TD8 : Un mini SGF (système de gestion de fichiers)

On se propose lors de ce TD de programmer les algorithmes du SGF qui gèrent l'implantation physique des fichiers sur disque. Cette tâche étant particulièrement ardue nous allons admettre les simplifications suivantes :

Lecture et écriture de blocs. Les entrées/sorties seront gérées de manière synchrone par le biais des fonctions :

```
typedef char BLOCK[ BLOCK_SIZE ];
void read_block (int n, BLOCK* b);
void write_block (int n, BLOCK* b);
```

Descripteur de fichier (i-node). Un fichier est complètement décrit par un *i-node* qui est aussi un bloc. Un *i-node* sur disque a la structure suivante :

```
typedef struct INODE     /* INODE: Descripteur de fichiers   */
    {                    /* -------------------------------- */
    int  length;         /* taille du fichier (en octets)    */
    int  first;          /* adresse du premier bloc logique  */
    int  last;           /* adresse du dernier bloc logique  */
    }
    INODE;
```

Codage du répertoire unique. Les couples (*nom-de-fichier*, *adr-inode*) sont contenus dans un répertoire unique. Il n'existe donc pas de structure arborescente. Ce répertoire est composé d'un ensemble de blocs. Un couple a la structure suivante :

```
typedef struct DIR_ENTRY        /* Une entrée de répertoire         */
    {                           /* -------------------------------- */
    char nom [LONG_FILENAME];   /* nom du fichier                   */
    int  inode;                 /* adresse du descripteur (i-node)  */
    }
    DIR_ENTRY;
```

et un bloc contenant plusieurs couples est défini comme suit

```
#define BLOCK_DIR_SIZE     (BLOCK_SIZE / sizeof(DIR_ENTRY))
typedef DIR_ENTRY          BLOCK_DIR [ BLOCK_DIR_SIZE ];
```

Définition du Super Bloc. Le premier bloc du disque (d'adresse 0) est appelé le bloc de définition (ou *Super Bloc*). Il a la structure suivante :

```
typedef struct SUPER_BLOCK      /* Bloc d'un <<super bloc>>          */
    {                           /* ------------------------------- */
    int  signature;             /* signature du système de fichiers */
    int  adr_dir;               /* adr du 1er bloc du répertoire    */
    }
    SUPER_BLOCK;
```

Chaînage des blocs. Le chaînage des blocs qui composent un fichier ou le répertoire est placé dans une table unique appelée la FAT (*File Allocation Table*). Cette table comporte autant d'entrées que de blocs sur le disque. Elle est stockée sur disque à partir du bloc 1. La valeur de chaque entrée est interpretée comme suit

$$\text{FAT}[n] = \begin{cases} \texttt{FAT\_FREE} & \text{le bloc } n \text{ est libre} \\ \texttt{FAT\_RESERVED} & \text{le bloc } n \text{ est réservé} \\ \texttt{FAT\_INODE} & \text{le bloc } n \text{ contient un } \textit{i-node} \\ \texttt{FAT\_EOF} & \text{le bloc } n \text{ est le dernier du fichier auquel il appartient} \\ m & \text{avec } m > 0, \text{ le suivant du bloc } n \text{ est le bloc } m \text{ dans le fichier} \\ & \text{auquels ils appartiennent} \end{cases}$$

Pour manipuler plus facilement la FAT, un module du S.G.F nous offre les fonctions suivantes :

```
int alloc_block (void);          /* rechercher un bloc libre  */
int get_fat (int n);             /* lire une entrée de la FAT */
void set_fat (int n, int val);   /* modifier la FAT           */
```

Utilisation d'un bloc typé. Pour terminer, on définit un bloc *typé* comme une *union* des quatre types de bloc vus précédemment.

```
typedef union {
    SUPER_BLOCK super;
    BLOCK_DIR   dir;
    INODE       inode;
    BLOCK       data;
    }
    TBLOCK;
```

Recherche dans le répertoire

Écrivez la fonction `find_inode` qui recherche l'adresse d'un *i-node* à partir d'un nom en parcourant les couples du répertoire (elle renvoie -1 en cas d'échec).

```
int find_inode(char* name);
```

Ouverture d'un fichier en lecture

Écrivez la fonction `sgf_open_read` qui ouvre un fichier en lecture à partir de son nom.

```
OFILE* sgf_open_read(char* name);
```

La structure `OFILE` stocke toutes les informations permettant l'accès à un fichier ouvert :

```
struct OFILE            /* "Un fichier ouvert"                  */
    {                   /* ------------------------------------ */
    int   length;       /* taille du fichier (en octets)        */
    int   first;        /* adresse du premier bloc logique      */
    int   last;         /* adresse du dernier bloc logique      */
    int   inode;        /* adresse de l'INODE (descripteur)     */
    int   ptr;          /* n° logique du prochain caractère     */
    int   mode;         /* READ_MODE ou WRITE_MODE              */
    BLOCK buffer;       /* buffer contenant le bloc courant     */
    };
```

Lire dans un fichier

– Écrivez la fonction `sgf_read_block` qui lit un bloc en accès direct à l'intérieur d'un fichier ouvert.

```
ERR sgf_read_block(OFILE* file, int nubloc);
```

– Écrivez la fonction `sgf_getc` qui lit le caractère suivant dans un fichier ouvert. Elle renvoie -1 en cas de fin de fichier.

```
int sgf_getc(OFILE* file);
```

TD9 : Un mini SGF, suite et fin

Ajout d'un couple dans le répertoire

```
int add_inode (const char* name, int inode)
```

Écrivez la fonction ci-dessus qui ajoute un nouveau couple au répertoire. Trois cas sont à étudier

- Si il existe déjà un couple de même nom, alors il suffit de changer l'adresse du descripteur et la fonction renvoie l'ancienne adresse présente dans le répertoire.
- Si un couple de même nom n'existe pas, mais qu'il y a des entrées libres dans le répertoire, alors il faut utiliser une de ces entrées. La fonction renvoie -1 dans ce cas.
- Finalement, si le nom n'est pas déjà utilisé et qu'il n'existe aucune entrée disponible, il faut ajouter un nouveau bloc et utiliser la première entrée de ce bloc. La fonction renvoie -1 dans ce cas de figure.

Ouverture d'un fichier en écriture

```
OFILE* sgf_open_write(const char* nom)
```

Écrivez la fonction ci-dessus qui ouvre un fichier en écriture à partir de son nom. Si le fichier existe déjà, il est détruit au moyen de la fonction

```
void sgf_remove(int inode)
```

Écrire dans un fichier

1. Écrivez la fonction qui ajoute un bloc (le tampon) à la fin d'un fichier ouvert. Lors de cet ajout, toutes les informations définissant le fichier sur disque sont mise à jour.

   ```
   ERR sgf_append_block(OFILE* f)
   ```

2. Écrivez la fonction qui écrit le caractère suivant dans un fichier ouvert. On part du principe qu'il existe toujours une place libre dans le tampon. Si, après le placement du caractère, le tampon est plein, il faut le sauver sur disque.

   ```
   ERR sgf_putc(OFILE* file, char  c)
   ```

3. Écrivez la fonction de fermeture qui prend soin de vider correctement le tampon sur disque.

   ```
   ERR sgf_close(OFILE* file)
   ```