Stockage Permanent

Michel Billaud (michel.billaud@u-bordeaux.fr, michel.billaud@laposte.net)

14 octobre 2020

Table des matières

L'ordinateur comporte des périphériques de stockage permanents (disques magnétiques, mémoire flash, etc.).

Pour le stockage et le manipulation des données enregistrées sur ces supports, le système d'exploitation implémente la notion de **système de fichiers**.

1 Système de fichiers

Abstraction selon laquelle (en général), pour le programmeur d'applications

- les données sont stockées dans des *fichiers*,
- ces fichiers sont contenus dans une structure arborescente (répertoires, dossiers,).

1.1 API des fichiers :

- ouvrir un fichier (`open`) : le système d'exploitation fournit au programme d'application un identifiant (*file descriptor*) qui correspond à un fichier.
- lire/écrire (`read`, `write`) des données (un certain nombre d'octets, à partir de la position courante
- connaître/changer la position (`lseek`),
- verrouiller le fichier (`flock`) pour s'assurer qu'aucun autre processus n'y accès en même temps,
- fermer (`close`) : mettre fin à cette correspondance.
- …

1.2 API sur l'arborescence

- supprimer (`unlink`) un fichier
- le déplacer (`rename`)
- créer un *lien* symbolique (`symlink`) ou matériel (`link`) sur un fichier existant
- parcourir un répertoire (`opendir`, etc.).
- consulter les *caractéristiques* d'un fichier/répertoire (`stat`, …),
- modifier les droits d'accès (`chmod`)
- …

1.3 Exemple

Programme en C (sous UNIX) qui montre ce qu'il y a dans le répertoire courant. Une barre de fraction est ajoutée après les noms de répertoires.

```c
#include <dirent.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <assert.h>

int main()
{
  DIR* dir = opendir(".");
  assert(dir != NULL);

  while (true) {
    struct dirent* entry = readdir(dir);
    if (entry == NULL) break;

    printf("%s", entry->d_name);

    // si c'est un répertoire, afficher un / à la fin
    struct stat entry_stat;
    int r = stat(entry->d_name, &entry_stat);
    assert(r == 0);
    if ((entry_stat.st_mode & S_IFMT) == S_IFDIR) {
```

```
      printf("/");
    }
    printf("\t");
  }

  printf("\n");
  return 0;
}
```

Note : le traitement des erreurs est sommaire. La macro `assert` arrête le programme si la condition est fausse.

Exemple d'exécution

```
$ ./dir
dir .vscode/    dir.c   Makefile    ../ ./
```

2 Stockage, aspects matériels

2.1 Disque magnétique

Dit "disque dur" (Hard disk)

2.1.1 Principe

- plateau (verre, céramique, …) couvert d'un enduit ferro-magnétique, en rotation rapide autour d'un axe,
- tête de lecture et enregistrement, portée par un bras mobile.

Quand le disque est en fonction, il tourne constamment (3600-15000 tours/minute). Pour une position donnée du bras, la tête de lecture parcourt donc (relativement) une **piste** circulaire découpée en secteurs (angulaires) sur lesquels sont enregistrés des blocs d'octets (typiquement 4K octets).

Un bloc est repéré par

- son numéro de piste
- un numéro de secteur

Pour accéder à un bloc, il faut

- déplacer le bras pour l'amener sur la piste voulue,
- attendre que la rotation fasse passer le bloc voulu sous la tête.

2.1.2 Plateaux

En réalité, l'axe d'unité de disque modernes entraîne plusieurs plateaux, et le bras mobile porte une tête pour chaque face.

Les blocs décrits pour un position du bras s'appellent un **cylindre**.

2.1.3 Temps d'accès, localité

Le temps d'accès est donc lié à des facteurs mécaniques. Il peut donc varier considérablement, selon qu'on doit ou non changer de piste/cylindre, et qu'on lit/écrit plusieurs blocs d'une piste consécutivement.

Quand un programme traite un fichier, il est assez probable qu'il le fasse séquentiellement. Pour ces raisons, un système d'exploitation essaiera (principe de localité) de "loger" les blocs d'un même fichier dans une même région du disque (avec des numéros de blocs voisins), pour éviter d'avoir à bouger le bras qui porte la tête de lecture.

Quand un programme demande à lire un bloc, on en profitera pour lire d'autres blocs du même cylindre, que l'on stockera dans des tampons mémoire , au cas où on en aura besoin ensuite (lectures anticipées).

Le temps d'accès moyen, de nos jours, est de l'ordre de 10 ms, soit environ 100 IOPS (Input-Output operations Per Second). Le débit est de l'ordre de 260 Mo/s.[1]

Prix grand public 2020 : environ 50 € pour 2 To. 500 € pour 12 To.

1. Ne pas confondre avec la vitesse de transfert (SATA, SCSI, …) entre le contrôleur du disque et la carte mère.

2.2 Stockage en mémoire flash (Solid State Drive)

Technologie complètement différente, utilise des mémoires à semi-conducteurs.

2.2.1 Cellules, pages et blocs

L'élément de base est une **cellule** qui mémorise (comme un condensateur) un niveau de tension, et peut représenter quelques bits (1 à 4 ?).

Une **page** regroupe des cellules, avec une capacité typique de 512, 2048 ou 4096 octets. Les pages sont regroupées en **blocs**[2] de 32-128 pages. Un circuit mémoire contient des blocs etc.

Les opérations de lecture/écriture se font par page. Mais l'écriture ne peut se faire que dans une page qui a été préalablement effacée, et l'effacement se fait par bloc complet.[3]

La vitesse de lecture/écriture est bien plus élevée (jusqu'à 3 Go/s) et le temps de latence beaucoup plus court, qui ne dépend pas de la localité des données. Il faut noter aussi que le nombre d'écritures par cellule est limité, ce qui réduit la durée de vie.

Les problématiques de gestion sont donc très différentes de celles des disques durs. On essaie de répartir l'usure sur les différents blocs de pages, en minimisant le nombre d'effacements. On tient donc une comptabilité des pages occupées, libres (effacées) et périmées (que l'on peut effacées) et défectueuses.

2.2.2 Comparaison avec disque magnétique

Avantages :

- beaucoup plus rapide,
- accès aléatoire en temps constant.

Inconvénients par rapport au disque magnétique :

- prix beaucoup plus élevé (en 2020 : 2To pour 500 €)
- durée de vie moindre

2.3 Ordonnancement des requêtes

Contexte : plusieurs processus demandent au système d'exploitation de faire des requêtes d'entrées-sorties sur un disque. L'unité de disque ne peut en traiter qu'une à la fois.

Le système d'exploitation a donc un stock de requête à faire traiter par le disque. Une**stratégie d'ordonnancement** indique comment choisir la prochaine requête à exécuter.

Dans le cas des disques magnétiques, la durée d'exécution d'une requête dépend fortement de la distance à parcourir par le bras qui porte les têtes de lectures. Plusieurs stratégies sont présentées ci-dessous.

Plusieurs critères pour juger d'une stratégie

- traiter un maximum de requêtes par seconde (utilisation efficace du disque),
- ne pas trop différer l'exécution des requêtes (équitabilité).

2.3.1 Stratégie FIFO

Consiste à exécuter les requêtes dans l'ordre où elles sont arrivées.

Exemple : 4 processus A, B, C, D qui lisent des données sur disque. A demandera des blocs consécutifs à partir de la piste 0, B à partir de la piste 100, C à partir de 10, D à partir de 110.

En les traitant dans cet ordre on aura des déplacements de tête de la piste 0 à 100 puis 10 puis 110, puis 1, 101, 11, 111, etc.

C'est une stratégie équitable, mais peu performante, le temps de déplacement du bras étant proportionnel au nombre de pistes traversées.

2. Attention aux conflits de terminologie, **bloc** désigne ici un bloc de pages.
3. Par envoi d'une tension plus élevée qui efface toute une région du circuit.

2.3.2 Stratégie “au plus proche d’abord”

Pour minimiser les mouvement du bras, une idée est de choisir la requête qui demandera le moins de déplacement. Après avoir lu la piste 0, on choisira la requête concernant la piste 10, qui est la plus proche.

Mais si, pendant cette lecture, le processus A a eu le temps d’émettre une autre requête, on reviendra à la piste 1. Puis 11, 2, 12, 3, etc. : les requêtes des processus C et D sont remises à plus tard, parce qu’elles sont “trop loin”.

Cette stratégie est donc plus performante en nombre d’entrées-sorties, mais moins équitable.

2.3.3 Stratégie de l’ascenseur

Dans cette stratégie, on traite les requêtes dans une direction (numéros de pistes montants, ou descendants), puis on change de direction, à la manière d’un ascenseur qui “ramasse” les passagers en montant, puis en descendant.

En partant de la piste 0, on passe à 10, puis 100, puis 110. On change alors de direction : 101, 21, 1. On change encore : 22, 102, 202, etc.

Ainsi on a un compromis qui réduit le nombre de mouvements du bras, tout en garantissant une certaine équitabilité.

2.3.4 Ordonnancement sur échéance (Deadline Scheduling)

Dans cette stratégie, on fixe un délai après lequel une requête est considérée comme devenue urgente[4].

On choisit alors

- la plus ancienne des requêtes urgentes, si il en existe
- sinon celle qui demande le moins de déplacement.

3 Architecture d’une unité de stockage

Interopérabilité des matériels, Communication uniformisée avec l’ordinateur, standards SATA, SCSI, …

L’unité de stockage comporte (en plus du dispositif matériel) un contrôleur avec une interface hôte, de la mémoire RAM (pour les tampons) et l’électronique de commande.

L’électronique de commande peut assurer des fonctions qui étaient autrefois prises en charge par le système d’exploitation (gestion des blocs défectueux, lectures anticipées, tampons, …), et qui sont donc décentralisées dans l’unité de disque.

3.1 Protocoles de communication avec l’ordinateur hôte

Nécessité d’avoir un standard pour interconnecter des matériels de constructeurs différents.

- ATA : Advanced Technology Attachment
- PATA : parallel ATA. Autrefois IDE Integrated Device Electronics. Obsolète.
- SATA : serial ATA
- SCSI : Small Computer System Interface
- SAS : Serial attached SCSI

https ://en.wikipedia.org/wiki/List_of_interface_bit_rates

techno	année	type	débit
PATA	1986	parallèle (16 bits)	8.3 MB/s
ATA 7 / UDMA 133	2002		133 MB/s
SATA 1.0	2003	série	150 MB/s
SATA 2.0	2004		300 MB/s
SATA 3.0	2008		600 MB/s
SCSI-1	1986	parallèle (8 bits)	5 MB/s
Fast SCSI	1994	parallèle (16 bits)	10 MB/s
Ultra 640 SCSI	2003	parallèle (16 bits)	640 MB/s
SAS-1	2004	série	300 MB/s
SAS-4	2017		2.4 GB/s

4. Valeurs typiques sous Linux : 500 ms pour une lecture, 5000 ms pour une écriture.

Le débit indiqué concerne la vitesse de communication entre le contrôleur et la machine hôte, qui peut être bien supérieure au débit de lecture/écriture sur disque.

Chacune de ces normes contient des commandes, pour

- identifier le matériel qui est connecté
- transmettre des requêtes d'entrée-sortie (lecture, écriture, formattage, etc).

3.2 Utilisation de mémoire dans le contrôleur.

Définition : Une mémoire cache ou antémémoire est une mémoire qui enregistre temporairement des copies de données provenant d'une source, afin de diminuer la durée d'un accès ultérieur (en lecture) à ces données.

La mémoire vive du contrôleur a plusieurs usages : mémoire tampon pour les écritures, et mémoire cache pour les lectures.

- Le résultat d'une lecture physique est mis dans un tampon du contrôleur avant d'être transmis à l'hôte.
- Si une demande de lecture concerne un bloc déjà lu et encore présent (en cache), le contrôleur peut répondre immédiatement sans lancer une lecture physique.
- Lorsque l'hôte envoie une requête d'écriture, le bloc à écrire est copié dans une zone tampon du contrôleur, en attente d'une écriture effective sur le disque physique, qui peut être donc différée. Le contrôleur peut les réordonner pour minimiser les déplacements.

La mémoire tampon améliore nettement les performances. Mais elle est en quantité limitée[5], le contrôleur la gère avec des algorithmes d'éviction (similarité avec la mémoire virtuelle).

3.3 Disques hybrides (SSHD, Solid State Hybrid Drive)

Les fabricants proposent divers compromis prix/performance/capacité. Depuis 2007, on trouve dans le commerce des unités de stockage hybrides, avec trois niveaux

- stockage sur disque magnétique (permanent, faible coût)
- stockage en mémoire flash (permanent, rapide)
- mémoire RAM utilisée comme cache (très rapide).

Le contrôleur fait en sorte que la mémoire flash contienne une copie des blocs les plus fréquemment utilisés. Il assure la synchronisation avec ces trois supports, et l'interface avec le système hôte.

L'intérêt est de regrouper ces deux types de stockage (SSD rapide, HD avec grande capacité) dans une seule unité physique, ce qui a un intérêt pour les portables où il n'y a de place que pour une unité (et où les disques durs à 5400 t/mn sont relativement lents).

Exemple : grand public pour environ 60 € (2020), Disque dur hybride Seagate FireCuda SSHD 500 Go (ST500LX025) Disque dur 2.5" 500 Go (5400 t/mn), 8 Go NAND, 128 Mo Mémoire.

En réalité cette technologie a connu un succès relatif, à cause de l'évolution des prix des composants : on trouve des unités SSD de même capacité pour quasiment le même prix, et de performances bien supérieures.

4 Virtualisation du stockage

Dans cette partie on regarde les techniques employées pour virtualiser le stockage, c'est-à-dire simuler un "disque virtuel" qui soit

- plus gros
- plus fiable (redondance)
- plus performant

(pas forcément les 3 à la fois !) , à partir de plusieurs disques physiques.

Depuis 1987, c'est ce qu'on appelle les technologies RAID (*Redundant Array of Independant Disks*).

5. La mémoire vive coûte beaucoup plus cher, au méga-octet, que l'espace sur disque.

4.1 Agrégation de données

À partir de plusieurs disques D1, D2, … (on parle de “grappe de disques”) on forme un disque virtuel DV plus gros en considérant

- que les premiers blocs de DV se trouvent sur D1
- les blocs suivants sont sur D2,
- etc.

Par exemple, avec un disque de 2 To et un autre de 1 To, on aura un disque virtuel de 3 To.

Évaluation :

- la taille est augmentée
- les opérations peuvent être effectuées en parallèle si elles concernent des blocs appartenant à des disques différents.

4.2 RAID 0 : agrégation par bande

Objectif : mieux répartir la charge.

On utilise des disques de même taille. Chaque disque est considéré comme une succession de “bandes” (strips)[6]. Le disque virtuel est constitué des premières bandes des disques, puis des secondes etc.

D1	D2	D3
b1	b2	b3
b4	b5	b6
b7	b8	b9
…	…	…

Évaluation

- la capacité du disque virtuel est la somme des capacités.
- temps d’accès amélioré : la charge de travail est répartie de façon homogène entre les disques physiques, qui travaillent en parallèle.

4.3 RAID 1 : miroir

Objectif : une redondance par duplication des données.

On utilise (au moins) deux disques de même taille. Chaque disque contient une copie des informations.

D1	D2	D3
b1	b1	b1
b2	b2	b2
b3	b3	b3
…	…	…

Fonctionnement

- Chaque écriture sur le disque virtuel se traduit par une écriture simultanée sur chaque disque physique.
- Les lectures sont réparties sur les disques.

Évaluation

- amélioration de la fiabilité : on peut récupérer les informations tant qu’il y reste au moins un disque en état de fonctionner.
- performances meilleures en lecture que les disques physiques (du fait de la répartition)

6. Typiquement, 16KB à 256KB.

Exercice : soit des disques ayant 5% de chances de tomber en panne dans l'année qui vient. Quelle est la probabilité de perdre ses données avec un système RAID 0 à 2 disques ? À 3 disques ? (On suppose des pannes indépendants).

4.4 Reconstruction, spare disks

Panne : Lorsqu'un système RAID détecte des anomalies sur un disque, le disque est mis hors service, et une notification est envoyée à l'administrateur de le remplacer. Pendant ce temps, le travail peut continuer, avec un disque en moins, et des conditions dégradées.

Remplacement : Quand l'administrateur remplace le disque défaillant, le contrôleur RAID "reconstruit" automatiquement le contenu de ce disque à partir des données qui sont sur les autres, et le remet en fonction.

Spare disks : dans les matériels professionnels, une "baie de disques RAID" contient aussi des disques de rechange, qui sont immédiatement mis en service dès qu'un disque tombe en panne. Ceci minimise la durée de fonctionnement en mode dégradé. Le rôle de l'administrateur est alors de remplacer dès que possible les disques défectueux par des disques neufs.

4.5 RAID 4 : agrégation par blocs avec parité

Objectif : redondance sans duplication.

Parité et détection d'erreur : Si on transmet un octet (ex 01101101), 1 bit supplémentaire permet de contrôler que la transmission s'est faite correctement. Par exemple, on peut convenir qu'il y aura en tout un nombre pair de bits à 1. On adjoindra donc un 1[7], et on transmettra 011011011.

À la réception, on détectera un problème si, parmi les 9 bits reçus, ceux qui sont à 1 sont en nombre impair.

Parité et correction : ce bit de parité permet aussi de corriger un bit manquant. Si on reçoit 01101x011, il est facile de voir que le bit x doit être à 1. On reconstitue l'information manquante.

C'est parce qu'avec un système de parité paire, le "ou exclusif" de tous les bits (y compris bit de parité) vaut 0. Et donc, la valeur d'un bit est le ou-exclusif de tous les autres.

Application au RAID Dans un système RAID 4 à 4 disques par exemple, on constituera une agrégation par blocs de 3 disques, et le dernier disque contiendra le ou-exclusif des bandes de même niveau. En cas de panne d'un disque, on pourra reconstituer son contenu.

Fonctionnement

- lecture : similaire au RAID 0
- pour chaque écriture, il faut recalculer les informations de parité pour les stocker sur le disque supplémentaire.

Évaluation

- capacité : avec N disques de capacité C, on a une capacité de (N-1)C.
- fiabilité : on peut encore accéder à toutes les données si un disque est en panne.
- répartition de la charge pour la lecture
- ralentissement de l'écriture, qui mobilise toujours le disque de parité.

4.6 RAID 5 : agrégation par blocs à parité répartie

C'est une variation du RAID 4, pour éviter le goulet d'étranglement que constitue un disque dédié à la parité.

Les blocs de parité sont répartis sur les disques.

D1	D2	D3	D4
b1	b2	b3	p(b1,b2,b3)
b4	b5	p(b4,b5,b6)	b6
b7	p(b7,b8,b9)	b8	b9
p(b10,b11,b12)	b10	b11	b12
b13	b14	b15	p(b12,b14,b15)
...			

7. On le calcule en faisant le "ou-exclusif" des bits de l'octet.

4.7 Exemple de combinaison : RAID 5+1

Pour améliorer à la fois les performances et la capacité, on réalise des combinaisons :

Par exemple, le RAID 5+1 (ou 51) consiste à utiliser plusieurs systèmes RAID 5, montés en "miroir" RAID 1.

Ceci utilise évidemment beaucoup de disques, mais la sécurité a un coût.

Exercice

- quel est le nombre minimum de disques pour réaliser un système RAID 51 ?
- combien faut-il de disques défaillants simultanément pour que le système soit en panne ?

5 Système de fichiers

5.1 Définition

Avant les ordinateurs, le terme **système de fichiers** désignait une méthode pour stocker et retrouver des documents en papier. Au début des années 60, il s'est étendu au stockage informatisé.

Définition : Un système de fichiers (*File System*) est une façon de stocker les informations et de les organiser dans des fichiers sur des mémoires secondaires (disque dur, CD-ROM, etc)

Il offre à l'utilisateur une vue abstraite sur ses données (fichiers, répertoires, …) et permet de les localiser à partir d'un chemin d'accès.

5.2 Couches

Un système de fichiers comporte au moins deux couches

- la **couche logique** qui interagit avec l'utilisateur, et fournit une interface de programmation (API) avec `open()`, `read()`, `write()`, `close()` etc.
- la **couche physique** qui s'occupe de la lecture et l'écriture des données sur le support physique. Elle fait appel pour cela aux pilotes (drivers) qui agissent sur le matériel.

5.3 Questions abordées

- représentation des métadonnées
- représentation des données du fichier
- représentation des fichiers et répertoires

5.4 Blocs

La plupart des périphériques de stockage sont "orientés blocs". C'est à dire que les informations contenues sur le périphérique sont accessibles par zones de même taille.

Physiquement, les informations sont enregistrées sur des **secteurs**, habituellement 512 octets (disques), 2Kio (CD et DVD), 4Ki (disques Advanced Format). Dans un secteur on trouve des "données utiles" et des codes correcteurs d'erreur.

Le terme "bloc" est utilisé de façon large. Dans le contexte des périphériques, il peut désigner la taille des informations échangées entre l'ordinateur et le contrôleur du disque, par exemple un bloc de 2Kio correspond à 4 enregistrements consécutifs sur disque. En principe consécutifs, parce que le contrôleur du disque, qui est intelligent, peut utiliser une table de "secteurs de remplacement" pour les zones défectueuses : le disque est alors vu comme une suite numérotée de blocs de 2kio.

Le bloc peut être aussi une unité de taille pour le système de fichiers : par exemple paramétré pour stocker les fichiers dans des zones d'une taille multiple de 4kio (qui correspondront à 2 blocs du disque, soit 8 secteurs).

5.5 Extent, domaine

Un *extent* (ou domaine) est une zone de stockage **contiguë** réservée pour un fichier sur le système de fichiers d'un ordinateur.

En allouant un extent à un fichier, les écritures qui suivent se font à proximité les unes des autres, ce qui limite la fragmentation qui est un problème pour les supports magnétiques dont le temps d'accès n'est pas uniforme.

5.6 Exemple simple : le système de fichiers de CP/M

http ://www.informit.com/articles/article.aspx ?p=25878&seqNum=3

CP/M est un système d'exploitation du début des 80's. Équipement typique : processeur 8 bits Intel 8080, 4Kio de RAM et un lecteur de disquette 8 pouces de capacité 160 Kio.

Le système de fichiers contient un seul répertoire (mais qui aurait besoin de plusieurs répertoires, sur 160 Ko ?), une table avec des éléments de 32 octets. Les fichiers sont alloués par blocs de 1ko (8 secteurs physiques de 128 octets).

Quand la disquette est "montée" , le système d'exploitation lit le répertoire et calcule une table des blocs occupés (160 bits) qui tient sur 23 octets.

Pour chaque entrée de la table on a (dans le désordre)

- le nom de fichier : 8 caractères pour le préfixe, 3 pour l'extension qui indique son type (idée reprise par MS-DOS)
- une table avec 16 numéros de **blocs**, ce qui limite la taille d'un fichier à 16K. Chaque numéro de bloc codé sur un octet
- le nombre de **secteurs** utilisés, sur un octet

Pour les fichiers de taille supérieure à 16Ko, il y a plusieurs entrées dans le répertoire, qui donnent donc les 16 premiers blocs, puis les 16 suivants, etc. Un champ supplémentaire indique l'ordre des entrées d'un même fichier.

5.7 Exemple simple : Le système de fichiers FAT

Système de fichiers conçu pour les disquettes en 1977 par par Bill Gates et Marc McDonald, utilisé dans QDOS, MS-DOS, et Windows jusqu'à Windows 9x. Trois variantes : FAT12, FAT16 et FAT32. Windows repose maintenant sur NTFS mais reconnaît toujours FAT. De nombreux appareils électroniques (appareils photos, téléphones, etc) l'utilisent.

Voir https://fr.wikipedia.org/wiki/File_Allocation_Table

Un disque FAT contient essentiellement

- une zone avec les blocs de données des fichiers (leur contenu) proprement dits
- une zone réservée (File Allocation Table) indiquant, pour chaque bloc de données, le numéro du bloc suivant.
- le répertoire racine

Un répertoire est une table qui contient, pour chacun de ses éléments (fichier ou répertoire) :

- son nom (et son extension), 8+3
- son taille, son type (fichier ou répertoire)
- un index donnant la position de son premier bloc

Les blocs sont des "clusters" (groupes de secteurs) dont la taille est choisie au formattage.

Pour des raisons de sécurité, la FAT est dupliquée.

Les versions :

- FAT 12 (utilisé sur les disquettes), les numéros de blocs sont sur 12 bits et les clusters vont de 512 octets à 4 Kio. La capacité maximale est donc de 4086 x 4 Kio, soit 16 Mio.
- En FAT 16, numéros sur 16 bits, soit 65536 clusters de 2 à 64 Kio. Capacité max = 6 Gio
- FAT 32 : 2^28 (pas 32…) clusters de 2 à 32 Ko. Capacité maximale : 16 Tio avec des secteurs de 4KB).

5.8 NTFS

New Technology FileSystem, développé par Microsoft Corporation pour Windows NT.

Apporte de nombreuses améliorations techniques par rapport à la FAT

- support amélioré des métadonnées,
- structures de données avancées (B+ trees) pour améliorer les performances et la fiabilité,
- listes de contrôles d'accès (ACL),
- journalisation (assure l'intégrité des données en cas d'arrêt brutal ou de problème matériel).

Voir par exemple https://fr.wikipedia.org/wiki/NTFS_(Microsoft)