

CHAPITRE 2

SYSTEME DE FICHIERS

2.1 INTRODUCTION

Un système de fichiers (file system ou filesystem en anglais) ou système de gestion de fichiers (SGF) est une façon de stocker les informations et de les organiser dans des fichiers sur des mémoires secondaires (disque dur, CD-ROM, clé USB, SSD, disquette, etc.). Une telle gestion des fichiers permet de traiter, de conserver des quantités importantes de données ainsi que de les partager entre plusieurs programmes informatiques. Il offre à l'utilisateur une vue abstraite sur ses données et permet de les localiser à partir d'un chemin d'accès.

La plupart des cartes SD et MMC (Cartes Multimédia), particulièrement celles de taille au-dessous de 2 GB, utilisent le FAT16 standard. La norme FAT32 peut être utilisée pour adresser une mémoire de taille entre 2 gigaoctets et 2 teraoctets. Ce chapitre fournit une méthode de lire et-ou écrire sur ces dispositifs de stockage par un microcontrôleur. Les données de ces dispositifs de stockage peuvent être lues par un PC et les données écrites par un PC peuvent être lues par un microcontrôleur. La plupart des systèmes d'exploitation (c'est-à-dire, Windows® XP) Supportent le FAT16 et les systèmes de fichiers FAT32.

2.2- Structure du système de fichier FAT :

2.2.1 Structure physique de la mémoire :

Dans le cas du système de fichiers FAT, on distingue trois unités de mesures :

- L'octet
- Le secteur, composé de plusieurs octets
- Le cluster, composé de plusieurs secteurs

Ainsi, il y a deux paramètres importants : le nombre d'octets par secteur et le nombre de secteur par cluster.

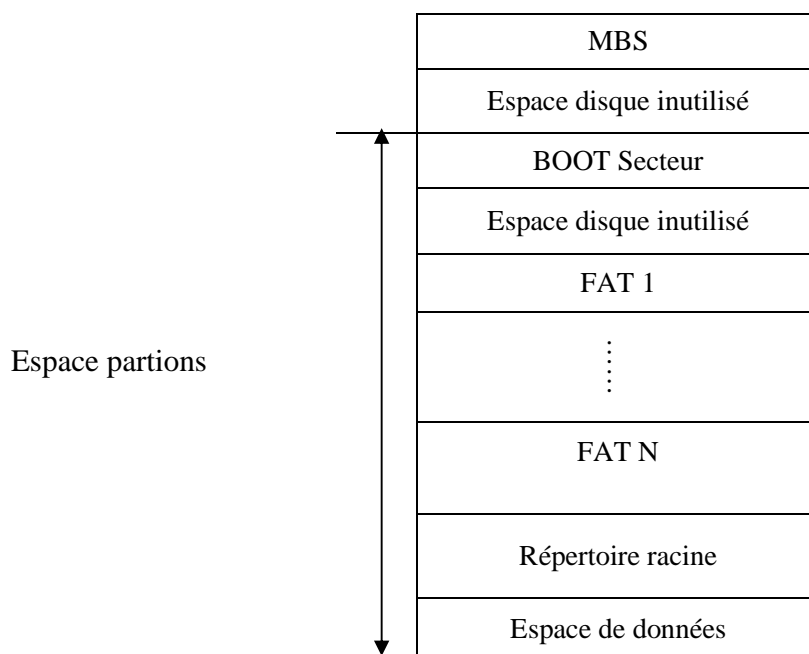
Ces paramètres sont fixés lors du formatage du système de fichier. Ils dépendent, entre autre, du système d'exploitation utilisé. Par exemple sur Linux, on pourra les renseigner individuellement grâce à la commande `mkdosfs` (voir man `mkdosfs`).

Sur Windows, on ne pourra pas les paramétrer aussi précisément.

Ces deux paramètres sont très importants afin de pouvoir se repérer dans la mémoire. Nous verrons dans la suite du rapport comment récupérer ces paramètres et comment les utiliser.

On remarquera que dans le cas de la FAT, on utilise le codage littleendian. Ainsi, lorsque l'on a deux octets qui se suivent, l'octet de poids fort est à droite.

2.2.2 Structure du système de fichier



La figure 2.1 : structure de disque

Le système de fichier se décompose en trois parties principales :

- Le secteur de boot (ou BPB, pour Boot Parameter Block)
- Les tables de FAT, qui sont une carte de la mémoire de masse
- La mémoire de masse, qui dans le cas de la FAT12/16 contient une partie spéciale, le root directory

2.2.2.1- Secteur de boot

Dans cette partie de la mémoire sont recensés plusieurs paramètres important concernant le système de fichier.

Le secteur de boot débute à l'adresse 0x00 et s'étend sur un secteur. Cependant, dans le cas de la FAT1 2/16 seuls les 62 premiers octets sont utilisés. Le reste du secteur n'est pas utilisé.

Offset	Description	Size
00h	Commande de Saut(Jump)	3 Octets
03h	OEM Nom du programme qui a formaté le disque	8 Octets
0bh	Nombre d'octet par secteur (512, 1024, 2048 ou 4096)	2 Octets
0dh	Nombre de secteur par cluster (1, 2, 4, 8, 16, 32, 64 ou 128)	1 Octets
0eh	Nombre de secteurs réservés en comptant le secteur de boot (32 par défaut pour FAT32, 1 par défaut pour FAT12/16)	2 Octets
10h	Nombre de FATs (2 par défaut)	1 Octets
11h	Nombre de header dans le root directory	2 Octets
13h	Nombre total de secteur 16-bits (0 pour FAT32)	2 Octets
15h	Type de disque (0xF8 = disque dur, 0xF0 = disquette)	1 Octets
16h	Nombre de secteur par FAT	2 Octets
18h	Nombre de secteur par piste	2 Octets
1ah	Nombre de Responsables	2 Octets
1ch	Secteurs cachés (0 par défaut si le disque n'est pas partitionné)	4 Octets
20h	Nombre de secteurs 32-bits (seulement si le nombre de secteurs 16-bits est égal à 0)	4 Octets
24h	Identifiant du disque (à partir de 0x00 pour disques amovibles, à partir de 0x80 pour les disques fixes)	1 Octets
25h	Reservé	1 Octets
26h	Signature (0x29 par défaut)	1 Octets
27h	Volume ID.	4 Octets
2bh	Nom du disque sur 11 caractères	11 Octets
36h	Type de systèmes de fichier (FAT, FAT12, FAT16)	8 Octets
1feh	Type de systèmes de fichier (FAT, FAT12, FAT16)	2 Octets

Tableau 2.1Secteur de boot

Les principales informations que l'on utilise sont surlignées. On ne peut pas se fier au champ indiquant le type de formatage, on doit calculer « à la main » le nombre total de cluster pour déterminer quel est le type de la FAT (voir fonction getFATtype).

On pourra également calculer les adresses de fin du secteur de boot, c'est-à-dire de début des tables de FAT, et de début du root directory.

2.2.2.2- Table de FAT

Généralement au nombre de 2, pour des raisons de redondance, les tables de FAT permettent de cartographier la mémoire de masse.

Chaque élément de cette table correspond à un cluster et dans le cas d'un fichier qui occupe plusieurs clusters, de connaître les différents clusters qui le composent.

Cette entrée serait 2 octets en cas de FAT16 et 4 octets dans le système de fichiers FAT32 et contient l'une des valeurs suivantes :

FAT16	FAT32	Description
0000h	0000h	Le groupe est disponible pour l'utilisation.
0001h	0001h	Le groupe est réservé.
0002-FFEFh	0000 0002-0FFF FFEFh	Points au groupe suivant dans le fichier(dossier).
FFF0-FFF6h	0FFF FFF0-0FFF FFF6h	Le groupe est réservé.
FFF7h	0FFF FFF7h	Le groupe est mauvais.
FFF8h-FFFFh	0FFF FFF8h-0FFF FFFFh	Le dernier groupe d'un fichier(dossier).

TABLEAU 2.2: VALEURS DE FAT

Chaque fichier y fait assigner au moins un cluster. Si cette taille de fichier est plus petite que la taille d'un groupe, l'entrée FAT pour ce groupe contiendra la dernière valeur de groupe indiquant qu'il n'y a plus de groupes assignés à ce fichier ; autrement, il contiendra la valeur du groupe suivant du fichier. En liant des groupes de cette façon, la FAT peut créer une chaîne de cluster pour contenir de plus grands fichiers et peut allouer des groupes non-séquentiels à un fichier (La figure2.2).

Il est important de noter que les valeurs qui indiqueraient 0 et 1 sont réservées pour indiquer des conditions spéciales. À cause de ceci, on étiquette le premier groupe dans la région de données comme le cluster 2. Les entrées FAT correspondant aux Groupes 0 et 1 contiennent le descripteur médiatique, suivi par des octets contenant la valeur, FFh.

"Premier Groupe" Valeurs
d'Entrées de Fichier

Valeur deFAT

Contenu des cluster
de données

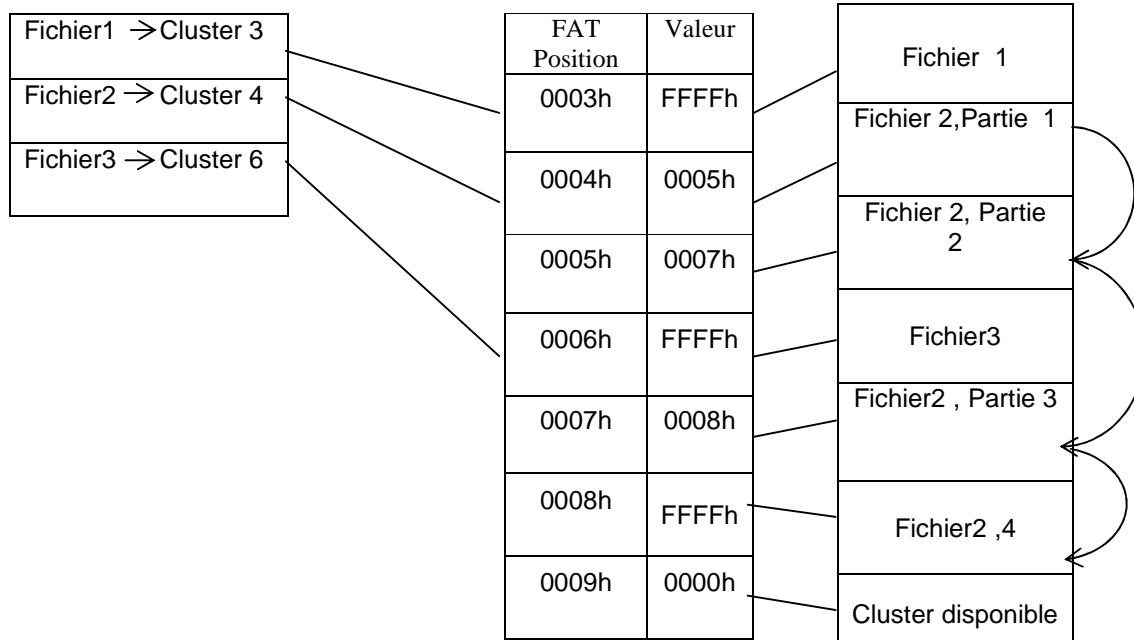


Figure 2.2

« Le Premier Groupe" trois entrées de fichier dans le répertoire racine indique le début de trois fichiers . Les valeurs FAT démontrent les liaisons entre les fichiers. Le fichier1 et 3 est plus petit que la taille d'un cluster ; de là, seulement un groupe leur est assigné. Les entrées dans la FAT qui correspondent à ces fichiers contiennent seulement le valeur Fin-de-fichier (EOF) .

Le fichier2.2 est plus grand que trois groupes, mais plus petit que quatre; de là, quatre groupes y sont assignés. Puisque trois groupes consécutifs n'étaient pas disponibles quand le Fichier 2.2 a été créé, des groupes non consécutifs y ont été assignés; ceci est appelé "la fragmentation". Chaque valeur dans La FAT pour le Fichier2.2 indique au cluster suivant dans le fichier. La dernière entrée dans la FAT pour le Fichier2 contient la valeur de Fin-de-fichier

2.2.2.3- Mémoire de masse :

Dans le cas de la FAT12 et de la FAT16, le début de la mémoire de masse est spécial. Il s'agit du root directory, le répertoire racine qui contient la description de tous les fichiers et sous-répertoires de la partition.

On appelle header la description de ces fichiers / sous-répertoires. Le nombre de header dans le root directory est stocké dans le BPB (offset 17). Chaque header est composé de 32 octets (tableau 2.3) :

Offset	Taille	Description		
0x00	8	Nom du fichier (rempli par des espaces), le premier octet peut avoir l'une des valeurs spéciales suivantes :		
		0x00	Entrée disponible, marque la fin du répertoire	
		0x05	Le nom de fichier commence en fait par le caractère ASCII 0xE5	
		0x2E	Entrée '.' ou '..'	
		0xE5	Entrée supprimée	
0x08	3	Extension du fichier (rempli par des espaces)		
0x0B	1	Attributs du fichier :		
		Bit	Masque	Description
		0	0x01	Lecture seule
		1	0x02	Fichier caché
		2	0x04	Fichier système
		3	0x08	Nom du volume
		4	0x10	Sous-répertoire
		5	0x20	Archive
		6	0x40	Device (utilisé en interne, jamais sur le disque)
		7	0x80	inutilisé
		La valeur 0x0F est utilisé pour désigné un nom de fichier long.		
0x0C	1	Réservé (utilisé par NT)		
0x0D	1	Heure de création : par unité de 10ms (0 à 199)		
0x0E	2	Heure de création:		
		Bits	Description	
		15-11	Heures(0-23)	
		10-5	Minutes (0-59)	
		4-0	Secondes (0-59)	

0x10	2	Date de création :	
		Bits	Description
		15-9	Année-1980(0=1980,127=2107)
		8-5	Mois (1 = Janvier, 12 = Décembre)
		4-0	Jour (1-31)
0x12	2	Date du dernier accès (voir offset 0x10)	
0x14	2	Index EA (utilisé par OS/2 et NT) pour FAT12/16, octets de poids fort du numéro du premier cluster pour FAT32	
0x16	2	Heure de dernière modification (voir offset 0x0E)	
0x18	2	Date de dernière modification (voir offset 0x10)	
0x1A	2	Numéro du premier cluster du fichier pour FAT12/16, octets de poids	

Tableau 2.3

2.3 Le contenu des répertoires

Les répertoires stockent sur 32 octets les informations sur les fichiers qu'ils contiennent

- nom de fichier
- taille du fichier
- date et heure de dernière modification du fichier

les attributs du fichier (lecture seul, archive ...)

- numéro du cluster auquel fichier commence les autres clusters constitutifs étant retrouvés

par la FAT

- le répertoire parent (pour les répertoires autre que la racine)

Les répertoires sont gérés comme des fichiers particuliers, qui contiennent une entrée avec les informations ci- dessus(tableau 2.4)

Descriptif	Taille
Partie principale du nom du fichier en texte	8
Extension du nom en texte	3
Attributs du fichier	1
Réservé	10
Heure de dernière écriture	2
Date de dernière écriture	2
Première unité d'allocation du fichier	2
Taille du fichier en octets	4

Tableau 2.4

Organisation des 32 octets d'un fichier dans le répertoire

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Accès
00002600	49	4F	20	20	20	20	20	20	53	59	53	27	00	00	00	00	IO SYS'
00002610	00	00	00	00	00	00	C0	B2	A5	26	02	00	B6	64	03	00À²¶&...¶d...
00002620	4D	53	44	4F	53	20	20	20	53	59	53	27	00	00	00	00	MSDOS SYS'
00002630	00	00	33	33	00	00	C0	B2	A5	26	B5	01	09	00	00	00	..33...À²¶&µ.....
00002640	43	4F	4D	4D	41	4E	44	20	43	4F	4D	20	00	00	00	00	COMMAND COM
00002650	00	00	00	00	00	00	C0	B2	A5	26	B6	01	82	76	01	00À²¶&¶...lv...
00002660	44	52	56	53	50	41	43	45	42	49	4E	27	00	00	00	00	DRVSPACEBIN'
00002670	00	00	00	00	00	00	C0	B2	A5	26	72	02	07	0E	01	00À²¶&r.....
00002680	42	4F	4F	54	43	44	20	20	20	20	20	08	00	00	00	00	BOOTCD
00002690	00	00	00	00	00	00	19	90	C3	2C	00	00	00	00	00	00IÃ.....

Répertoire avec des noms courts

Les noms de répertoire dans une série de chemin sont délimités par le caractère d'antislash (\). En dénotant un caractère d'antislash dans une série, un antislash supplémentaire doit être ajouté comme la partie d'un ordre d'évasion, comme l'antislash est utilisé par C pour commencer des ordres d'évasion.

- si le premier caractère d'une série de chemin est un antislash, le chemin sera assumé pour être spécifié quant au répertoire racine.
- si une série de chemin commence par un nom de répertoire, le chemin sera assumé pour être spécifié quant au courant travaillant l'annuaire(le répertoire).
- si un point (.) Ou point pointé (..) Est inclus dans le chemin comme un nom d'annuaire(de répertoire), le code opérera(exploitera) l'utilisation de ces entrées d'annuaire(de répertoire).

Par exemple, si l'utilisateur change le CWD à

".\TEST\..\TEST\..\\".", le courant travaillant l'annuaire(le répertoire) ne changerait pas d'où il a à l'origine commencé, supposant que l'annuaire(le répertoire), le TEST, existe dans l'annuaire(le répertoire) original.

Le tableau 2.5 fournit plus d'exemples des cordes de chemin.

Chemin	Signification
"\"	Le répertoire racine
"."	répertoire courant.
".."	répertoire précédent.
"ONE"	répertoire ONE dans l'annuaire courant.
".\ONE"	répertoire ONE dans l'annuaire courant.
"\ONE"	répertoire ONE dans le répertoire racine.
"..\ONE"	répertoire ONE dans l'annuaire précédent.
ONE\TWO"	répertoire TOW dans l'annuaire ONE dans l'annuaire courant.
"\ONE\TWO"	répertoire TOW dans l'annuaire ONE dans le répertoire racine
"ONE\..\TWO"	répertoire ONE et TOW dans l'annuaire courant (ce chemin pourrait être utilisé pour créer les annuaires inexistantes dans le même lieu utilisant FATmkdir fonction).

Tableau2.5 séries DE CHEMIN DE RÉPERTOIRE

2.4 Nom de fichier long

On ne gère que les noms de fichiers courts, c'est-à-dire de la forme 8+3 (8 caractères pour le nom et 3 pour l'extension). Cependant, on est obligé de créer un header supplémentaire car c'est ainsi que les systèmes d'exploitation repèrent les fichiers dans la partition. Il s'agit du système VFAT, qui s'applique à toutes les versions de FAT à partir de la FAT12.

A chaque header classique s'ajoute un ou plusieurs header spéciaux, repérés grâce à la valeur 0x0F à l'offset 0x0B.

Le nom du fichier est stocké dans le header, chaque caractère étant codé sur 2 octets (codage UTF-16). Comme on utilise uniquement des caractères ASCII, l'octet de poids fort sera toujours à 0.

OFFSET	TAILLE	Description
0x00	1	Numéro de séquence (voir ci-dessous)
0x01	10	Nom du fichier (5 caractères)
0x0B	1	Marqueur de header de nom de fichier long (toujours 0x0F)
0x0C	1	Type (toujours 0x00)
0x0D	1	Checksum du nom de fichier DOS (toujours 0x00)
0x0E	12	Nom du fichier (6 caractères)
0x1A	2	Numéro du premier cluster (toujours 0x0000)
0x1C	4	Nom du fichier (2 caractères)

Tableau :2.6

On peut ainsi stocker 13 caractères par header supplémentaire (en comptant l'extension).

Le numéro de séquence identifie cette suite de caractère par rapport au nom du fichier. On donne l'exemple suivant, pour coder le nom du fichier « Un nom de fichier vraiment long.txt » :

Numéro de séquence	Partie du nom stockée
0x43	« <u>long.txt</u> »
0x02	« hier vraiment »
0x01	« Un nom de fic »
	Entrée classique 8+3 (nom stocké : « UNNOM D~1.TXT »)

Tableau :2.6

Dans notre cas, on utilisera seulement un header supplémentaire puisque l'on se limite aux noms de 8 caractères (+3 pour l'extension).

2.5 Implémentation du système de fichiers :

Cette API est utiliser l'ensemble de fonctions permettant à un utilisateur de gérer simplement un système de fichier FAT 12, 16 ou 32. Les fonctionnalités demandées étaient les suivantes :

Fonctionnalités minimales:

- créer un fichier
- remplir un fichier (au choix : création ou erreur si inexistant)
- lire un fichier (par octet ou par bloc (taille max définie à la compilation))
- lister les fichiers présents sur l'unité de stockage(noms en 8.3, dans un ordre quelconque)
- fermeture propre de fichier (en faisant le moins d'accès possible à la carte)

Lors de l'utilisation de ces fonctionnalités nous avons dû garder à l'esprit les contraintes inhérentes à la programmation sur microcontrôleur (code utilisant peu de ressources), et une contrainte fixée par l'encadrant : l'utilisation de types de données portables.

Voici les prototypes des principales fonctions:

- intFSInit(void)
- intFSfclose(FSFILE *stream)
- FSFILE * FSfopen (const char * fileName, const char *mode)
- size_tFSfwrite(const void *ptr, size_t size, size_t n, FSFILE *stream)
- intFSchdir (char * path)
- intFindFirst (const char * fileName, unsigned intattr, SearchRec * rec);
- intFindNext (SearchRec * rec);

2.6 La structure de fichier

Le type file est une structure contenant les différents champs utiles à la gestion d'un fichier dans l'API : (pour plus d'information voir Microchip application help)

```
typedef struct {  
    char filename[FILE_NAME_SIZE_8P3 + 2];  
    unsigned char attributes;  
    unsigned long filesize;  
    unsigned long timestamp;  
    BOOL AsciiEncodingType;  
    unsigned short int * utf16LFNfound;  
    unsigned short int utf16LFNfoundLength;  
    unsigned int entry;  
    char searchname[FILE_NAME_SIZE_8P3 + 2];  
    unsigned char searchattr;  
    unsigned long cwdclus;  
    unsigned char initialized;  
} SearchRec;
```

2.7 Conclusion :

Le system FAT32a été utilisé avec l'unité de stockage USB avec l'application de Microchip.

Dans le chapitre 1 paragraphe 1.5.1 'L'architecture d'application ', le système de fichiers nous a facilité la lecture des fichiers contenus dans les unités de stockage.