe pas.

5. Suppression d'un fichier

5.1

Programmez fonction ***int delete_file(char * file)*** qui supprime le fichier ***file***. La fonction renvoie ***0*** en cas de succès et ***-1*** si le fichier ***file*** n'existe pas ou erreur d'écriture sur disque. Les blocs de données du fichier doivent être libérés.

Testez la fonction avec:

```
make test_del_file
./test_del_file fichier3
./test_del_file fichier5
```

Observations :

1. Pour indiquer que le fichier a été effacé du répertoire, vous affectez simplement le champ ***del_flag*** de l'entrée du fichier à ***0***.
2. Après avoir supprimé le fichier dans le répertoire en mémoire, et libéré les blocs des données, il faut sauvegarder le répertoire et la FAT sur disque en utilisant la fonction ***write_DIR_FAT_sectors ( )***.
3. ***fichier3*** existe, mais le ***fichier5*** n'existe pas.

6. Création d'un fichier vide

6.1

Complétez le code de fonction ***int create_file(char * file)*** qui permet de créer un fichier vide dont le nom est ***file***. La fonction renvoie ***0*** en cas de succès et ***-1*** si le fichier ***file*** existe déjà, pas d'entrée libre ou erreur d'E/S.

Testez la fonction avec:

```
make test_create_file
./test_create_file fichier5
./test_create_file fichier2
```

Observations :

1. La taille du fichier doit être ***0*** et la valeur du premier bloc, ainsi que celle du dernier bloc doit être égale à ***FIN_FICHIER*** (***-1***).

2. Après avoir créé le fichier dans le répertoire en mémoire, il faut sauvegarder le répertoire sur disque en utilisant la fonction ***write_DIR_FAT_sectors()***.
3. Le fichier ***fichier5*** n'existe pas, mais ce n'est pas le cas pour ***fichier2***.

7. Allocation d'un bloc de donnée

7.1

Programmez la fonction ***int alloc_bloc()*** qui alloue un bloc de la FAT. Le numéro de ce bloc est renvoyé par la fonction et la valeur de ce bloc (qui était égale à ***0***), est remplacée par ***FIN_FICHIER*** (***-1***). La fonction renvoie le numéro du bloc alloué en cas de succès ou ***-1*** s'il n'y a plus de bloc libre.

Testez la fonction avec:

```
make test_list_FAT
./test_list_FAT
```

8. Ajout de données à la fin du fichier

8.1

Programmez la fonction ***int append_file(char* file, char buffer, short size)*** qui ajout ***size*** caractères du tampon ***buffer*** à la fin du fichier. Considérez que le paramètre ***size*** est multiple de ***128*** (***SIZE_SECTOR***). La fonction renvoie ***0*** en cas de succès et ***-1*** si le fichier ***file*** n'existe pas, s'il n'y a plus de bloc ou erreur d'écriture sur disque.

Testez la fonction avec:

```
make test_append_file
./test_append_file
```

Le programme ***test_append_file*** appel la fonction ***append_file*** en ajoutant ***256*** caractères à la fin du fichier ***fichier1*** dont la taille est ***512*** (multiple de ***128***) :

```
memset(data, '%', SIZE_SECTOR);
memset(data, '$', SIZE_SECTOR);
append ("fichier1", data, 2*SIZE_SECTOR);
```

Observations :

1. Utilisez la fonction ***write_sector(short num_sect, char*buffer)*** pour écrire un bloc sur disque.
2. Après avoir ajouté les blocs, il faut sauvegarder le répertoire et FAT sur disque en utilisant la fonction ***write_DIR_FAT_sectors ( )***.

8.2

Modifiez la fonction ***int append_file(char* file, char buffer, short size)*** de la question précédente en considérant que ***size*** n'est pas forcement multiple de ***128***. Dans ce cas, il se peut que le dernier bloc du fichier ne soit pas totalement rempli et qu'une partie ou tous les caractères de ***buffer*** soient ajoutés dans ce bloc.

Testez la fonction avec :

```
make test_append2_file
./test_append_file2 fichier2  Bonjour
```

Dans ce test, 4 octets ("***Bonj***") seront ajoutés dans le dernier bloc de ***fichier2*** (bloc ***13***) et 3 octets ("***our***") dans un nouveau bloc.

9. Exercices optionnels

9.1

Modifiez la fonction ***void list_dir()*** de l'exercice 2.2 pour qu'elle accepte un paramètre : ***void list_dir (char* file)***. Si ***file*** égale à '*', la fonction affiche les informations sur tous les fichiers du répertoire. Sinon elle affiche les informations du fichier dont le nom est ***file***. Elle indique toujours le nombre total de fichiers (***0*** s'il ne trouve pas ***file*** ou le répertoire est vide).

9.2

Programmez la fonction ***struct ent_dir* read_dir(stuct ent_dir* pt_ent)*** qui renvoie un pointeur vers la prochaine entrée du répertoire par rapport à l'entrée ***pt_ent***. Si la valeur de ***pt_ent*** est la dernière entrée ou ***NULL***, la fonction renvoie la première entrée du répertoire. La fonction doit renvoyer ***NULL*** si l'argument passé en paramètre n'est pas valide.

9.3

Modifiez la fonction ***append_file*** pour qu'au lieu d'accepter un nom de fichier comme premier argument, elle accepte un descripteur de fichier : ***int append_file(struct ent_dir*pt_dir, char buffer, short size)***.

Pour cela vous devez aussi programmez la fonction ***struct ent_dir* open_file (char *file)*** qui ouvre un fichier en renvoyant un pointer vers l'entrée dans le répertoire correspondant au fichier. Elle doit être appelée avant la fonction ***append_file***.

Vous devez programmez aussi la fonction ***close_file (struct ent_dir* file)*** responsable pour enregistrer la FAT et le répertoire sur disque (cela était fait avant dans la fonctions ***append_file***).

Annexe - fat.h

```
/* nombre d'entrees dans le repertoire */
#define NB_DIR 16

/* taille d'un  bloc (secteur) */
#define SIZE_SECTOR 128

/* nombre de secteurs occupés par la FAT */
#define NB_SECTOR_FAT 2

/* nombre d'entrees de FAT par bloc */
#define NB_FAT_ENT_PER_SECTOR 64

/* nombre total d'entrees de la FAT */
#define NB_ENT_FAT  NB_SECTOR_FAT*NB_FAT_ENT_PER_SECTOR

/* taille en octet de la FAT */
#define SIZE_FAT NB_FAT_ENT_PER_SECTOR*NB_SECTOR_FAT*sizeof(short)

/* taille en octet du repertoire */
#define SIZE_DIR  NB_DIR*sizeof(struct ent_dir)

/*FIN FICHIER : utilisé dans la FAT */
#define FIN_FICHIER -1

#define DISC "disque_sim"


/* pointeur debut de la FAT */
extern short* pt_FAT;

/* pointeur debut du repertoire*/
extern struct ent_dir* pt_DIR;


/* entree d'un repertoire */
struct ent_dir{
  char del_flag; /* 0: entree libre ; 1 : entree occupe */
```

```
  char name[9]; /* nom du fichier : 8 caracteres + \0 */
  short first_bloc; /* premier bloc du fichier */
  short last_bloc; /* dernier bloc du fichier */
  short size; /* taille */
};

void open_FS ( );
void close_FS ( );
int read_sector (short, char*);
int write_sector (short, char* );
int write_DIR_FAT_sectors ( );
void list_fat ();
int file_found (char* );
void list_dir ( );
int cat_file (char*);
int mv_file (char*, char*);
int delete_file (char*);
int create_file (char*);
short alloc_bloc ();
int append_file (char*, char *, short);
struct ent_dir * read_dir (struct ent_dir*);
```