Systèmes d'exploitation.

a) Introduction

1) RAM (Random Acces Memory) – Mémoire centrale

- Ensemble de mots adressables
- On peut y lire et y écrire un mot à une adresse donnée
- Contient les données et les instructions à exécutées
- Volatile

RAM et CPU utilisent un bus d'adresse et un bus de données pour communiquer

2) CPU (Central Processing Unit) – processeur (serveur)

Interprète et exécute une instruction à chaque cycle

- ALU – Unité de calcul
- UC – Unité de commande
- Registre IP – Adresse de l'instruction à exécuter
- Registre RI – Instruction à exécuter
- Décodeur d'instruction
- Séquenceur de commande
- Horloge (interne ou externe)

MOV AX,[100h]

- Cette instruction MOV, met dans AX le contenu de l'adresse 100h de la RAM

Ex : Départ :

- IP vaut 300h
- En 300h se trouve l'instruction MOV AX,[100h] (codée par exemple 8B 0100h)
- En 100h se trouvent les caractères « abcd »
-

Top 1 : 300h sur bus d'adresse RAM (adresse contenu dans IP sur bus d'adresse RAM)
Top 2 : IP <- 301h et via bus 8B 0100h data -> RI (IP <- adresse+1 et instruction via bus data -> RI
Top 3 : 100h sur bus d'adresse RAM (exécution de l'instruction)
Top 4 : abcd (codes correspondants) VIA le bus data et ALU ->

Les programmes ne sont pas toujours des *séquences*.
IP n'est pas toujours simplement incrémenté de 1.
Ordinateur : Permet d'exécuter les instructions d'un programme chargé en mémoire
C'est un programme nommé **CHARGEUR (LOADEUR)** qui charge le code binaire en mémoire

3) Périphériques

Dispositifs permettant d'échanger des données avec un ordinateur et/ou stocker des données de manière permanente. (Clavier, souris, carte réseau, écran,…) Utilisent aussi des bus pour communiquer.

b) Emergence et évolution des systèmes d'exploitation

1) Protection mémoire

La mémoire contenant du code système ne peut être modifiée par un programme d'utilisateur. On peut réaliser cette protection par deux registres spéciaux (délimitent la zone mémoire réservée au programme). Ils seront modifiés à chaque exécution du code système.

⇨ Nécessite un fonctionnement du processeur en **mode privilégié.**

2) Adresse de retour

Le code système est mis à disposition via les instructions **CALL/RET.**

- L'adresse de retour lors d'un CALL (appel de fonction) doit être mémorisée
- On peut la stocker dans un registre du processeur (AR), l'instruction CALL peut s'en charger)
- L'instruction RET restaure IP à partir du registre AR

CALL : Permet d'exécuter un bout de code situé ailleurs et de revenir au code appelant

Hypothèses de départ :

- IP vaut 432
- En 432h se trouve l'instruction CALL 724h

Après l'exécution du CALL :

- AR vaut 433h (IP← IP+1)
- IP vaut 724h
- Les instructions en 724h et suivantes (bout de code) sont exécutées jusqu'à l'instruction **RET** :
- IP ← AR (=433h)
- Après l'exécution du RET, IP vaut 433h

Le programme continue à s'exécuter et récupère la donnée lue

Un code système peut en appeler un autre

Ses bouts de codes sont l'ancêtre des appels systèmes open, read, write…
Par exemple, l'appel système write permet d'écrire n octets dans un fichier : dans le cas d'un disque dur, le code va positionner les têtes, commander la lecture, et vérifier toutes sortes d'erreurs…

⇨ Mais cet appel système permet aujourd'hui aussi bien d'écrire des caractères a l'écran, carte son et carte réseau.

Une étape dans l'évolution pour rentabiliser le CPU en diminuant les temps morts entre les exécutions de programmes. L'enchaînement automatique de programmes : le travail en mode **batch**

Les systèmes batch permettent de rentabiliser l'utilisation du CPU par les *moniteurs d'enchaînement*.

- Permet d'enchaîner plusieurs programmes
- Notion de JOB/travail
- Nécessite un langage de description des étapes et de leurs **données**
- Nécessite un programme capable d'interpréter ce langage
- Nécessite un programme de chargement

Le transfert élémentaire RAM < = > périphériques (écriture ou lecture d'une unité adressable) est assuré par le CPU (programme) vitesse périph < vitesse CPU

Le contrôle des périph lents ralentit le processeur, le programme attend à chaque transfert élémentaire. Le CPU est donc en attende pendant le transfert.
Le programme utilise une scrutation (**polling**) de l'état du périphérique pour savoir si une lecture est terminée. ➔ Utilisation inefficace du processeur

3) Mémoire tampon

Le travail du CPU peut être allégé par l'utilisation de :

- Contrôleurs de périphériques plus « intelligents »
- Mémoires tampon dans le contrôleur

Ceci permet de combiner une technique de ***polling et lecture anticipée***.
Transfert élémentaire mémoire tampon < = > périph réalisés par périph.
Lecture anticipée et polling par le programme rendent la programmation trop complexe

- Le programme doit anticiper ses lectures de données et doit s'occuper à autre chose pendant celle-ci
- Le programme est aussi responsable du polling du périph

4) Interruptions et appels systèmes

L'interruption est un mécanisme hardware(le processeur s'en charge) permettant d'apporter une réponse :

- A une erreur de fonctionnement (erreurs d'exécution : division par 0 …)
- A un évènement soudain externe (bloc disponible, horloge)
- Les appels systèmes (interruptions logicielles, produites sur demande du programme : demande d'E/S.

A chaque interruption est associé un morceau de code de traitement de l'interruption. Ce code fait partie du *système d'exploitation* et est stocké dans un tableau dont les entrées sont indicées par le numéro de l'interruption. L'appel système n'est autre qu'une interruption qui n'est pas générée par des évènements externes, mais par une instruction spécifique sur demande du programme.

- Un fil sert à signaler au processeur si il y'a une interruption ou pas.
- Un traitement spécifique software doit être démarré à l'insu du programme

Le traitement d'une interruption consiste à :

- Arrêter le programme en cours : sauvegarder l'état du processeur (software, hardware) et exécute le code de service de l'interruption
- Rétablir l'état de la machine
- Reprendre l'exécution du programme interrompu (si possible)

Si une interruption est détectée :

- Le CPU sauvegarde la valeur d'IP courant (pointe vers la prochaine instruction) dans la Pile
- Le CPU donne à IP la valeur de l'adresse du code de traitement de l'interruption (software)
- Traitement des instructions (par la machine simplifiée)
- La dernière instruction de ce code restaure la valeur de l'IP

Le mode privilégié

Il reste un problème à résoudre : un programmeur peut écraser, par erreur, l'adresse de la routine de gestion de l'interruption (stockée en 0) ou la routine même.

⇨ Le système d'exploitation doit se protéger :
Interdit à l'utilisateur mais pas au système d'exploitation.

```
MOV [0], EAX
```

Comment se protéger?

Un mécanisme de protection : **le mode privilégié**.

Les routines d'interruption : **code** qui s'exécute en mode privilégié, exécutent certaines instructions non accessibles à un programme normal :

- Instruction qui permet de terminer un programme
- Les instructions qui permettent de modifier l'adresse de la routine d'interruption
- Utiliser des registres avec des plages d'adresse
- Interdire l'utilisation de certaines instructions en mode utilisateur
- Plusieurs modes de fonctionnement du processeur

Un tournant décisif est lié à l'utilisation des interruptions pour signaler la fin d'une E/S. On utilisera également les interruptions pour exécuter le code système de lancement d'une lecture.

Appels Systèmes

Les appels systèmes sont des interruptions logicielles demandées par un programme lorsque celui-ci a besoin d'exécuter du code système (privilégié) : demande d'E/S, fin d'un programme (read, write, exit), …

⇨ L'instruction **INT** permet de simuler une interruption, elle sert essentiellement à appeler les appels systèmes.

c) Emergence : Multiprogrammation

Le CPU est libre pendant toute opération d'entrée/sortie et c'est toujours par soucis de rentabiliser l'utilisation des Unités Centrales (éviter les temps d'attente) qu'émerge la **multiprogrammation** :

- Plusieurs programmes sont chargés en mémoires en vue d'une exécution entrelacée.
- Lorsqu'un programme demande une entrée/sortie il est bloqué au profit d'un autre qui récupère le CPU.

1) La notion de processus

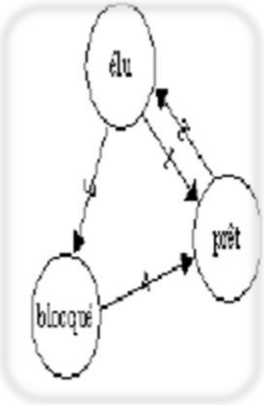
La notion de processus est un modèle simple pour représenter l'exécution concurrente de tâches au sein d'un système d'exploitation. Elle a été utilisée la première fois avec le système Multics et popularisée depuis.

Un processeur modélise l'exécution d'un programme et **possède** :

- Un compteur ordinal
- Un jeu de registres
- Sa zone mémoire (peut être virtuelle)
- Une information d'état

Un processus alterne **3 états** :

- Prêt : suspendu en faveur d'un autre, il ne lui manque que le CPU
- Élu : en cours d'exécution (a le CPU)
- Bloqué : en attente d'un évènement externe (bloc disque, ressource…)



1 : le processus à épuisé le quantum de temps qui lui est attribué. L'ordonnanceur appelé par une interruption horloge élit un autre processus
2 : L'ordonnenceur élit ce processus parmi les prêts
3 : Le processus s'endort en attente d'un évènement externe (horloge, données, …)
4 : L'évènement attendu par le processus se produit, il est géré par le Système d'exploitation en interrompant temporairement le déroulement du processus élu pour faire passer le processus bloqué à prêt

2) Le time slicing

C'est le temps maximum pendant lequel un programme peut garder le processeur. Une fois ce temps écoulé, le système doit pouvoir récupérer la main pour la donner éventuellement à un autre. Mais comment faire pour récupérer la main alors que c'est un programme qui occupe le processeur n'a aucune intention de rendre la main ?

Par l'interruption horloge et mécanisme de préemption
–>l'**Ordonnanceur** est appelé par l'interruption horloge et a le choix du prochain programme à exécuter
L'enjeu : minimiser les temps de réponse, permet une exploitation interactive
L'envers de la médaille : gérer des changements de contexte et occupe du temps CPU

Ordonnancement **coopératif** : laisse s'exécuter un processus jusqu'à ce qu'il rende la main spontanément (premiers Windows)
Ordonnancement **non préemptif** : laisse s'exécuter un processus jusqu'à ce qu'il bloque (E/S, ...) ou qu'il rende la main spontanément
Ordonnancement **préemptif** : laisse s'exécuter un processus jusqu'à ce qu'il bloque (E/S, ...) ou qu'il rende la main spontanément, ou que son temps soit écoulé (Windows NT, 2K, XP, linux depuis noyau 2.6,...)

Rôles d'un OS identifiés jusqu'ici :

- Périphériques virtuels (appels systèmes et mécanisme d'interruption)
- Moniteur d'enchaînement
- Protection de l'accès à la mémoire
- Ordonnancement
- Un Ordonnanceur peut être non préemptif même en présence d'interruptions horloge
- Un Ordonnanceur préemptif ne peut exister sans interruptions horloge

V ou F ?

- CALL permet de mémoriser IP sur la pile
- un appel système se fait grâce à l'instruction CALL
- une interruption oblige le processeur à s'arrêter
- RET est une instruction qui provoque la mémorisation d'IP sur la pile
- la multiprogrammation nécessite un processeur du genre dual core
- un processus peut passer de l'état prêt à bloqué.
- le SE s'exécute toujours en même temps qu'un processus.

d) Emergence : Ressources

Une ressource qui ne peut être partagé par plusieurs processus se dit **non partageable** (ex : CPU, imprimante, graveur, …). Exemple de ressource partageables : écran, disque,…)

Une ressource qui ne peut être retirée à un processus tant qu'il n'a pas fini de l'utiliser se dit non préemptible. Ex : imprimante, graveur,…

Un processus qui demande une ressource non partageable et non préemptible, déjà attribué à un autre, est bloqué dans l'attente que l'autre processus la restitue, il ne peut donc obtenir le CPU (➔ appel système et ordonnanceur).

Il peut survenir des situations où plusieurs processus se bloquent les uns les autres indéfiniment : C'est ce que l'on nomme **étreinte mortelle** ou **interblocage** ou **deadlock** conditions nécessaires :
- utilisation de **plusieurs** ressources
- ressources **non préemptibles**
- ressources **non partageables**

En effet :

- Une ressource **partageable** ne bloque pas un processus
- Une ressource **préemptible** peut toujours être retirée momentanément à un processus pour permettre à un autre de se terminer.
- Dans le cas d'**une ressource unique** un des deux processus pourra bloquer l'autre, mais pas l'inverse-->le processus qui a la ressource ne peut se terminer et céder la ressource à l'autre ensuite.

Solutions :

- ignorer ? (politique de l'autruche)
- prévenir ?
- récupérer ? La prévention n'est pas sans coût

Ignorer

Certains OS comme linux et windows2000 ignorent certaines situations d'inter blocage parce qu'ils considèrent que la situation est suffisamment rare que pour ne pas justifier la mise en œuvre d'une gestion complexe.

Prévenir

Le problème de l'accès à l'imprimante est résolu par une **technique** de spool : aucun processus n'accède directement à l'imprimante à l'exception du **Spooler** : processus dédié aux impressions.

La technique du spooler permet d'éviter les blocages des processus qui impriment :
Les processus qui ordonnent des impressions écrivent dans des fichiers temporaires déposés dans un répertoire dédié au spooler Ce dernier commande l'impression d'un fichier à la fois et évite ainsi le mélange des sorties.

Le **séquencement** est assuré par le spooler. De plus les programmes qui désirent imprimer ne sont pas bloqués en attente de l'imprimante. Evidemment le spooler ne peut commencer à imprimer un fichier avant que celui-ci ne soit complet. Cette approche ne s'applique pas à toutes les utilisations de ressources (lectures sur bande, ...).

VRAI- FAUX ?

- un interblocage ralentit le processeur
- un interblocage ne peut être détecté par l'OS
- une ressource préemptible et non partageable peut occasionner un interblocage F

Questions ouvertes

- Citez quelques rôles d'un OS identifiés jusqu'ici.
- Quel est l'avantage d'utiliser l'instruction INT pour un appel système ? Expliquez
- Avantage(s) et inconvénient(s) du Time Slicing par rapport à la multiprogrammation ? Expliquez

e) **Système fichiers**

Pour stocker les informations de manière **permanente** la RAM ne convient pas :
- **volatile** : plus de courant => information perdue
- chère
- limitée en taille

On va donc stocker l'information dans des disques ou autres supports extérieurs **mémoires de masse** (bande, disque magnétique, . . .) sous forme de **fichiers**

Un **Système de fichiers** permet de :
- stocker les informations de manière **permanente**
- nommer les informations
- partager des informations volumineuses
- . . .

Quelques noms de **systèmes de fichiers** auxquels vous êtes confrontés dans le cadre de vos études : FAT, NTFS, EXT

On distingue deux vues d'un système de fichiers :
- utilisateur - application
- système d'exploitation - mise en œuvre

La vue utilisateur - application : voit le service offert et donc **utilise l'appel système.**
La vue système : il s'agit de la mise en œuvre du service et donc le **code de l'appel système**

Structure d'un fichier : (suite d'octets, suite d'enregistrements, arborescence)

Accès à un fichier :
- accès séquentiel (bande ou disque)
- accès aléatoire (devenu possible à partir des disques)

Le système doit mettre à disposition des appels système : **read, write, seek** sur linux
Où se trouve le fichier sur le disque ? -> "open" le cherche et mémorise son emplacement en mémoire
Où en est-on dans la lecture du fichier ? -> "read" mémorise la position courante en mémoire il existe des appels système **open, close.**

1 bloc = n secteurs
Quelle taille donner à un bloc ?
Exemple : secteur = 512 bytes bloc = 8 secteus (4k)
Grands blocs -> peu de lectures nécessaires, **fragmentation interne** plus importante
Petits blocs -> **beaucoup de lectures**, peu de fragmentation interne

Les techniques d'allocation par blocs (non contigu) finissent par générer des fichiers **fort fragmentés** (blocs éparpillés) Tiens, quelle est la différence entre fragmentation externe / interne / des fichiers ?
L'accès à des fichiers fragmentés demande beaucoup de déplacements des têtes -> LENT
-> *Solutions* :
- réorganisation continue (adopté par linux)
- outils de défragmentation utilisables de temps en temps (adopté par Windows)

Un répertoire :

Fonction principale d'un répertoire :
Fournir les informations pour trouver les blocs de disque
- adresse disque et longueur du fichier entier (cas de l'allocation contiguë)
- numéro de bloc (cas de la liste chaînée)
- n° d'inode (EXT)
- . . .

Sert à établir le lien **nom du fichier** <–> **blocs de données**

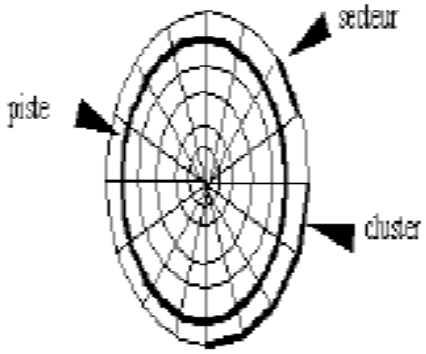
Localisation d'un fichier :

Localiser un fichier sur le disque : utiliser les informations dans la description du **répertoire parent**. En linux et windows, les données d'un répertoire sont stockées sur le disque sous la forme d'**un fichier**. La lecture du fichier /home/mba/test en linux demande de **localiser, ouvrir et lire**
- /
- /home
- /home/mba
- /home/mba/test

VRAI - FAUX ?
- en allocation de l'espace par blocs, une plus grande taille de bloc augmente la fragmentation externe
- read est un appel système
- un répertoire est décrit sur le disque
- en allocation contiguë pour retrouver un fichier j'ai besoin de connaître uniquement son adresse de début

A quoi sert le "open" ?

f) **FAT**



La taille des secteurs et des clusters est définie au formatage.
1 cluster = n secteur (n = puissance de 2)
Quelques valeurs par défaut :
Secteur : 512 Octets
Pour une disquette : un cluster = 2 secteurs
Un cluster : Plus petit espace contigu sur le disque, pouvant être alloué pour y stocker un fichier. Unité d'allocation.

Structure d'une partition Fat :

- zone réservée (Boot Sector 1 secteur pour FAT 12-16, 32 pour FAT32)
- zone FAT
- zone du répertoire racine (pas pour FAT32)
- zone pour les fichiers et répertoires

Les entrées de la FAT ont une taille fixe :
FAT12 - 12 bits => MAX 4096 clusters
FAT16 - 16 bits => MAX 65536 clusters
FAT32 - 32 bits (28 utiles) => jusqu'à 2exp28-1 clusters (en théorie)
La FAT est chargée en RAM pour des raisons de performance
Tailles théorique des FAT 16 et 32 ? (2 exp 17, 2 exp 30 ! ! !)

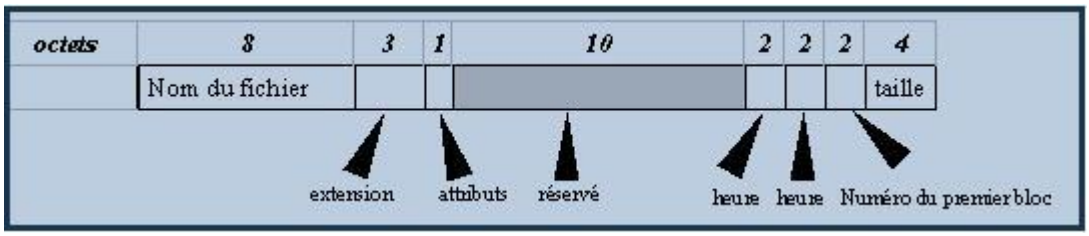
Tailles de partitions maximales (et théoriques) pour différentes tailles de clusters

Taille cluster	**FAT-12**	**FAT-16**	**FAT-32**
512 octet	2 Mo		
1 Ko	4 Mo		
2 Ko	8 Mo	128 Mo	
4 Ko	16 Mo	256 Mo	8 Go (1 To)
8 Ko		512 Mo	16 Go (2 To)
16 Ko		1 Go	32 Go (4 To)
32 Ko		2 Go	2To ? (8 To)

Fin de fichier ? Cluster défectueux ? Cluster libre ? Voici quelques valeurs particulières des entrées de la table
(FAT 12 ou 16) :
- 0 - cluster disponible
- (F)FF0H à (F)FF6H (65520 - 65526) - cluster Réservé
MS-DOS
- (F)FF7H - cluster défectueux
- (F)FF8H à (F)FFFH - dernier cluster du fichier (fin de fichier)
1-65519 : clusters utilisables (non réservés)
1-65526 : plage d'adresses des clusters pouvant être chaînés dans la FAT

Un répertoire

Un répertoire est un fichier comme un autre : il est décrit dans des clusters chaînés
Le répertoire décrit les fichiers qu'il contient (nom, premier cluster, taille ...)
Un répertoire est un **fichier d'enregistrements** :
1 enregistrement = entrée de répertoire (32 bytes) (une par fichier/sous-répertoire)



Tout fichier ou répertoire d'un système FAT (sauf la racine) est décrit dans une entrée de répertoire : 32 octets entrée de répertoire dans la FAT
Nom et extension : 8+3 nous n'abordons pas ici la représentation des noms longs.

Légende :
Attributs :
- archivage
- répertoire
- lecture seule
- caché
- système
- . . .. . .

Heure : création et dernier accès (même en lecture ! !) les deux champs heure sont en réalité une heure et une date (2 bytes chacun)
- heure : (heure : 5 bits, minutes : 6 bits, secondes : 5 bits) -> max secondes 32 (30) -> précision 2 secondes ! !
- date : (jour : 5 bits, mois : 4 bits, année : 7 bits)
->Max année = 127 (+1980) = 2107
Et si on avait représenté heure et date avec un nombre de secondes sur 32 bits ?
(2exp32 sec)/(1 jour =86400 sec) / 365 = 136 ans (+1980) = 2116
->(9 ans de plus avec 1 seconde comme précision !)

Répertoire racine :

Particularités du répertoire racine : aucun autre répertoire ne le décrit
Nom? Premier cluster ? Longueur ?
Sa position résulte d'une convention
Son contenu, comme celui des autres répertoires, est une suite de descripteurs de fichiers/répertoires : un par entrée
En FAT12-16
- le répertoire racine a un **emplacement fixe calculable**
- sa taille est fixe !

En FAT32
- répertoire racine = chaîne de clusters dont seulement **le premier est figé**

Appel de l'appel système open

On va simplement utiliser **les registres** pour passer ces informations au bout de code.
Il faudra convenir
1. qu'un registre contiendra l'information 'nom de fichier à ouvrir'
2. qu'un autre registre contiendra le n° de l'appel système
3. et que, après l'exécution du bout de code, un registre contiendra un n° représentant l'entrée de ce fichier dans la table en RAM
Le numéro d'entrée dans cette table pourra ensuite être utilisé pour toute opération (read, seek,. . .) sur le fichier

La commande **chkdsk** utilise cette redondance :
1. parcourt la FAT à la recherche des clusters chainés : ceux qui appartiennent à un fichier
2. cherche les clusters libres : ceux qui ont un 0 en FAT
En théorie un cluster "normal" doit soit être libre soit être chaîné
La comparaison entre ces deux résultats permet d'isoler des **"chaines de clusters perdues"** : les clusters marqués non libres, n'appartenant à aucun fichier
Ces derniers sont récupérés sous forme de fichiers nommés FILE0000.CHK, FILE0001.CHK dans le répertoire racine.

Cluster défectueux ?

Cluster défectueux dans un fichier -> perte d'une partie de données cluster défectueux dans la FAT -> le système de fichiers risque d'être **fortement compromis**
-> Une **copie de la FAT** peut être maintenue pour garantir une plus grande fiabilité et si le cluster défectueux est au niveau d'un fichier utilisateur c'est moins grave
La commande **chkdsk** tente également de récupérer l'information d'un cluster défectueux.

Pour parcourir un fichier **plusieurs accès à la FAT** sont nécessaires. Pour limiter les accès disque, **la FAT est chargée en RAM** et mise à jour en même temps en RAM et sur le disque
Mais quelle est la **taille** de la FAT (en RAM) ?

Taille de la FAT pour quelques exemples d'organisation :
Taille FAT = (Taille Part / Taille Cluster) * taille entrée FAT

FAT	Taille Partition	Taille Blocs (Kb)	Taille FAT
16	1Gb	32Kb	64Kb
32	1Gb	4Kb	1Mb
32	32Gb	4Kb	32Mb
32	200Gb	4Kb	200Mb !

- **taille FAT** très importante pour grandes partitions, avec clusters de petite taille
- chargement de la FAT en RAM : une opération **lente** au démarrage du système

La taille des fichiers en FAT est **limitée à 4Gb**
De plus la FAT ne prévoit pas la gestion d'attributs pour la **protection d'accès** entre utilisateurs différents.
Ce qui en fait un système de gestion non adapté à des environnements multiutilisateurs.

g) **NTFS**

NTFS : **N**ew **T**echnology **F**ile **S**ystem
Système de fichiers **natif** de Windows NT, utilisé également par Windows 2000, Serveur 2003 et XP
rupture de la compatibilité ascendante (FAT) :
Créé pour satisfaire des critères de
- **taille** des volumes et fichiers
- **sécurité**
Un enjeu : la performance

LCN (Logical Cluster Number) : numéro d'ordre dans la partition, n° du secteur où démarre un cluster sur la partition : LCN * taille du cluster
VCN (Virtual Cluster Number) : numéro d'ordre dans le fichier

Valeurs par défaut :

Taille partition	***Taille Cluster***	***secteurs***
512 Mb	512b	1
512 Mb à 1 Gb	1 Kb	2
1Gb à 2 Gb	2 Kb	4
2Gb	4 Kb	8

4kb est la taille de cluster pour des partitions > à 2 Gb
NB. La compression de fichiers ne supporte pas des tailles de clusters supérieures à 4Kb

Structure d'une partition NTFS

En NTFS toute l'information est stockée sous forme de **fichier les métadonnées du système** de fichiers sous forme de **fichiers système !**
- fichier des blocs libres
- fichier des blocs défectueux
- fichier "répertoire racine"
- fichier MFT : **le plus important** (une entrée par fichier du système)
- . . .

NTFS retrouve les données des fichiers via la MFT

Dans une partition NTFS on distingue 4(+1 ?) zones :

1. le $Boot, position fixe
2. une zone "réservée" : buffer MFT, pour maintenir le fichier $MFT non fragmenté (accès fréquents)
3. les espaces fichiers et répertoires : pour les données qui ne sont pas contenues dans la MFT
4. le fichier $MFTMirr (sauvegarde début MFT)
5. la copie du secteur 0 est actuellement sauvegardée à la fin de la partition (vrai ?)

La zone réservée à la MFT

- Sa taille peut être configurée au départ : 12,5 25, 37,5 50 % de l'espace disque
- Son but : maintenir le fichier $MFT non fragmenté
Chaque fois que le reste du disque se remplit -> taille diminuée de la moitié
Taille de la zone MFT au départ ? Le défaut est 12,5%
Permet aussi de déplacer la MFT en cas de défaillance d'un secteur.

Les premières 16 entrées de la MFT décrivent des **métadonnées du système de fichiers**.
Il s'agit de fichiers système dont le nom commence par $ quelques fichiers système :
- $MFT : Master File Table - fichier des descripteurs de fichiers
- $MFTMirr : sauvegarde des 4 premières entrées de la MFT
- répertoire racine
- $Bitmap : 1 bit par cluster, par groupes de 8 bytes ; clusters occupés (bits en trop ou correspondant à la copie du secteur de boot sont à 1)
- . . .

Attribut

Un attribut peut être **résident** ou non selon que sa description est entièrement contenue dans l'enregistrement de la MFT ou non, exemple d'attributs de fichiers NTFS :
- le nom (résident) (un fichier peut avoir plus d'un nom : nom DOS, lien Hard)
- les données (résident pour un "petit fichier", non résident pour un "grand fichier") petit = les données sont dans l'enregistrement de la MFT même **Nom** et **données** d'un fichier font partie des attributs.

L'attribut $DATA

Utilisé **surtout** pour le flux de données.
Dans l'entête de l'attribut sont codées notamment :
- *si attribut résident et petit fichier*
. Le début de l'attribut
. La longueur de l'attribut
- *si attribut non résident et grand fichier*
. La taille physique (allouée) de l'attribut (arrondie au multiple de la taille d'un cluster) sur 64 bits.
. La taille logique (réelle) de l'attribut également sur 64 bits.
–> taille théorique maximum fichier : 2exp64 bytes

L'attribut $DATA et **les petits fichiers,** un petit fichier est un fichier dont les données tiennent dans l'entrée de la MFT (Attribut $DATA **résident**)
Attribut résident : contenu dans l'enregistrement de la MFT décrivant le fichier
Un flag **"résident"** dans l'entête de l'attribut indique cette propriété

L'attribut $DATA et **les grands fichiers**
Pour représenter de "grands fichiers" un mécanisme de **chainage de blocs** est nécessaire
Le chainage est entièrement décrit dans l'entête de l'attribut $DATA dans le champ $DATA_RUN présent si l'attribut est **"non résident"**
Dans ce cas :
L'attribut DATA est limité à son entête

DATA-RUN : liste de blocs

Idée : au lieu d'un "descripteur" par cluster on a un descripteur par **groupe de clusters contigus**
NTFS tire avantage de la "non fragmentation des fichiers" dans sa représentation :
Un **groupe de clusters** aussi dénommé **RUN**
Un **RUN** est identifié par son premier cluster et le nombre de clusters consécutifs : un RUN est décrit par un couple d'informations chacune de ces informations à une longueur variable
Choix entre utiliser des longueurs maximum ou stocker l'info de longueur :
->Pas de place inutile
->nécessité de deux descripteurs supplémentaires pour la longueur

Exemple :
Un run représenté dans le champ DATA_RUN
13 18 56 34 11 : 5 bytes d'information
1 - 3 - 18 - 563411 : 4 informations
- 0x1 (4 bits) : la longueur est codée sur 1 byte
- 0x3 (4 bits) : le décalage est codé sur 3 bytes
- 0x18 (1 byte) : la longueur vaut 0x18 (nb de clusters)
- 0x653411 (3bytes) : le décalage vaut 0x563411
Décalage : LCN du premier RUN, ou décalage par rapport au RUN précédent

Grands répertoires

Décrit dans la MFT en partie dans l'$INDEX_ROOT et en partie par un champ DATA_RUN semblable à celui des fichiers Un grand répertoire, utilise deux attributs supplémentaires : |STANDARD_INFORMATION|$FILE_NAME|$INDEX_ROOT|$INDEX_ALLOCATION|$BITMAP|
L'attribut $INDEX_ALLOCATION contient notamment le champ $DATA_RUN qui adresse plusieurs "Record Index" Chaque "Record Index" contient plusieurs IndexEntry.

Fichiers – Sécurité

Sécurité : le point de vue protection des fichiers
La problématique existe pour d'autres objets que les fichiers
Notion de **domaine** : utilisateur/groupe à qui on confère un ensemble de droits par fichier : paires <fichier, droits> exemples :
- d1 : F1[R], F2[RW]
- d2 : F3[R], Print[W]
- d3 : F6[RWX], Print [W]
Di correspond à un utilisateur ou à un groupe

NTFS utilise la technique des ACL pour protéger ses fichiers
Access Control List

Les Autorisations NTFS définissent :
- qui peut accéder à un fichier/répertoire
- quelles opérations lui sont permises les autorisations s'expriment en
- interdictions
- permissions

Les permissions s'appliquent :
- par fichier/répertoire
- pour des opérations spécifiques (lire, écrire, modifier, exécuter...)
- à des utilisateurs ou groupes d'utilisateurs

Quelques règles
- le refus l'emporte sur les autres autorisations
- les autorisations sont cumulatives (pour un utilisateur membre de différents groupes)
- les autorisations sont héritées du répertoire parent (comportement par défaut), mais les autorisations explicites priment sur ces dernières

Le système devra mettre à disposition des appels système qui permettent de modifier les descripteurs de sécurité. Les autorisations de sécurité interviennent à chaque ouverture de fichier :
=>Un des rôles de l'**appel système open** : vérifier les droits du processus appelant.
Si les droits ne sont vérifiés lors de l'open, toute modification de droits ne prend effet que pour les fichiers nouvellement ouverts.

Le pour

NTFS par rapport à FAT32
- adapté à l'évolution vers grands volumes (déconseillé pour petits <400Mb)
- plus sur (info critique redondante, droits d'accès)
- plus rapide au démarrage sur grands volumes
- plus performant dans l'accès aux grands répertoires
- consistance données (journalisation)

Le contre

- blocs -> souffre de la fragmentation de fichiers
- mise en œuvre assez lourde (remarquable redondance
. . .)
- quid de la performance avec disque plein ?
Fragmentation de la MFT.

Pour ou contre ?
- la zone réservée à la MFT est réduite si le disque devient plein
. Ok (si définitif)
. Fragmentation MFT (si temporaire)
- la MFT n'est jamais réduite
. Ok (si allocation définitive) la zone reste occupée (même si on supprime tous les fichiers !)
Une partition NTFS supporte mal les variations d'occupation de l'espace disque