

# Système d'exploitation

## Introduction

### L'ordinateur et ses composants

Un ordinateur se compose :

- ☐ d'une **mémoire centrale (RAM)** qui contient les programmes et données durant leur exécution/utilisation ;
- ☐ d'une **unité centrale de traitement (CPU)** qui exécute un programme qui se trouve en mémoire centrale ;
- ☐ d'**unités périphériques** qui permettent l'échange de données (avec l'utilisateur, une mémoire de masse...).

L'**unité centrale** est la partie essentielle de l'ordinateur. Elle **interprète** et **exécute** les **instructions**.

Le CPU est essentiellement composé d'une

- ☐ **Unité de Contrôle (UC)**, chargée d'extraire de la mémoire l'instruction, la décoder et l'exécuter (décodeur / séquenceur) ;
- ☐ **Unité Arithmétique et Logique (ALU)**, à laquelle l'UC peut faire appel.

Les principaux éléments de l'UC sont :

- ☐ le **compteur ordinal (IP – Instruction Pointer)** : **registre** qui contient l'**adresse de l'instruction** en mémoire à exécuter ;
- ☐ le **registre d'instruction (IR – Instruction Register)** qui contient l'**instruction** à traiter ;
- ☐ le **décodeur d'instruction** qui détermine l'**opération à effectuer** et les opérandes ;
- ☐ le **séquenceur** qui génère les **signaux** de commande à **envoyer** aux différents composants ;
- ☐ l'**horloge** qui permet la **synchronisation** grâce aux **impulsions** qu'elle émet.

La **mémoire centrale (RAM)** permet de conserver pendant un certain temps des informations binaires, par exemple des instructions et des données.

La RAM (**Random Acces Memory**) est une mémoire à « **adressage à accès aléatoire** », c'est-à-dire que chaque mot peut y être lu ou écrit directement et que le temps d'accès à un mot est **indépendant de son emplacement**. La mémoire vive est **volatile**, elle a besoin d'électricité pour garder son contenu.

La mémoire centrale est **subdivisée en mots** (unités adressables) et est reliée à l'unité centrale par trois types de bus :

- ☐ le **bus d'adresse**, qui contient l'adresse du mot ;
- ☐ le **bus de données**, qui contient le contenu de mot ;
- ☐ le **bus de commande**, qui communique si on lit ou si on écrit dans la RAM.

La RAM peut communiquer des données par ses 8 fils reliés au registre de données (bus interne). Elle peut envoyer  $2^n$  mots,  $n$  étant le nombre de fils.

Si le bit d'adresse dispose de 32 fils, il peut communiquer 4Gb  
( $2^{32} = 1024 * 1024 * 1024 * 4 = 4Gb$ )

## Fonctionnement du CPU

☐ **fetch** : lecture de l'instruction à traiter

1. IP communique l'adresse de la prochaine instruction à traiter au registre d'adresse (RAD) via le bus interne.
2. Le registre d'adresse envoie l'adresse dans le bus externe à destination de la RAM.
3. Le séquenceur envoie l'instruction de lecture via le bus externe de commande.
4. L'instruction de lecture arrive à la RAM par le fil de commande.
5. L'adresse de l'instruction arrive à la RAM par les 4 fils d'adresse.
6. Le contenu de l'emplacement mémoire lu est placé dans le tampon de la mémoire RAM.
7. Les données transitent par le bus de données jusqu'au registre de données (RDO).
8. Le registre de données envoie le flux de données au RI, et IP est incrémenté pour pointer sur l'adresse de la prochaine instruction à traiter.

L'incrémentation automatique d'IP permet une séquence d'instructions.

☐ **decode/execute** : décodage et exécution

## Interruptions

- ☐ **matérielles** (asynchrone) (entre système et OS) : action venant d'un périphérique, par exemple la souris ;
- ☐ **déroutements** (erreurs cpu) (idem) : exemple, une division par 0 ;
- ☐ **logicielles** (appels systèmes) (entre applications et OS) : accès à un périphérique par un programme.

Une interruption asynchrone se produit à intervalles irréguliers, contrairement à une interruption synchrone qui se produit périodiquement.

**Mode privilégié** : accès à toute la RAM et toutes les instructions

### Que fait le CPU lors d'une interruption ?

- ☐ Modifier IP avec l'adresse de traitement de l'interruption  
IP sera à restaurer plus tard, il faut prévoir sa sauvegarde
- ☐ Basculer en mode privilégié
- ☐ Exécute le code de l'interruption
- ☐ Restaure la valeur du mode et l'IP
- ☐ Reprend le traitement de l'instruction précédente

Le **mode privilégié** ne peut être lancé que par l'OS. Par souci de sécurité, les bouts de code exécutés lors d'une interruption sont des codes systèmes.

La notification d'interruption arrive au CPU par hardware.

La valeur d'IP est **modifiée par hardware** directement par le CPU.

En mono-programmation, l'**IRET** gère la restauration.

Sinon l'**ordonnanceur** s'en occupe, en multi-programmation.

Le **code des interruptions** se trouve dans le **vecteur des interruptions**. Il est constitué de **256 adresses de 4 bytes** chacune.

L'adresse de traitement de l'interruption **N** se trouve à l'adresse  **$4 * N$**  (car 4 bytes).

## 1. Sauvegarde

Stack Pointer (SP) : sommet de la pile (stack)

La pile va servir à garder en mémoire l'IP de l'instruction interrompue et le mode d'exécution.

CPU empile l'IP puis le mode

IRET dépile le mode puis l'IP

## 2. Exécution du code d'interruption

IP  $\leftarrow$  adresse traitement

mode  $\leftarrow$  privilégié

Le code d'interruption sauvegarde le contenu des registres sur la pile, traite les instructions qui le constituent, et restaure le contenu des registres.

## 3. Restauration du mode et de IP

IRET (Instruction Return) : restaure Mode & IP depuis la pile (en mono-programmation)

Mode  $\leftarrow$  Stack Pointer, IP  $\leftarrow$  Stack Pointer

en multi prog : IP  $\leftarrow$  adresse ordonnanceur

## Amorce - Chargeur / Loader

Un **chargeur** est un programme de l'OS qui charge en mémoire un programme depuis un périphérique vers la RAM.

Au démarrage de l'ordinateur, le CPU communique à IP une adresse en ROM (**Read Only Memory**) qui va permettre de lancer le BIOS (**Basic Input/Output System**).

Le **BIOS** permet à la carte mère d'effectuer des opérations très basiques au démarrage de l'ordinateur, par exemple la lecture d'un secteur sur un disque. Le BIOS charge le premier périphérique bootable qu'il trouve.

Le BIOS cherche un **périphérique bootable** au démarrage de l'ordinateur, puis charge en RAM les **512 premiers bytes** du périphérique.

L'adresse du premier byte est ensuite communiquée au registre IP.

**MBR** (Master Boot Record) : constitué des 512 premiers bytes d'un périphérique bootable, il est composé d'un **chargeur de démarrage** (446 bytes) et de **4 descripteurs de partition** de 16 bytes.

Les 2 derniers bytes servent à indiquer si le périphérique est **bootable** (**55AAh**).

## Émergence

Un **appel système** est un bout de code de l'OS qui réalise un **service** en mode **privilégié** pour un programme.

**INT** (Interrupt) : un appel système correspond à une interruption logicielle.

Le numéro d'interruption est différent sur les différents OS (80 pour Linux, 20 pour Windows).

EAX : numéro du service

EBX : numéro du fichier

ECX : adresse (en bytes)

EDX : compteur

**Monoprogrammation** : mode d'exploitation d'un ordinateur où un et un seul programme s'exécute en mémoire principale.

Le **canal** est un processeur spécialisé dans les entrées/sorties. Il permet un accès direct à la mémoire (**Direct Access Memory**) en même temps que le CPU. Une synchronisation entre les 2 permet d'éviter qu'ils travaillent tous les 2 en même temps au même endroit dans la RAM.

Le canal est relié aux bus, au CPU et à la RAM. Il permet, avec les interruptions, de **décharger le CPU** des opérations qui le ralentissent (entrées/sorties).

Un **appel système** est un appel d'un programme pour demander un **service** à l'OS. Il provoque toujours une interruption.

## Multi-programmation

Un **processus** est un programme chargé dans la RAM en attente de son exécution.

La **multi-programmation** consiste en l'exécution en alternance de programmes.

Problème de la multi-programmation : partage des ressources implique qu'il faut veiller à ce qu'un programme n'interfère pas avec un autre (écriture dans une zone mémoire non allouée au programme par exemple).

L'**ordonnanceur** se charge de sauvegarder dans la **table des processus** (stockée en RAM) les données relatives aux programmes (le **contexte**) entre 2 changements d'état (le mode, l'IP et les valeurs des différents registres).

Chaque programme stocké en RAM doit conserver son **contexte** entre 2 interruptions :

- ☐ sa valeur IP ;
- ☐ une valeur SP pour sa pile ;
- ☐ l'état des registres du CPU (AX, BX...) ;
- ☐ son mode de fonctionnement (privilegié ou non) ;
- ☐ son état (élu, prêt, bloqué) ;

Les processus changent d'état grâce à l'ordonnanceur :

- ☐ **prêt** : attend que le CPU l'exécute (suspendu en faveur d'un autre)
- ☐ **élu** : en cours de traitement
- ☐ **bloqué** : subit une interruption, par exemple une entrée/sortie

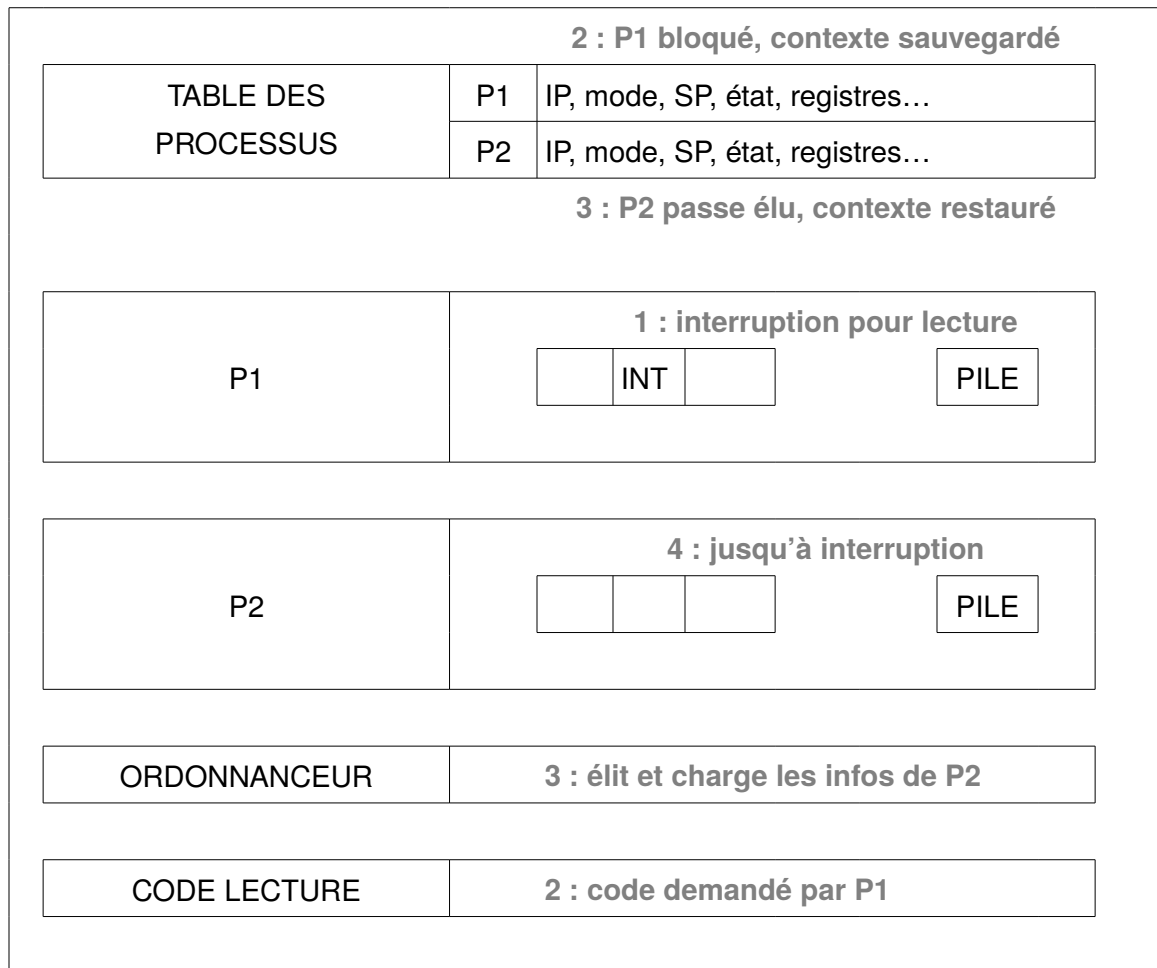
**Polling** : technique de programmation que les processus utilisent lorsqu'ils vérifient si une condition est vraie, comme une entrée utilisateur (CPU bloqué par l'utilisation du clavier en mode privilégié)

## **Time slicing**

Le time-slice est le temps maximum pendant lequel un programme peut tourner sans être interrompu en faveur d'un autre. Une fois la portion de temps écoulée, le système doit passer à un autre processus à traiter. Si le même processus tourne en permanence sans demander d'interruption, le processeur sera sous-utilisé.

Pour récupérer le contrôle, l'horloge du CPU provoque une interruption, ce qui permet à l'ordonnanceur de changer l'état du processus actuel à "prêt" et passer à "élu" un autre processus.

## Demande de lecture



- 1. Une interruption** sauvage apparaît ! L'interruption s'occupe de sauvegarder **IP** et le **mode** sur la **pile de P1**.  
 Le **mode** passe en **privilegié** et **IP** reçoit l'**adresse de l'interruption**.
- 2. P1** passe à l'état **bloqué** en attendant la **fin de la lecture**. L'**appel système** s'occupe de **sauvegarder le contexte de P1** (IP et mode depuis la pile, les registres...).  
 Il envoie la **commande** de lecture au **périphérique** et au processeur **canal**.  
 IP reçoit une nouvelle adresse de l'ordonnanceur.
- 3. L'ordonnanceur** change l'état de **P2** (seul processus prêt) en **élu**. Il charge les **registres** et le **mode de P2** depuis la **table**. **IP** reçoit l'**IP de P2** sauvegardée précédemment **dans la table**.
- 4. P2** s'exécute jusqu'à ce qu'il reçoive une interruption.  
 Cette interruption peut venir du canal qui a fini la lecture (P1 est donc "prêt" à être exécuté). Si c'est le cas, P2 est interrompu, il passe en "prêt" et l'ordonnanceur décide si P1 ou P2 doit être exécuté.



## Système de fichiers

Les systèmes de fichiers servent à :

- ❑ **stocker** des fichiers de manière **permanente** ;
- ❑ **nommer** des informations ;
- ❑ **partager** des informations.

sous forme de fichiers.

Un **disque** est une suite de **secteurs** (512 bytes) pouvant être lus et écrits.

Deux vues du système de fichiers :

- ❑ vue **utilisateur** : quel service ?

noms de fichiers, structure des données, type de fichier/répertoire, fichier texte/binaire, type d'accès séquentiel/aléatoire, attributs (métadonnées), opérations (appel système)...

create, open, read, write, seek, append, close, chmod...

- ❑ vue **système** : comment est-ce réalisé ?

## Mise en œuvre - Système

Il faut organiser la partition en un système de fichiers :

- ❑ **fichiers** : données et méta-données ;
- ❑ **répertoires** : structure des données, méta-données ;
- ❑ **métadonnées du système de fichiers** : fichiers remarquables, taille secteurs, localisation de la racine...

**Exemples de métadonnées** pour un fichier : droit, taille, propriétaire, date de création...

Les **rôles** d'un système de fichiers :

- ❑ **allouer l'espace** aux fichiers/répertoires ;
- ❑ **définir l'implémentation** des fichiers/répertoires ;
- ❑ **gérer l'espace disque** (espace libre, quotas...) ;
- ❑ **assurer fiabilité et performance** ;
- ❑ **assurer chiffrement et compactage** ;
- ❑ ...

## Allocation contiguë

Les données d'un fichier sont **consécutives** sur la partition.

Les lectures sont **rapides**, la tête de lecture ne faisant pas de mouvements inutiles.

Si les fichiers sont modifiés, les performances se dégradent.

La **taille du secteur** est une **métadonnée** du système.

Les **métadonnées** nécessaires à la **localisation** d'un fichier :

- ❑ le numéro du **secteur de début** ;
- ❑ la **longueur** du fichier, en bytes.

Un fichier démarre à une frontière de secteur.

Comment traduire la longueur en bytes d'un fichier en nombre de secteurs utilisés

$(\text{longueur en bytes} + 511) / 512$

Comment localiser le byte N ? ( $N < \text{longueur}$ )

- ❑ Quel est le numéro du secteur dans lequel N se trouve ?  
**numéro secteur = numéro début secteur + N DIV TS (taille secteur)**
- ❑ Quelle est la position de N, son décalage, dans le secteur ?  
**position secteur = N MOD TS (taille secteur)**

**Exemple :**

Un fichier démarre au **secteur 24** et contient **2060 bytes**.

- ❑ Ce fichier occupe :  $(2060 + 511) \text{ DIV } 512 = 5$  secteurs
- ❑ Le byte n°1055 se trouve :
  - au byte :  $(1055 \text{ MOD } 512) = 21$
  - au secteur :  $24 + 1055 \text{ DIV } 512 = 26$

**Avantages :**

- ❑ rapidité d'accès en lecture

**Inconvénients :**

- ❑ lenteur due à la fragmentation externe suite à des modifications  
fragmentation externe = fragmentation de l'espace libre

## Allocation par blocs

Les fichiers et l'espace disque sont **découpés en blocs de taille fixe**.

L'espace alloué aux fichiers n'est **pas forcément contigu**.

L'**ajout** ou la **suppression** de données est **plus aisée**.

Un **bloc** est un **nombre entier de secteurs** (cluster Windows).

Un **bloc** est une **unité d'allocation** et d'accès logique.

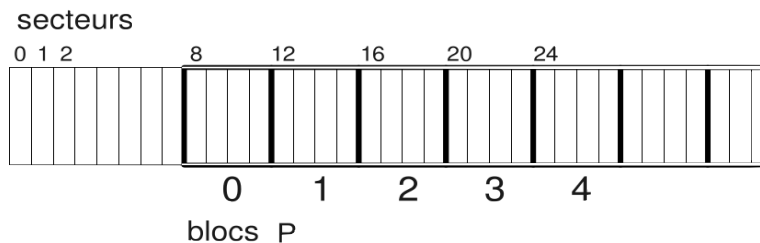
Les blocs sont numérotés.

L'adresse du bloc P est déterminée par :  $P * nb\_de\_secteurs\_par\_bloc$

**Exemple :**

Le bloc 3 commence au secteur :

$$8 + 3 * 4 = 20$$



1. Le bloc 0 commence au secteur 8.

2. Le bloc dont on cherche l'adresse est le bloc 3.

3. Un bloc est constitué de 4 secteurs.

Le byte N d'un fichier se trouve dans le bloc n°F

$$F = (N \text{ DIV } taille\_bloc)$$

Le début et la fin du fichier ne suffisent plus à le localiser : il faut **localiser chaque bloc** du fichier, avec des **blocs chaînés** et des **tables d'index**.

### Comment localiser le byte N ?

- Comprendre : Quelle est la position du byte N dans le secteur n°X ?

1. Trouver le *bloc du fichier F* contenant le byte N :

$$F = N \text{ DIV } \text{taille\_bloc}$$

2. Pour trouver le *bloc physique P* correspondant au *bloc du fichier F*, il faut parcourir les F premiers blocs du fichier pour localiser le bloc F

3. Calculer le premier secteur du *bloc physique P* :

$$P * \text{nb\_secteurs\_par\_bloc} \text{ (+ début du bloc 0)}$$

4. Calculer le *secteur X* :

$$x = \text{premier\_secteur} + ((N \text{ MOD } \text{taille\_bloc}) \text{ DIV } \text{taille\_secteur})$$

5. Position du *byte N dans X* :

$$N \text{ MOD } \text{taille\_secteur}$$

Les **métadonnées** nécessaires à localiser un byte sont :

Pour le système de fichier

- ☐ la taille du secteur ;
- ☐ le nombre de secteurs par bloc / la taille du bloc ;
- ☐ le début du bloc 0.

Pour le fichier

- ☐ le bloc physique associé au premier bloc ;
- ☐ la longueur du fichier, en bytes.

Le bloc F-1 renseigne la position P du bloc F, le F-2 la position du F-1, ...

On perd l'avantage de l'**accès aléatoire**.

## Table d'index

L'index est stocké sur la partition, et contient le **chaînage des blocs de fichier**.

Les blocs d'un même fichier sont dispersés. Pour un accès plus rapide, la **table d'index** est **chargée en RAM** au démarrage (**faiblesse de la FAT**).

Comment localiser le byte N (avec l'index) ?

- Comprendre : Quelle est la position du byte N dans le secteur n°X, en se basant sur l'index ?

1. Trouver le *bloc* du fichier *F* contenant le byte N :

$$F = N \text{ DIV } \text{taille\_bloc}$$

2. Pour trouver le *bloc physique* *P*, il faut regarder dans l'index.

3. Calculer le premier secteur du *bloc physique* *P* :

$$P * \text{nb\_secteurs\_par\_bloc} \text{ (+ début du bloc 0)}$$

4. Calculer le *secteur* *X* :

$$x = \text{premier\_secteur} + ((N \text{ MOD } \text{taille\_bloc}) \text{ DIV } \text{taille\_secteur})$$

5. Position du *byte* *N* dans *X* :

$$N \text{ MOD } \text{taille\_secteur}$$

Les **métadonnées** nécessaires à localiser un byte sont :

Pour le système de fichier

- ☐ la taille du secteur ;
- ☐ le nombre de secteurs par bloc / la taille du bloc ;
- ☐ le début du bloc 0 ;
- ☐ la **table d'index** du système de fichier.

Pour le fichier

- ☐ le bloc physique associé au premier bloc ;
- ☐ la longueur du fichier, en bytes.

### Exemple :

Dans le cas du fichier B, si le bloc 0 démarre en 0, que B démarre en 22 et qu'un bloc mesure 1Kb, pour  $N = 3000$  on aura :

- ❑ Dans quel bloc de fichier est contenu N ?  
 $F = 3000 \text{ bytes DIV } 1\text{Kb} = 2$
- ❑ Quel est le bloc de partition correspondant à N ?  
 $P = \text{bloc } 15$  (suivre le chaînage)
- ❑ Quel est le premier secteur du bloc 15 ?  
 $15 * 2 = 30$
- ❑ Quel est le secteur contenant N sur la partition ?  
 $30 + (3000 \text{ bytes MOD } 1\text{Kb}) \text{ DIV } 512 = 31$
- ❑ Quel est le décalage en bytes dans le secteur 31 ?  
 $3000 \text{ MOD } 512 = 440$

Le **byte 3000 de B** est dans le **secteur 31** à la **position 440**.

## Fragmentation interne

Le dernier bloc n'est pas toujours entièrement rempli.

La **fragmentation interne** est l'espace perdu **à l'intérieur** des derniers blocs **des fichiers**.

Comment choisir la taille des blocs ?

Grands blocs	→	Perte de place
Petits blocs	→	Perte de temps

Les fichiers étant fragmentés, le nombre de **déplacements de tête de lecture** est plus important, ce qui **ralentit** la lecture. Une solution est de **réorganiser systématiquement** les fichiers, une autre est d'utiliser un outil pour **défragmenter**.

## Répertoire

Un **répertoire** est un **fichier** particulier qui contient certaines **métadonnées des fichiers** qu'il « contient », qui permettent de localiser ces fichiers.

Exemples de métadonnées d'un fichier :

- ❑ nom (8 bytes) + extension (3 bytes) ;
- ❑ adresse de début (n° du premier bloc) ;
- ❑ longueur du fichier ;
- ❑ attributs (ARCHIVE, READ\_ONLY, HIDDEN, SYSTEM, DIRECTORY...) ;
- ❑ numéro de l'entrée de la MFT (NTFS) ;
- ❑ numéro d'inode (EXT) ;
- ❑ ...

Le premier byte du nom peut prendre des valeurs particulières :

- ❑ 0x00 : première entrée libre ;
- ❑ 0xF5 : fichier supprimé ;
- ❑ 0x05 : représente 0xF5.

Possibilité d'utiliser des **noms longs** depuis VFAT. En cas de nom long, on utilise plusieurs entrées répertoire. Chaque nouvelle entrée est notée ATTR\_LONG\_NAME et permet d'étendre le nom de 13 caractères.

La **combinaison d'attributs est ignorée par DOS**, ce qui assurait la rétro-compatibilité.

Un répertoire contient au moins **deux sous-répertoires** : . et ..

Chaque fichier/sous-répertoire est décrit dans un **descripteur de 32 bytes**.

Pour localiser un fichier, il faut lire le répertoire parent. Un répertoire étant un type de fichier, il faut aussi regarder son répertoire parent pour le localiser.

Pour **localiser le répertoire racine**, EXT, FAT et NTFS utilisent des **conventions** :

- ❑ EXT : le répertoire racine a une **position fixe**, l'inode 2;
- ❑ FAT – NTFS : la **position** est **calculable** ou renseignée à un endroit calculable.

## Appels système

open localise un fichier avant toute lecture/écriture. Il mémorise l'emplacement du fichier dans la **table des descripteurs des fichiers ouverts (TDFO)**, qui est en RAM, puis retourne l'indice de la description du fichier dans la table.

La position courante est également mémorisée dans la table et mise à jour par les autres appels système (read, write...). L'appel système close libère l'entrée de la table.

En NTFS, il faudra refaire la table de chainage à chaque défragmentation.

En NTFS, le premier bloc est libre pour la défragmentation.




## FAT = File Allocation Table

Système de fichiers de DOS, reconnu par d'autres OS : Windows, GNU/Linux, Mac OS...

L'allocation de l'espace se fait **par blocs** et avec un **index** :

- ❑ clusters (blocs) ;
- ❑ FAT (File Allocation Table - index) :
  - FAT12 (12 bits) ;
  - FAT16/VFAT (16 bits) ;
  - FAT32 (28/32 bits).

Boot	Fat	Copie FAT 	Répertoire racine	Espace libre
------	-----	--	----------------------	--------------

Le **secteur de boot** contient les **métadonnées** du système de fichiers.

Les clusters ne sont pas alignés sur le début de la partition.

La taille des clusters est définie au formatage.

Un **cluster** représente  $n$  secteurs (512 bytes),  $n$  étant une puissance de 2.

La FAT correspond à l'index et :

- ❑ contient une entrée par cluster de la partition ;
- ❑ décrit le chainage des clusters des fichiers ;
- ❑ chaque entrée contient le numéro du cluster suivant.

Des valeurs sont réservées pour indiquer :

- ❑ un **cluster disponible** (0) ;
- ❑ un cluster **défectueux** ( (F)FF7H ) ;
- ❑ le **dernier cluster** du fichier ( (F)FFFH, -1 ).

## Tailles maximales théoriques des partitions

<u>Nom</u>	<u>Bits / Bytes</u>	<u>Max Clusters</u>	<u>Taille Max</u>
FAT16	16 / 2	$2^{16}$	4 Gb
FAT32	32 / 4	$2^{32}$	16 Tb

### Calculer la plus grande partition FAT 16

$$2^{16} * 64Kb = 2^6 * 2^{10} * 2^6 * 2^{10} = 2^{32} \text{ bytes} = 2^2 * 2^{10} * 2^{10} * 2^{10} = 4Gb$$

### Quelle est la taille de la table FAT/de l'index ?

Le **nombre d'entrées dans la FAT** est le *rapport entre la taille de la partition et la taille des clusters*.

La **taille de la FAT** est égale au *nombre d'entrées dans la FAT \* la taille d'une entrée FAT*.

### Exemple :

<input type="checkbox"/> Type de FAT	FAT32
<input type="checkbox"/> Taille de la partition	500 Gb
<input type="checkbox"/> Taille d'un cluster	4 Kb
<input type="checkbox"/> Taille d'une entrée FAT	4 bytes = 32 bits

Le **nombre d'entrées de la FAT** est de  $500.000.000 \text{ Kb} / 4 \text{ Kb} = 125.000.000$ .

La **taille de la FAT** vaut  $125.000.000 * 4 \text{ b} = 500.000.000 \text{ b} = 500 \text{ Mb}$ .

### Taille des tables

La taille maximum d'une **table FAT16** vaut **128 Kb**.

La taille maximum d'une **table FAT32** peut devenir très grosse, ce qui est source de ralentissement pour le système, la table étant **chargée en RAM au démarrage**.

En FAT16, la table peut avoir 512 entrées dont 511 sont dédiées au répertoire racine.

Si les 511 entrées sont occupées, il n'est plus possible d'y insérer de fichier.

La 512e entrée est utilisée pour stocker le Volume-ID.

### Répertoires

En FAT16, le **répertoire racine** est de taille **2 bytes** et sa position (fixe) doit être calculée.

En FAT32, le **premier cluster du répertoire racine** se trouve au **cluster 2** et est de taille **4 bytes**. La suite est **chaînée dans la table FAT** comme pour tout autre fichier.

### Localisation

Les clusters sont numérotés **depuis le cluster 2**.

En FAT16, le cluster 2 suit le **répertoire racine**.

En FAT32, le cluster 2 suit les **tables FAT**.

La position d'un cluster est calculée en fonction des métadonnées du système de fichiers.

## EXT

### Structure interne

Un **mini-disque/partition** est composé de 4 zones :

- ❑ le **boot area** ;
- ❑ le **superbloc** ;
- ❑ le **tableau d'inodes** ;
- ❑ le **tableau de blocs**.

Un **bloc** est un ensemble de bytes contenant uniquement les données du fichier. Sa taille est exprimée en nombre de secteurs.

Un **inode** est un ensemble structuré qui contient uniquement les métadonnées du fichier, sauf son nom. Chaque inode est repéré par son numéro dans le tableau d'inodes.

On retrouve dans un inode :

- ❑ la **taille** du fichier en bytes ;
- ❑ l'**identifiant** du **propriétaire** du fichier (uid) ;
- ❑ l'**identifiant** du **groupe** auquel appartient le fichier (gid) ;
- ❑ les **droits d'accès** (rwx) ;
- ❑ les **dates** (de dernière modification de l'inode, de dernière modification des données, de dernier accès, d'effacement) ;
- ❑ la **liste continue des blocs** de ce fichier ;
- ❑ le **nombre de liens physiques** de l'inode.

La **liste des blocs** est composée de 13 pointeurs vers d'autres blocs :

- ❑ les 10 premiers pointent vers d'autres blocs contenant des données du fichier ;
- si le fichier ne tient pas dans ces 10 premiers blocs, les données restantes sont stockées dans :
- ❑ les blocs qui sont stockés dans le bloc pointé par le 11e pointeur de la liste des blocs ;
- si le fichier dépasse de ces derniers blocs, les données restantes sont stockées dans :
- ❑ les blocs pointés par les blocs pointés par le 12e pointeur de la liste des blocs.

Un **répertoire** est un fichier structuré en enregistrements. Un **enregistrement** par fichier contenu dans le répertoire. Chaque enregistrement contient **2 champs** : le **nom du fichier** et son **numéro d'inode**.

La racine/root correspond à l'inode 2 (l'inode 1 gère les blocs défectueux).

Le **superbloc** contient des informations pour savoir :

- ❑ où commencent les tableaux d'inodes et de blocs ;
- ❑ le nombre d'inodes et de blocs sur ce système de fichiers ;
- ❑ la taille d'un bloc ;
- ❑ si le système de fichiers a été correctement démonté ;
- ❑ les emplacements des blocs libres.