
# Cours de Système d'exploitation

## **ISIMA** FX

Christophe Gouinaud

# Table des matières

# 1 Introduction

## 1.1 historique

### 1.1.1 Préhistoire (1940 - 1950)

Les premiers ordinateurs se programmaient directement en langage machine (binaire) et n'exécutaient qu'un seul programme à la fois. Le programme devait contenir toute la gestion du matériel.

Il faut cependant dire qu'à cette époque, le matériel était limité : un processeur, une mémoire, clés au pupitre, une machine à écrire et le lecteur de cartes ou de rubans perforés.

### 1.1.2 Antiquité : années 50

Deux grands groupes d'éléments sont apparus : des outils de développement pour faciliter la construction des programmes, des bibliothèques de sous-programmes contenant la gestion du matériel, c'est-à-dire des sous-programmes que les programmeurs pouvaient intégrer dans leurs programmes évitant la gestion directe des phériphériques.

C'est aussi à cette époque que se sont popularisés :

- les assembleurs puis les langages évolués, pour faciliter la conception des programmes : ils évitaient de travailler en langage machine. En 1956 apparaît le premier FORTRAN.
- les chargeurs pour lire des programmes exécutables enregistrés sur cartes ou rubans perforés et les copier en mémoire centrale.
- les premiers processeurs spécialisés gérant des unités périphériques ont d'une part soulagé le processeur central, d'autre part facilité la gestion des unités périphériques.
- le traitement par lots [Batch], c'est-à-dire l'enchaînement automatique des programmes. Avant le traitement par lots, l'utilisateur devait réinitialiser l'ordinateur et charger son programme avant de le faire exécuter. Le traitement par lot, reposant sur une liste de descriptions de travaux à exécuter, exécute ces opérations automatiquement à la fin de chaque travail. Il faut noter que les utilisateurs n'ont alors plus d'accès direct à la machine. Ils doivent construire une description du travail qu'ils veulent faire exécuter, placer cette description dans la liste de travaux à exécuter (à la fin du paquet de cartes), puis quelque temps plus tard, peuvent venir récupérer le listing contenant la trace de comment leur travail s'est exécuté, et ses résultats éventuels.

Durant cette période les entreprises commencent à utiliser l'ordinateur pour des raisons de prestige.

### 1.1.3 Les années 60

Durant cette période, ont été mis au point et proposés aux utilisateurs :

- la multiprogrammation, c'est-à-dire la possibilité d'exécuter plusieurs programmes simultanément,
- l'accès interactif, c'est-à-dire le retour de l'accès direct au système pour les utilisateurs.

C'est à cette époque que les études théoriques ont abouti à deux systèmes d'exploitation : Multics pour les gros ordinateurs, et son petit frère Unix pour les petits.

La théorie a conduit ces deux systèmes à mettre en valeur :

– la différenciation entre services de gestion du système, et programmes d'application,
– la notion d'interface et de primitive système,
– la structure arborescente des fichiers,
– la notion de shell (interface utilisateur).

Durant cette période l'ordinateur devient réellement populaire dans les entreprises, mais c'est encore une grosse boite située dans une pièce pas trop loin des bureaux directoriaux afin de le montrer lors des visites d'entreprise.

### 1.1.4 Les années 70

Pour les systèmes d'exploitation, ces années ont surtout apporté des améliorations :

– en terme de performances : meilleure gestion des éléments du système,
– en terme de sécurité : partage des éléments en multiprogrammation,
– en terme de convivialité : amélioration des interfaces homme machine...

C'est aussi la période de mise au point des premiers systèmes téléinformatiques : d'abord avec des terminaux distants, puis avec l'interconnexion d'ordinateurs.

Par ailleurs, des services applicatifs ont été élaborés et offerts, dont les plus importants sont sans doute les systèmes de gestion de bases de données, et les services transactionnels.

### 1.1.5 La révolution 1978

Enfin, la fin des années 70 a vu apparaître les premiers micro-ordinateurs, qui ont révolutionné à la fois les systèmes d'exploitation, mais aussi l'idée des ordinateurs chez les utilisateurs :

– pour qu'une gamme de micro-ordinateurs soit d'un prix acceptable, il fallait que le système d'exploitation soit de faible coût, ce qui a entraîné la réutilisation du même système d'exploitation pour des matériels différents. Cela a été en fait la première fois que le système d'exploitation est devenu en partie indépendant du matériel. La révolution a été que ce principe est actuellement repris pour presque tous les ordinateurs.
– la facilité d'utilisation des micro-ordinateurs a rendu les utilisateurs exigeants envers les gros systèmes, pour lesquels ils ont demandé la même facilité d'utilisation que les autres ordinateurs.

Là, le vers était dans le fruit. Il ne restait plus à cette époque qu'à assembler les divers éléments de calcul et de communication pour créer les systèmes tels que nous les connaissons maintenant.

### 1.1.6 Fin des années 80 : la banalisation de l'interface et des réseaux

Un gros effort a été fait sur l'interface homme machine, à la suite de l'apparition des micro-ordinateurs Apple Lisa puis Macintoch. Ces machines ont été les premières à considérer l'interface et la librairie graphique comme un composant à part entière du système d'exploitation.

Cela a amené à présenter aux utilisateurs une interface conviviale qui a la même allure quelque soit le système informatique. Il en existe diverses variantes : Macintoch bien sûr, mais aussi X-Windows du MIT et Windows de Microsoft.

De plus, les capacités des réseaux ont été normalisées et leurs applications se sont démocratisées (Novel Netware). Ils offrent aux utilisateurs la possibilité de disposer simultanément de ressources

installées dans des systèmes divers (terminaux, processeurs, fichiers).

### 1.1.7 Les années 90 : le triomphe du modèle clients-serveur.

Cette période a été celle de la fédération des micro-ordinateurs et des gros systèmes par le réseau. Le prix des micro-ordinateurs avait suffisamment baissé, dans le même temps les besoins de gestion avaient cru, ce qui fait que l'informatique est apparue sur tous les bureaux. Cette situation a créé un bazard sans précédent. La nécessité de centraliser les informations stratégiques a alors poussé au développement de serveur de fichiers, puis de tâches.

Ce développement s'est fait selon un modèle où les micro-ordinateurs se sont vu chargés de toutes les parties interactives (traitement de textes, saisie de données et élaboration de requêtes) et les serveurs de tous les traitements lourds ou de publication de données.

La gestion réseau et l'interface graphique sont passés d'éléments de confort à des parties stratégiques des systèmes d'exploitation.

Un autre fait marquant de cette période est que la puissance de calcul et la mémoire des ordinateurs ont dépassé les besoins de la plupart des applications. Nous sommes donc passés d'une situation où le programmeur était toujours à la recherche d'économies à une situation où la recherche d'économies coute plus cher que l'augmentation de la puissance. Cela a conduit à une complexisation des systèmes d'exploitation sans précédent.

### 1.1.8 L'an 2000 et le triomphe du client léger.

L'apparition du Web et sa démocratisation ont conduit à une standardisation sans précédent des interfaces et donc à créer une situation propice à une extension et à une généralisation du modèle clients-serveur.

On voit donc apparaître partout la notion de client léger c'est-à-dire de micro-ordinateur dont la tâche n'est plus que d'afficher des pages Web plus ou moins sophistiquées. Si on y regarde à deux fois, il ne s'agit ni plus ni moins que de la réapparition des terminaux **VT** des années 70-80, la convivialité en plus.

Cette année a aussi été marquée par la prise de conscience de la durée des systèmes et des applications informatiques. Le fameux BUG de l'an 2000 a montré que l'informatique avait maintenant une longue histoire derrière elle, et qu'il allait falloir la gérer et vivre avec elle pendant de nombreuses années.

### 1.1.9 2001-2005 du pas chers

Plusieurs faits marquants dans cette période :

– recentralisation des ressources informatique,
  Les directions informatique ont enfin pris conscience que la complexité des système pour micro-ordinateurs est la plus part du temps un frein à une bonne gestion. D'autre part les entreprise souhaite de plus en plus maitrisé leurs chaines de production par l'informatique, ces deux fait nous conduise à une diffusion importante des ERP (a decompacter ...) et à la popularisation des applications à bases de web services. Dans ce modèle de systèmes d'informations le systèmes

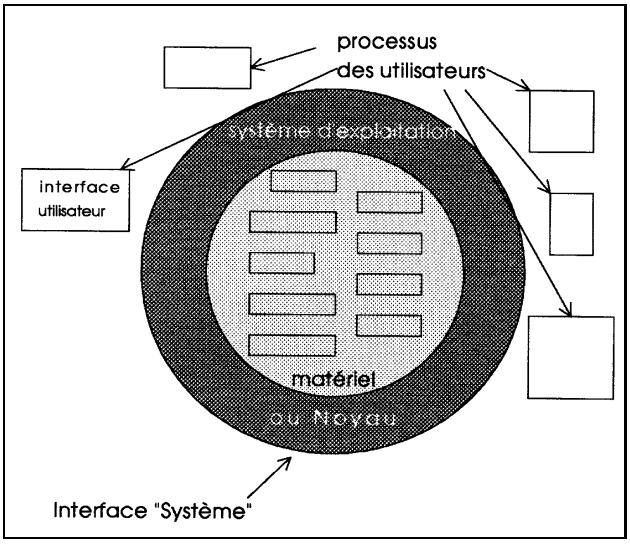


FIG. 1 – Structure des systèmes actuels.

d'exploitation devient secondaire car la gestions des utilisateurs, des données et des applications se fait via une couche de service, souvent appelés serveur d'application, tournant comme n'importe qu'elle application. Certaines versions très sophistiqués prennent également en charge des fonctionnalité de répartitions sur grille d'ordinateurs qui n'ont jamais été bien gérer par les système d'exploitation. Ces applications sont de tel catédrale du logiciel que l'on se demande pourquoi les éditeur de ces logiciels n'intégre pas le système d'exploitation à leurs offres.

- Pause dans la progression de la fréquence d'horloge des processeurs.
- Apparition des ordinateurs utilisant l'écriture manuscript dans leur interface. Il s'agit de l'assemblage de composant existant (ecran tactile, stylets, ...) mais avec des algorithmes de reconnaissance d'écriture simple qui le rende utilisable. Dommination de Microsoft dans ce domaine avec le Sytème Pocket Pc et la version pour TabletPc de Windows XP.
- L'internet devient extrèment bruillant ...
- Le retour d'UNIX sous toute ces formes

Bref, les utilisateurs essayent de dépenser au minimum.

## 1.2 Structure des systèmes informatiques actuels.

Un système informatique est actuellement composé de :

- matériel, une ou plusieurs unités centrales (processeurs, mémoire), qui peuvent être géographiquement distantes, des unités périphériques attachées à des processeurs spécialisés
- un système d'exploitation réparti dans les unités centrales.

Le système informatique est une machine à exécuter des activités que l'on appelle des processus. On distingue deux types de processus : les processus "système" dont le rôle est de gérer le système informatique (matériel et les autres processus) pour la sécurité, le partage des données, le confort de l'utilisateur (plug and play) et les performances et les processus "utilisateurs" qui sont les activités demandées par les utilisateurs.

Dans cette philosophie, le système d'exploitation est conçu pour :

– gérer les processus (créer, détruire, surveiller, ... ),
– gérer l'existence simultanée de plusieurs processus,
– fournir aux processus les moyens d'accès à tous les éléments matériels du système informatique dont ils ont besoin, en contrôlant le fonctionnement de l'ensemble et la validité des requétes,
– offrir une interface la plus conviviale possible,
– optimiser le fonctionnement du système informatique (productivité).

Ce qui est frappant à l'époque où j'écris ces lignes est que l'informatique et les systèmes d'exploitation sont sortis de l'anonymat pour devenir quelque chose de terriblement stratégique. La peur du millénaire s'est traduit par une crainte de l'arrêt brutal de tous les ordinateurs et les constructeurs d'informatique sont de plus en plus dépendants des consortiums développant des systèmes d'exploitation.

## 1.3 vocabulaire

– système multiprocesseur symétrique (SMP)
– Système maître-esclave
– Périphérique
– Ressource
– ERP
– Grille et cluster

## 1.4 Conclusion

Sur tous les ordinateurs, il y a un système d'exploitation.

Les utilisateurs ne travaillent jamais directement sur la machine réelle (ou physique), mais sur l'interface fournie par le système d'exploitation, ou machine virtuelle.

Le système d'exploitation fournit un environnement dans lequel les utilisateurs peuvent préparer des programmes, les faire exécuter, et stocker des données, par l'intermédiaire de processus adéquats.

Mais, les systèmes d'exploitation ont aussi pour objectifs d'optimiser l'utilisation du matériel (processeurs, mémoire et unités périphériques) et, dans les systèmes multi-utilisateurs, de gérer le partage des ressources.

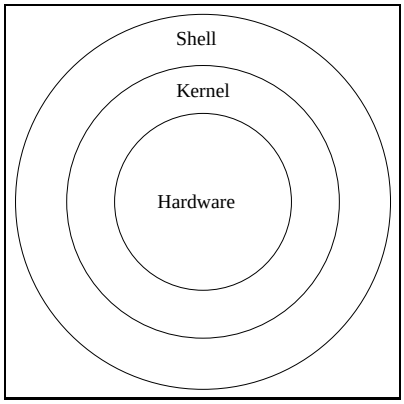


FIG. 2 – Hardware, Kernel et shell

# 2 Notion de noyau

Le rôle du noyau d'un système d'exploitation est de masquer le hardware au reste du système. C'est aussi ce composant qui permettra de construire une vision standard et portable d'un système d'exploitation.

## 2.1 Introduction

On appelle noyau d'un système la partie logicielle qui est chargée du dialogue direct avec le hardware de la machine. Suivant les systèmes, le noyau peut aussi prendre en charge certains aspects logiques tels que le temps réel ou l'accès utilisateur aux fichiers.

La couche visible de l'utilisateur est par opposition appelée Shell et sert donc d'interface entre l'utilisateur et le noyau. De fait, dans les systèmes informatiques modernes l'utilisateur n'accède jamais directement au hardware de la machine, que sont les ports, les contrôleurs. Ces trois couches sont représentées sur la figure 2.

Il existe actuellement 3 philosophies de constitution de noyaux :

- le noyau monolithique
- le micro-noyau
- l'absence de noyau, c'est-à-dire que celui-ci représente l'ensemble des fonctionnalités du système.

Dans tous les cas le noyau n'est qu'une bibliothèque de fonctions permettant de standardiser l'appel aux ressources matérielles.

Ce chapitre est structuré en 4 parties :

- Description des fonctions et éléments d'un noyau
- Philosophie de construction - Appel système
- Description des noyaux UNIX / NT / MS-DOS

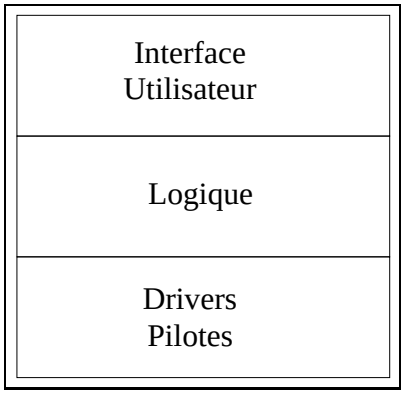


FIG. 3 – Les trois parties du noyau.

## 2.2 Description d'un noyau

Comme cela est écrit plus haut, un noyau n'est pas une tâche d'un système mais un composant logiciel (une sorte de librairie) auquel peut faire appel le programmeur du système d'exploitation.

Le noyau est donc toujours structuré en 3 couches tel que représenté sur la figure 3 :

- l'interface Shell est une série de routines permettant aux autres composants logiciels de dialoguer avec le hardware à l'aide d'appels standardisés. Ce sont les appels système.
- La partie logique prend en charge tous les mécanismes de bas niveaux tel que la gestion des interruptions, les fonctions d'initialisation, la tenue de compteur (horloge), la gestion haut niveau des systèmes de fichiers. Les appels direct à ces routines sont en général impossibles de l'extérieur du noyau.
- la partie driver qui standardise autant que possible des aspects de gestion du matériel. Son rôle est principalement de raccorder des périphériques de constructeurs différents à des mécanismes standards. Par exemple un scanner SCSI doit accepter certaines commandes, les drivers du scanner prennent en charge le formatage de ces ordres en langages propres au matériel et prennent en charge le formatage des données dans le standard du logiciel demandeur (standard TWAIN par exemple).

Un noyau peut donc se présenter schématiquement comme sur la figure 4.

Nous constatons donc que le noyau subit quelques contraintes et doit permettre de réaliser un certain nombre de tâches :

- Détection des périphériques
- Utilisation des périphériques
- Tenir à jour une table d'états du système
- Garantir l'accès exclusif aux ressources

Il dispose pour cela de ressources propres en mémoire dans une zone protégée par les systèmes sérieux. Les détections de périphériques se font par tests et envois de messages types sur les bus de la machine, la liste des périphériques est pris dans une table ou constituée par décodage de messages créés sur les bus (Plug and Play). L'initialisation est soit réalisée lors de la détection, soit par des séquences plus complexes adaptées à chaque constructeur. Ces deux opérations peuvent être faites après le chargement du noyau en mémoire ou au cours du fonctionnement du système sur demande d'un utilisateur (reconfiguration, exemple de lecteur amovible).

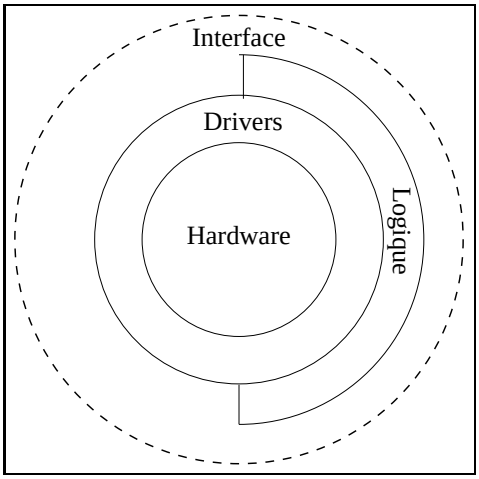


FIG. 4 – Le noyau en détail.

La garantie de l'accès exclusif aux ressources se fait par protection de mémoire et par masquage des interruptions ou par des mécanismes logiciels reposant sur des verrous en mémoire. Ces derniers sont le plus souvent des sémaphores, leur accès exclusif étant garanti par la séquentialité des opérations du processeur. Sur les systèmes multiprocesseurs symétriques (SMP), seul le mode interruptif peut être utilisé. Nous remarquons ici que les systèmes bien conçus ne restent pas en mode non interruptible durant toute l'opération d'entrée-sortie, mais juste le temps d'accéder au sémaphore de verrouillage, ainsi les opérations en multitâche peuvent continuer pendant le temps nécessaire à la synchronisation des périphériques.

La gestion des compteurs systèmes se fait soit par contribution volontaire (chaque code utilise un appel particulier pour demander une mise à jour) ou par interruption, le système déclenchant à un tic horloge certaines fonctions ou par le hard lui-même (Perf E/S) le noyau ne permettant alors que de dispatcher les informations.

**Remarques :**

On remarque donc ici que les appels système seront toujours non interruptibles. Impossible sinon de maintenir un disque à jour. De ce fait, il y a deux contraintes qui pèsent sur le programmeur de noyau :

- code non bloquant : toujours un nombre maximum de ré-essais d'une opération et toujours prévoir une procédure de fuite .
- code rapide : car un système multitâche ne le sera que si on ne perd pas trop de temps dans le noyau.

Nous remarquons également que tout système comportant des périphériques amovibles peut être planté par une éjection lors d'une phase critique. Séquence type :

- Portable avec carte réseau amovible.
- Le réseau est lent puis se bloque, pour réinitialiser la liaison,
- L'utilisateur éjecte la carte :
- La machine plante

L'arrêt brutal est dû à ce qu'elle était en phase d'appel du contrôleur de carte et ne peut déclencher l'interruption salvatrice qui rend le dit contrôleur indisponible. Dans ce cas, seule une coupure

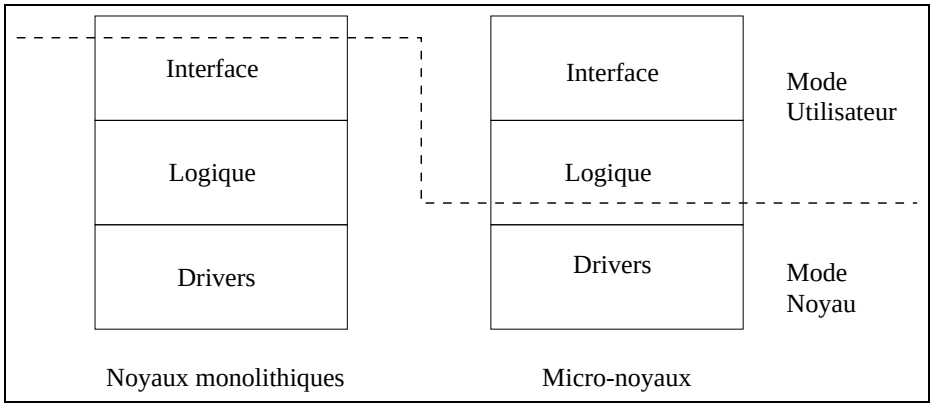


FIG. 5 – Extension des modes utilisateurs et noyau suivant l'architecture de ce dernier.

d'alimentation électrique peut sauver le monde.

### 2.2.1 Philosophie de construction

Actuellement les deux conceptions illustrées par la figure 5 s'opposent :

- le noyau monolithique : l'ensemble des fonctions décrites plus haut est stocké dans une mémoire et géré par un code tournant en mode privilégié.
- le micro-noyau : seule une partie minimale (drivers hard + leurs états) est utilisée en mode noyau, le reste est géré par des processus tournant en mode utilisateurs.

La première conception est le modèle historique. Elle garantit l'intégrité du système lors de son chargement. Ce modèle permet de gérer le plug and play par l'ajout de modules dynamiquement, mais seulement dans le cas de périphériques de type connu. Cela signifie que l'on ne peut pas ajouter d'appels système de type nouveau dynamiquement. Sous la pression de la concurrence, les constructeurs sont de plus en plus obligés de respecter des normes pour leurs appels système. De ce fait, les noyaux monolithiques sont donc de plus en plus difficiles à réaliser. L'idée qui est donc dans l'air est celle des micro-noyaux sur lesquels vont se greffer des couches spécifiques à la norme que l'on veut implémenter.

Cette conception est assez astucieuse dans la mesure où la logique du noyau est développée en langage de haut niveau. Elle peut donc aisément être portée si on normalise l'appel à l'interface matérielle. On dispose alors d'un système entièrement portable. Il est alors aisé de greffer un style de système d'exploitation particulier sur un hardware quelconque et il devient plus facile pour un constructeur de proposer une grande variété de systèmes d'exploitation. Dans le premier cas, les constructeurs de système auront juste à écrire une interface pour leur système vers le micro-noyau et dans le deuxième cas, le constructeur de matériel aura juste à réécrire le micro-noyau. On appelle les parties logicielles greffées sur un micro-noyau des sous systèmes intégraux et ce sont eux qui fourniront l'interface vers l'utilisateur, l'aspect et les qualités du système reposeront donc essentiellement sur eux.

De plus des fonctionnalités telles que la répartition et le temps réel deviennent de ce fait plus faciles à réaliser. Cette philosophie est poussée jusqu'au bout par l'emploi d'une sorte de modèle client/serveur. On déporte alors dans des processsus la gestion d'un certain nombre de tâches. Les processus ayant ensuite besoin d'accéder à des mécanismes noyaux font appel à ces sous-systèmes

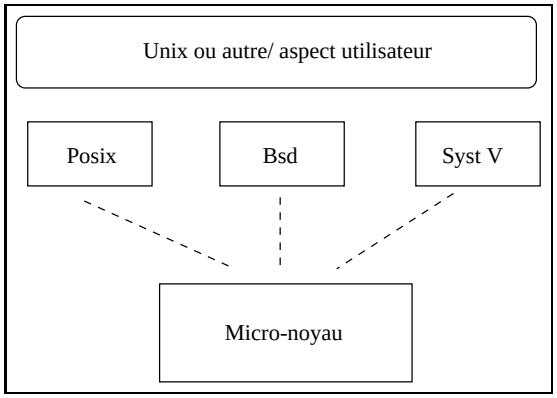


FIG. 6 – Sous systèmes intégraux greffés sur un micro-noyau.

via un serveur de procédures.

Ces deux logiques de construction s'opposent par le fait que l'une permet des performances et une sécurité maximum, l'autre garantit une portabilité maximum.

Du point de vue construction, les noyaux étaient historiquement produits par la compilation du code des drivers et des fonctions logiques, mais cela obligeait les constructeurs à fournir un compilateur C. Maintenant la philosophie est de fournir un noyau sous forme de modules objets linkables qui permettent d'adapter le système au moment du boot. La reconstruction de noyau se faisait à partir d'un fichier de commandes par invocation d'un ordre particulier (boot -r sur SUN).

Sous Solaris, les paramètres de la logique du noyau sont définis dans /etc/system. On trouve dans ce fichier les définitions du nombre d'utilisateurs, de fichiers, de processus max que peut gérer l'OS. Ces valeurs permettent un dimensionnement des tables du noyau au moment du boot.

### 2.2.2 Description des noyaux NT et LINUX

1. *Linux et UNIX en général*

   Le noyau est composé de cinq parties :

   – fs contient les descriptions des files system
   – init qui est chargé de tout initialiser
   – driver qui contient les gestionnaires hard
   – la partie spécifique à l'architecture
   – Kernel qui contient la logique de fonctionnement de l'ensemble . D'autres modules sont disponibles et chargeables dynamiquement. Il s'agit d'un noyau en C, il repose donc sur l'emploi d'une hiérarchie de fichiers C unix. Les sources sont dans /usr/src/linux et un programme de configuration permet de choisir les différents composants que l'on souhaite utiliser.

   Le noyau range les périphériques dans des tables par type de contrôleur puis par numéro de série dans le type :

   ```
   15 3 /dev/dsk/c0t0d0s3
   15 4 /dev/dsk/c0d0d0s4
   31 1 /dev/dsk/c0t1d0s0
   ```

   pour Linux :

```
3 0 /dev/hda       3 64 /dev/hd6
3 1 /dev/hda1      3 65 /dev/hdb1
3 16/dev/hda16    3 80 /dev/hdb16
22 0 /dev/hdc
```

En fait, l'expression utilisée est $CtrlIde + disk * 64 + 16partitions$.

On distingue deux types de périphérique : les blocks dev pour les E/S par block et les devices text pour les E/S mode commande.

Cela correspond à un tableau à double entrée de pointeurs de fonction dans le code C du noyau.

```
struct {
int *open(); int *close(); int *iotcl();
} kop ide_disk[][];
```

Il en va de même pour tous les autres périphériques.

Un appel système s'enregistre dans le noyau par la donnée de :

– un numéro dans asm/unisd.h
  `#define __NR__break     17`
  On constate que le système comprend déjà 190 appels système environ.
– un nom dans asm/i386Kernel/entry.S
– en incrémentant le nombre d'appels dans entry.S

On définit ensuite le corps de l'appel dans un des fichiers du répertoire /usr/scr/linux/kernel/sys.c

Exemple :

```
asmlinkage int sys_Call.Sys(arguments) { }
```

Le corps de l'appel est une fonction C classique, il faut régulièrement faire très attention à ne pas modifier involontairement de variables globales, car tout est permis et rien n'est interdit quand on est en mode noyau.

On reconstruit ensuite le noyau :

```
./configure
Make zlilo
Make instal
/sbin/lilo
```

L'appel peut ensuite être réalisé à l'aide des macros - sycall.xx où xx est le nombre d'arguments - syscall2(int, Callsys, type Arg1, Arg1, Type Arg2, Arg2)

Les appels systèmes sont ensuite exécutés par :

```
 int main(int argc, char *argv[]) {
CallSys(arg1, arg2);
}
```

Les appels systèmes dialoguent avec le processus appelant avec *put_user(value, void *add)* et *get_user(void *add)* plutôt que memcopy qui ne fonctionnera pas correctement du fait de la différence d'adressage entre le noyau et le mode utilisateur.

On peut également invoquer les fonctions du noyau avec la fonction *ioctl* qui permet d'accéder aux périphériques bas niveau, quand un driver n'est pas supporté par UNIX (ex : carte d'acquisition) :

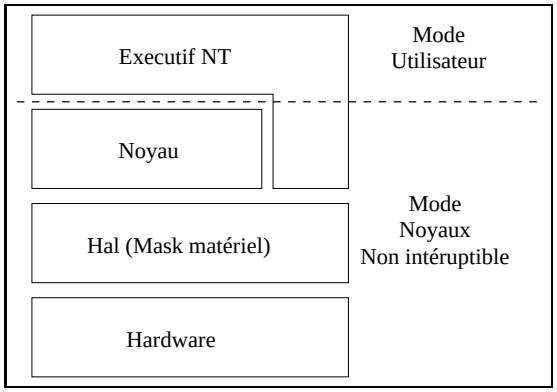


FIG. 7 – Structure des couches basses de WNT.

```
fd=fopen("/dev/daq",O_RONLY);
IOCTL(fd,RESET,NULL);
/* RESET: nom de la fonction */
/* NULL : les paramètres de cette fonction */
fclose(fd);
```

Cette fonction permet entre autre de gérer les terminaux alphanumériques sur ligne série. Cela ne marche que pour les devices en mode caractère. L'inconvénient principal de son utilisation est l'absence de normes dans la forme et les appels des fonctions. La liste des fonctions, ainsi que leurs paramètres, définies pour IOCTL est donnée par man ioctl_list.

Les modules chargeables sont manipulés par la commande modload et sont chargés sur requête d'un processus utilisateur ou sur demande modload sous linux. Leur liste est disponible dans /proc/module.

2. *description du noyau NT*

   Le système NT est basé sur une architecture à micro-noyau. Les couches basses du système sont représentées sur la figure 7 et ont le rôle suivant :

   – Le HAL est la couche d'abstraction matérielle, son rôle est de présenter au noyau et à l'exécutif une vision idéalisée de la plateforme matérielle. C'est en théorie la seule couche dont la réécriture est obligatoire pour un portage du système. Il fournit des fonctions standardisées pour la gestion des caches externes, les horloges, les bus d'E/S, les registres de périphérique, les contrôleurs DMA et les interruptions matérielles.
   – Le noyau contient les parties logiques et les compteurs nécessaires à la synchronisation du système. Il a notamment le rôle de distribuer les interruptions, de gérer le scheduling et la synchronisation des thread système, de synchroniser les processeurs et de fournir des compteurs de temps. Cette partie du système s'appuie sur une hiérarchie d'objets permettant une utilisation simplifiée des tâches bas niveau. Il s'agit en fait d'un ensemble de routines constituant un processeur modèle.
   – L'exécutif constitue le composant d'interfaces du système avec le mode utilisateur. Il est constitué de sous parties indépendantes gérant chacune des tâches distinctes telles que représentées sur la figure 8.

     LPC local procédure CALL
     Sched gestion des processus

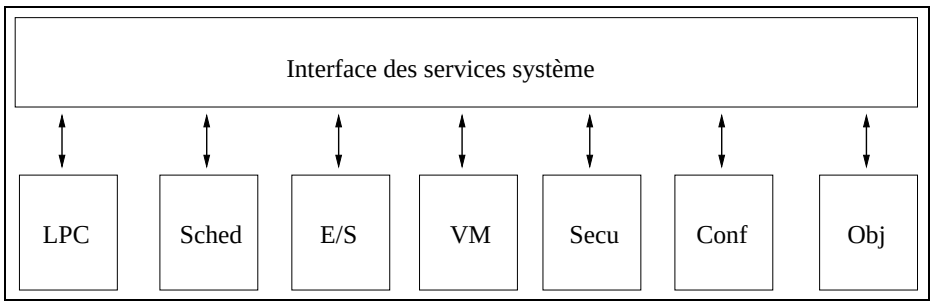


FIG. 8 – Composant de l'exécutif.

E/S gestion des entrées sorties
VM gestion de la mémoire virtuelle
Secu gestion de la sécurité
Conf gestionnaire de configuration
Obj gestionnaire des objets

Tous ces éléments communiquent via l'interface des services systèmes et toutes les communications extérieures se font via le LPC qui vérifie chaque demande auprès du gestionnaire de sécurité.

Cette séparation permet de renforcer la fiabilité du système et la souplesse de reconfiguration du système.

Le reste des couches basses du système est constitué des sous-systèmes intégraux qui dialoguent avec l'exécutif via le local procedure call.

La communication vers le HAL se fait à l'aide des fonctions *setHall()* et *getHall( )* qui permettent d'accéder aux fonctions des drivers et du matériel.

Tous les appels systèmes se passent par création d'une instance d'un objet. Cela présente l'avantage de faciliter l'initialisation des appels si la valeur par défaut est bien choisie. Cela représente par contre un mécanisme assez lourd et par conséquent consomme une quantité non négligeable des ressources systèmes.

# 3 Notion de processus

Tout d'abord, quelques définitions :

1. Programme

   C'est un ensemble d'instructions que l'on fait exécuter à un ordinateur dans le but d'obtenir un résultat précis. Ce vocable désigne aussi bien le langage source que le langage machine de cette suite d'instructions.

2. Processeur

   C'est une unité abstraite électronique dont le rôle est d'exécuter des instructions. Cette unité ne comprend que des ordres sous forme de charge électrique. En fait, il s'agit d'un petit automate électronique très sophistiqué qui permet de modifier l'état électrique de la mémoire de la machine. Le lecteur comprend donc de lui-même que la magie des ordinateurs vient uniquement de l'interprétation qu'il donne à ces états électriques.

3. Processus

   Un processus est un programme en cours d'éxecution sur un processeur, c'est-à-dire un ensemble d'instructions chargées dans la mémoire de l'ordinateur qui signifie des ordres de modification de l'état électrique de la machine. Un processus est caractérisé de plus par un état dans la machine. Cet état est simplement l'instruction qu'il est en train d'exécuter assortie à toutes les mémoire dont il dépend.

4. Thread

   A ces trois notions s'ajoute la notion moderne de Thread (ou tâche) qui est en fait une unité d'exécution d'un programme, c'est-à-dire un triplet instructions à exécuter, états des registres d'un processeur et état d'une sous partie de la mémoire de la machine. Un processus comporte au minimum un thread et dans le cas où il en comporte plusieurs, il s'agit d'unités d'exécution concurrentes de sous parties d'un même programme.

La conclusion de cette partie est qu'il ne faut pas confondre tous ces éléments et que finalement on décompose l'activité d'un ordinateur en tâches autonômes les moins dépendantes possibles. Les systèmes seront donc caractérisés par leur possibilité d'exécutions quasi simultanées de plusieurs processus et de leurs threads. Ces possibilités et un bon ordonnancement de celles-ci conditionneront la plupart des utilisations possibles d'un système.

## 3.1 Image mémoire des processus

Les caractéristiques des processus dépendent des capacités du système qui les fait tourner. Si nous envisageons facilement la liaison entre la notion de multitâches et la notion de processus, les liaisons entre l'aspect multi-utilisateurs et processus sont en général plus floues.

Pour fonctionner de façon sûre, un système doit distinguer des objets de propriétaires différents, dans le but de définir des flux d'information fiables dans la machine. Ces flux devront être correctement définis par leurs origines et leurs règles de modification. La solution unanimement adoptée à notre époque est le cloisonnement en mémoire des différents processus assortis de mécanismes de communication coopératifs et protégés.

Le modèle actuel pour l'espace mémoire d'un processus est celui représenté sur la figure 9.

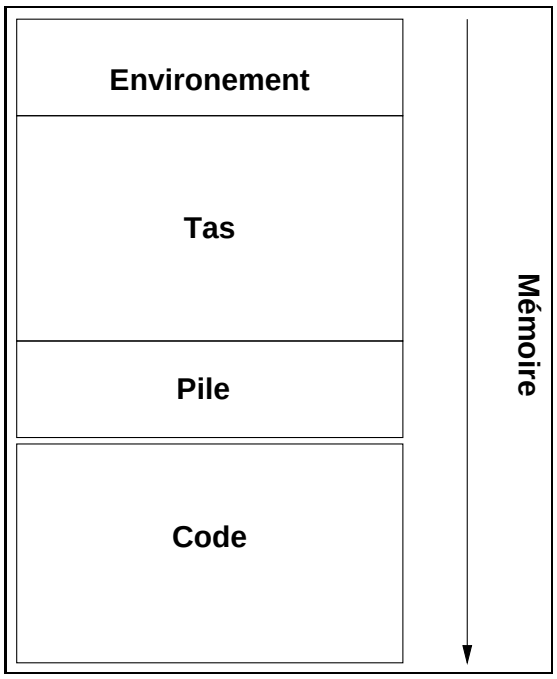


FIG. 9 – Image mémoire d'un processus

Chaque processus possède donc :

1. Un espace d'environnement

   Cette partie de mémoire contient tout ce que le programme doit connaitre du reste du système pour s'exécuter correctement. Il contient des informations sur l'emplacement des ressources du système (par exemple le PATH du shell). Cet espace est dépendant du processus, il a la possibilité de modifier les valeurs mais ces modifications seront ignorées par le système à la fin du processus.

   Sous Unix, il est constitué pour l'essentiel de chaînes de caractères de la forme VARIABLE="value" héritées lors de la création du processus par le système.

   Sous NT, il est constitué de ces mêmes chaînes de caractères plus des morceaux de la base de registres. La base de registres est en fait une suite de variables organisées sous forme arborescente permettant à un processus démarrant de modifier son état de façon à être conforme aux attentes des utilisateurs de la machine. Par exemple, l'application Word reçoit des clefs lui permettant de déterminer quel dictionnaire elle doit utiliser au démarrage de la machine.

   La convention actuelle est qu'en C, l'adresse de l'espace d'environnement est communiqué au programmeur en troisième paramètre de la fonction main :

   ```
   int main(int argc, char *argv[], char *envp[]) {}
   ```

   Le type utilisé dans cet exemple est celui pour UNIX. La librairie du système fournit en général un ensemble de fonctions permettant de manipuler cet espace.

2. Un tas (Heap in UK)

   Le tas est une partie de mémoire ou un processus range les informations qu'il manipule. Il n'a pas d'organisation intrinsèque particulière. C'est le code du programme qui détermine l'orga-

nisation qu'il souhaite lui donner. Cette partie est réservée à un processus donné mais tous les threads la partage. Là réside la source d'erreur principale de la programmation multithread.

3. Une pile (Stack in UK)

   La pile permet de stocker les variables temporaires sur lesquelles le programme travaille et les valeurs de retour des appels que génère le programme. C'est une structure de cases mémoires contigues caractérisée par deux pointeurs : BS la base de la pile, HS le haut de la pile. La valeur importante est le HS qui permet de localiser la prochaine donnée à lire.

   Toutes les opérations récursives faites par le processus seront gérées grâce à la pile et son extension est contrôlée par un pointeur MS qui contient l'adresse maximum pouvant être utilisée. La plupart des systèmes contrôle que la pile d'un processus ne puisse pas déborder de la valeur MS de façon à éviter les risques de corruption du tas.

   Cette pile joue un rôle fondamental pour le multitâche car la simple retenue de HS et de l'adresse suivante d'exécution permet de relancer un processus dans l'état exact où il se trouvait avant un arrêt.

4. Un code

   Le code d'un processus est en fait le code du programme qui est en cours d'exécution. Ce code est simplement une suite d'instructions en langage machine que le processeur exécute séquentiellement.

   Ce code est en général découpé par le programmeur en plusieurs parties qui peuvent être contenues dans des librairies du système. Le gestionnaire de processus est chargé de mettre à jour les adresses spécifiées dans le code de façon que tous les sauts correspondent à quelque chose et donc que les appels des fonctions des librairies puissent se faire.

   Au code contenu dans le ficher binaire du programme sont adjoints des parties de codes de librairies dans le cas où le système exploite un système de librairies dynamiques (les fameuses DLL de windows). Cette résolution est faite au moment du chargement du programme et le programme est ensuite autonome. Sur les systèmes qui manquent de mémoire, le code peut être segmenté en parties autonômes chargées successivement que l'on appelle des overlays.

   En général, les systèmes sérieux interdisent au programme de modifier son propre code pour éviter des effets de bord génants.

La raison principale de ce découpage est que le processeur exécute les instructions séquentiellement et qu'il n'y a donc pas de possibilité de loger des données au milieu du code.

L'image mémoire est, sur les systèmes sérieux, uniquement accessible au processus qu'elle représente. Seul, un appel système peut modifier la zone de code. Ceci permet de gérer le cloisonnement nécessaire à un fonctionnement harmonieux et prévisible du système.

Les processus possèdent de plus une zone propre dans les tables de description du système. Cette table est en général située dans la mémoire utilisée par le noyau du système et donc de taille définie au moment du démarrage du système.

Une description de cette table (struct task_struct) peut être trouvée sous linux dans les sources du système /usr/src/linux/include/linux/sched.h . Dans le cas de Windows NT, il s'agit d'une classe C++ équivalente mais qui contient également les fonctions de manipulation du processus. Dans ce cas, le cloisonnement décrit plus haut s'étend y compris aux appels systèmes du noyau qui ne peuvent manipuler les processus que par l'utilisation de ces méthodes.

Les fonctions du système qui manipulent les processus (ps) utilise une description simplifiée qui est par exemple :

```
struct _glibtop_proc_uid
{
        u_int64_t flags;
        int uid,                /* user id */
                euid,           /* effective user id */
                gid,            /* group id */
                egid,           /* effective group id */
                pid,            /* process id */
                ppid,           /* pid of parent process */
                pgrp,           /* process group id */
                session,        /* session id */
                tty,            /* full device number of controlling terminal */
                tpgid,          /* terminal process group id */
                priority,       /* kernel scheduling priority */
                nice;           /* standard unix nice level of process */
};
```

Vous remarquerez que toutes ces structures comportent l'identification du propriétaire du processus. Il en va de même pour tous les systèmes sérieux et ceci permet de gérer effectivement l'aspect multi-utilisateur du système en introduisant des règles minimales d'accès aux ressources du système. Les uid et gid d'unix sont remplacés sous NT par des objets ACL[1] plus complexes, permettant par exemple d'attribuer plusieurs propriétaires à un processus.

Cette identification des utilisateurs permet de définir à quelles ressources le processus pourra accéder et les attributs des ressources (fichiers résultats) que créera le processus durant son exécution. Dans tous les systèmes multitâches, un processus est identifié par un numéro unique qui n'est rien d'autre que l'indice du descripteur du processus dans la table du noyau.

Les systèmes UNIX fournissent deux choses intéressantes pour l'étude de l'image mémoire des processus : le système de fichiers /proc qui permet d'avoir un fichier pointant sur tous les éléments d'un processus :

```
[gouinaud@garp 680]$ ls -al /proc/680

dr-xr-xr-x   3 gouinaud gouinaud        0 Oct 19 12:30 ./
dr-xr-xr-x  67 root     root            0 Oct 19 11:46 ../
-r--r--r--   1 root     root            0 Oct 19 12:32 cmdline
lrwx------   1 root     root            0 Oct 19 12:32 cwd
-r--------   1 root     root            0 Oct 19 12:32 environ
lrwx------   1 root     root            0 Oct 19 12:32 exe
dr-x------   2 root     root            0 Oct 19 12:32 fd/
pr--r--r--   1 root     root            0 Oct 19 12:32 maps|
-rw-------   1 root     root            0 Oct 19 12:32 mem
lrwx------   1 root     root            0 Oct 19 12:32 root
```

[1] Access Control List

```
-r--r--r--   1 root     root            0 Oct 19 12:32 stat
-r--r--r--   1 root     root            0 Oct 19 12:32 statm
-r--r--r--   1 root     root            0 Oct 19 12:32 status
```

et les fichiers core qui sont un vidage de l'image mémoire sur disque, soit par le système du fait d'une erreur de protection mémoire, soit du fait de l'utilisateur par envoi du signal SIGTRAP.

## 3.2 Cycle de vie d'un processus

La vie d'un processus commence par l'exécution d'une instruction de chargement et finit par la libération des ressources qu'il occupe. Un bon système est au minimum un système qui gère parfaitement ces deux événements de façon à ne pas recourir à de fréquents reboots ou des visites tout aussi fréquentes chez le marchand de mémoires du quartier.

Les étapes importantes de la vie d'un processus sont résumées ci-après.

### 3.2.1 Il nait, ou le lancement

La naissance d'un processus se passe de la façon suivante :

- Le système attribue une zone de description dans la table du noyau
- Le code du processus est recopié en mémoire
- Les appels de librairie sont résolus
- La place pour le tas réservée
- Le pointeur de pile est mis à zéro
- L'environnement est dupliqué dans l'espace du processus
- La première instruction du programme est exécutée

Lors de la création du descripteur noyau, les descripteurs de sécurité (par exemple : Utilisateur, group) sont initialisés avec les valeurs had hoc. Un certain nombre de règles de fonctionnement du processus sont également ajustées tels que les espaces mémoire maximum utilisables ou les quotas de temps CPU.

Toutes les autres actions sont liées à des modes de fonctionnements particuliers des systèmes d'exploitation.

Sous unix, la création d'un processus se fait en clonant le processus en cours par l'appel de la routine *fork()* . Cette routine duplique le processus courant (code, tas, pile) et poursuit l'exécution des deux clones de façon concurrente. Les seules différences sont que *fork()* renvoie au père la valeur du PID du père et au fils, 0. Un des deux processus peut ensuite utiliser la routine *exec()* qui permet de remplacer le code du processus par un nouveau et donc exécuter ainsi un nouveau programme. Ces deux actions peuvent être conbinées en une seule grâce à la routine *system()* . Le process fils hérite par défaut des attributs du père (uid, gid) qui peuvent être changés par l'appel des fonctions *setuid()* et *setguid()*. Lors de l'utilisation de la fonction *exec()*, les attributs uid et gid prennent la valeur de ceux du fichier programme si celui-ci a le bit setuid positionné à 1.

Sous NT, deux fonctions sont disponibles *_spawn()* qui créé un nouveau process avec un nouveau code et *_exec()* qui fonctionne exactement de la même façon que sous UNIX. La création d'un processus s'accompagne sous NT du démarrage des sous systèmes intégraux dont il a besoin (MSDOS

par exemple) et l'héritage des droits se fait par héritage des objets ACL du père. L'environnement du processus est un objet copie hérité de la base de registre de la machine. L'exécution du processus démarre par le lancement d'un premier thread qui débutera par la première instruction du programme.

### 3.2.2 Il vit, ou en cours d'exécution.

Durant la vie du processus, les instructions du programme sont égrainées une à une par le processeur. Sur les systèmes multitâches, les processus sont périodiquement arrêtés et relancés.

Dans le cas du multitâche coopératif, chaque process décide du moment où il a suffisamment tourné et peut redonner la main au système d'exploitation qui détermine alors quel processus doit être réactivé. Dans ce type de système, le plantage d'un processus conduit irrémédiablement au blocage du système. Les versus modernes de ce genre de système réservent une interruption, souvent associée à une combinaison de touches, permettant de forcer le transfert de la main à des routines du système d'exploitation chargées de gérer les cas douloureux.

Dans le cas du multitâche préemptif, c'est le système qui décide à quel moment un process doit être activé ou mis en sommeil. Une interruption se déclenche périodiquement et appelle les routines de scheduling du système.

Ces routines choisissent alors le process qui sera maintenant exécuté et provoque son réveil. Le réveil d'un processus se fait en rechargeant son contexte précédent dans la pile des processus et l'exécution de l'instruction suivant la dernière avant l'interruption précédente. Cette opération n'est pas innocente, car il s'agit de recopier un grand nombre de blocs de mémoire et parfois de recharger des pages à partir du disque. Le changement de contexte a donc un coût dont il faut tenir compte dans toute tentative de parallélisation.

Les systèmes multitâches peuvent aussi interrompre l'exécution d'un processus pour les raisons suivantes :

- Violation de droit. Dans ce cas le process est interrompu définitivement et une routine de gestion d'erreurs doit être déclenchée.
- Attente d'une ressource, principalement E/S que ce soit sur un fichier utilisateur, un périphérique ou une page mémoire. Le process est alors relancé quand sa demande est satisfaite.
- Certains événements matériels. Dans ce cas c'est toute la gestion des processus qui est interrompue et le système reprendra ensuite celui qui était en cours lors du declenchement de l'interruption matériel.

Tous les systèmes appliquent ces consignes. Il existe par contre de grandes disparités dans la façon dont ils choisissent le processus suivant à exécuter. Cette ordonnancement fait l'objet d'un paragraphe de la section 3.4.

Afin de réduire les temps d'interruption des processus et de fluidifier le multitâche, un certain nombre de systèmes implémentent des threads qui sont en fait plusieurs unités d'exécution d'un même processus. Les différents threads sont schedulés par le système de façon concurrente, ont chacun leur pile d'exécution mais partagent par contre le même tas. En pratique, cela signifie qu'au moins deux parties différentes du code sont exécutées en même temps.

Cette technique permet d'alléger les tâches de communication du système par utilisation du tas commun par les threads d'un même processus. Cela semble faciliter la vie du programmeur, mais se paye par une moindre fiabilité et la nécessité que le programmeur implémente lui-même les méca-

nismes nécessaires à garantir la cohérence des données de l'ensemble du processus.

Le multithreading permet aussi de limiter le coût des changements de contexte et donc limiter l'overhead de charge induit par la gestion du multitâche.

Il présente aussi l'avantage de ne pas arrêter complètement les processus dans les cas d'arrêt cités plus haut. Ceci a un grand intérêt pour les systèmes fortement interactifs et permet à leurs applications de réaliser de façon transparente pour l'utilisateur leurs opérations d'entrée-sortie.

Le multithreading n'est par contre pas très efficace dans son utilisation sur des systèmes multiprocesseurs. En effet, afin d'accélérer les échanges entre mémoire et processeur, toutes les plateformes matérielles utilisent des systèmes de cache mémoire. Or, quand deux threads d'un même processus tournent sur deux processeurs différents, maintenir la cohérence de ces caches est une opération très lourde. On associe donc en général à un process une affinité de processeurs qui vise à maintenir tous les threads d'un même process sur un processeur donné.

### 3.2.3 Etat de santé d'un processus

Le système maintient en permanence l'état de santé des processus. Suivant le type de système, ces états peuvent signifier des choses différentes mais on retrouve le plus souvent.

- R (runable) prêt à être exécuté
- S (sleeping) endormi
- D sommeil ininterruptible
- T (traced) arrêté ou suivi
- Z (zombi) ou défunt

Dans tous ces états, le seul qui peut poser problème est le Z. Dans ce cas, le process est interrompu définitivement, mais reste résident en mémoire et occupe encore des ressources. Ceci est en principe dû au fait que le process s'est terminé mais n'a pas pu informer son père ou le système de son décès. Outre le fait qu'un nombre important de process zombis finiraient par consommer trop de ressources, le problème vient surtout du fait que les ressources matérielles utilisées par le process restent verrouillées. Par exemple, il sera impossible de démonter un système de fichiers sur CDrom tant que tous les process ayant ouverts des fichiers dessus ne sont pas terminés.

## 3.3 Allocation des ressources

Un process consomme durant son exécution des ressources, principalement de la mémoire et de la CPU. Il accède aussi au matériel, via le noyau, et se pose donc les questions de garantir le fonctionnement du système contre une surconsommation d'un ou plusieurs processus.

La gestion des ressources allouées peut être confiée au noyau, à un process du système ou aux process eux-mêmes. Dans ce dernier cas, le système alloue les ressources au système mais ne se préoccupe pas de leur libération tant que le process fonctionne. Quand il est informé du décès de celui-ci, il considère alors que les ressources sont libérées. Par exemple, la plupart des Unix alloue de la mémoire à un process qui en demande, mais ne considère cette mémoire comme désallouée que quand le process se termine. Les désallocations faites par le process lui-même sont donc uniquement locales à la mémoire du process et lui permettent juste de réallouer plus rapidement dans son espace d'adressage.

Ceci permet à un process bien fait d'allouer toute la mémoire dont il aura besoin durant son exécution et par conséquent de mieux garantir son fonctionnement. Le défaut de cela est que si un process s'alloue trop de ressources, il peut mettre en péril les possibilités de multi-exécutions du système. Il y a donc en général un système de quota implémentant les limites suivantes :

- Maximum de mémoire allouable, en général deux limites : une pour la pile et une pour le tas.
- Max de temps processus (CPU time limit), il s'agit bien évidemment du temps de calcul effectif et non du temps vu du point de vue de l'utilisateur.
- Maximum de fichiers ouverts
- Maximum de threads pour un même process

La valeur de ces limites et leur implémentation sont très variables suivant les systèmes. En général, elles sont fixées par l'administrateur et inscrite dans l'environnement de chaque process.

Il y a quelques années, les systèmes étaient munis de queues d'exécution qui définissaient des classes de process et permettaient une gestion fine de ces limites. Le renforcement récent de la puissance des ordinateurs a fait passer ces notions au second plan bien qu'elles permettaient une gestion souple des machines plus facile à faire accepter aux utilisateurs que la technique tout ou rien actuelle.

### 3.3.1 Décès d'un processus

Un processus décède suivant deux schémas possibles :

- Il est tué par le système (SIGKILL sous Unix).
  Dans ce cas le système oublie simplement de le scheduler, libére les ressources (fichiers et mémoire) qu'il occupait et informe les fils de ces processus de son décès (SIGCHLD 20 sous unix). Dans les systèmes intelligemment conçus, les orphelins ainsi créés se voient doter d'un tuteur (INIT sous UNIX), pour qu'ils puissent informer leur père de leur décès.
- Il décide de s'arrêter soit de lui-même, soit sur requête du système (envoi d'un signal SIGEND sous Unix).
  Dans ce cas, le processus appelle par défaut la routine exit ou la routine que le programmeur a programmé pour l'arrêt. Dans ce dernier cas, la routine d'arrêt réalise toutes les opérations de mise en cohérence des ressources et appelle à son tour exit() qui indique alors au système qu'il peut procéder au nettoyage décrit dans l'alinéa précédent.
  La routine d'arrêt personnalisé est précieuse pour pas mal de processus qui gèrent soit des fonctions systèmes (sendmail, automounter) serveur et indispensable pour les serveurs de bases de données. En effet, elle permet la mise en cohérence des données et des requêtes reçues et par conséquent de garantir l'intégrité du système. Le risque que l'on prend est par contre que cette routine plante aggravant encore les problèmes pour lesquels on a décidé l'arrêt du process.

En cas de disfonctionnement, il faut donc toujours se poser la question de quel type d'arrêt on veut obtenir. Si le process commence à corrompre les données, il faut l'arrêter brutalement, mais si on souhaite qu'il se termine, par exemple, pour pouvoir arrêter la machine, il faut lui envoyer un signal SIGEND puis un signal SIGKILL dans le cas d'un système UNIX.

Tout cela fonctionne en général pas trop mal en instantané mais pas toujours à long terme. Les principaux disfonctionnements ou problèmes sont :

- Le système ne possède pas de fonction SIGKILL ou bien elle ne fonctionne pas de façon satisfaisante. C'est le cas de Windows NT et cela pose pas mal de problèmes sur les versions

multi-utilisateurs (Winframe ou WTSE). Ceci est principalement du à la gestion des processus par un sous système intégral fonctionnant lui-même en mode utilisateur qui a donc bien du mal à se debarrasser des intrus.

– Le process accepte de s'arrêter mais certaines ressources ne sont pas libérées correctement. En principe, ce genre de chose est dû à une structure de gestion de mémoire trop complexe ou comprenant des zones partagées dont la libération est laissée à la charge des process eux-même (shared memory). C'est le cas type de la fuite de mémoire qui va conduire à saturer la zone de swap à plus ou moins long terme. Cette saturation sera soit du fait de la fragmentation, et réduira les performances, soit par simple manque de place et donc interdira la création de nouveaux process.
– Le process s'arrête mais la fonction de vidage mémoire vers disque est buggée ou consomme trop de ressources systèmes. Dans ce dernier cas, le ralentissement du système sera manifeste mais non létal. Dans le premier cas, les ressources occupées resteront verrouillées.

Dans tous les cas, le programmeur doit connaître les disfonctionnements dont est atteinte la plateforme sur laquelle il développe et doit en tenir compte dans son travail. Dans la plupart des cas, la seule technique miracle est la réinitialisation périodique du système (reboot) qui, correctement gérée, est très efficace bien que peu satisfaisante pour l'esprit.

## 3.4 Vie en société

Cette partie décrit la vie des processus mais du point de vue du système et de leurs interactions.

### 3.4.1 Sécurité

La gestion de la sécurité pour les processus repose principalement sur le cloisonnement mémoire. Il s'agit aussi bien de garantir la sûreté de fonctionnement du système que l'inviolabilité des données qu'il manipule.

Les modalités pratiques sont :

– Les zones de mémoire de travail du processus (pile et tas) ne sont en général modifiables que par lui même. Le code n'est pas modifiable en cours de fonctionnement.
– Chaque processus a un propriétaire et ne peut interagir qu'avec les objets sur lesquels il possède des droits suffisants (fichiers notamment).
– Les héritages de droits lors de la création des processus sont clairement définis et vont dans le sens de l'apauvrissement. Un processus ne peut en général pas acquérir de droit particulier pendant son fonctionnement et ne peut pas non plus augmenter ses privilèges.

Bien evidemment, toutes ces notions sont implémentées de façon différente suivant les systèmes.

Sous Unix, un processus a un UID, GID et un exécutive UID,GID. Ces deux valeurs conditionnent les droits d'accès sur les fichiers et les périphériques. Le système de bas niveau est basé sur des comparaisons bit à bit des numéros d'utilisateur masqués par les droits des fichiers sur le propriétaire, le groupe et le reste du monde :

| UID * droit U * bit OP + GID * droit G *bit OP + Droit O *bit OP = Droit total |
|---|

Ce système est très simple et donc très rapide.

Lors de la création de ressources, les UID et GID du process sont hérités et les objets créés reçoivent un droit par défaut défini par la valeur de la variable UMASK [2] du système. Ces droits peuvent être modifiés par un positionnement des bits Setuid des répertoires qui permettent de fixer le propriétaire des fichiers créés.

Sous NT, les droits sont définis par des objets ACL[3] et sont hérités du process père. Ces objets sont des listes d'utilisateurs, de groupes et de droits associés pris dans toutes les combinaisons (LIRE/ECRIRE/MODIFIER/CREER/DETRUIRE) et ont donc une signification différente suivant les ressources.

La vérification des droits d'accès est comparable à ce qui se fait sous UNIX à l'exception qu'elle est beaucoup plus complexe. Lors de la création des fichiers, ceux-ci héritent des ACL du process, mais aussi des ACL du répertoire où ils sont créés, ce qui peut conduire à créer des trous de sécurité difficilement maîtrisables.

### 3.4.2 Partage du temps CPU ou ordonnancement des processus

L'ordonnanceur du système constitue souvent la couche la plus déterminante du système pour ses utilisations potentielles. Cette couche doit gérer la répartition des processeurs entre les diverses tâches qu'il doit accomplir. Elle doit pour cela réquisitionner le processeur pour le traitement des interruptions, assurer l'arrêt et le réveil des processus de la façon la plus transparente possible. La manière dont le processeur fait ces choix est appelée stratégie d'ordonnancement. Il en existe de nombreuses, mais elles visent toutes à respecter le cahier des charges suivant :

- Equité. Chaque processus reçoit sa part de temps processeur.
- Efficacité. Temps de réponse minimum du système et consommation d'un minimum de CPU pour les changements de contexte.

Elles visent toutes à optimiser le rendement du processeur et le nombre de travaux que la machine peut traiter en un temps donné. Ces stratégies doivent tenir compte de l'ensemble du système et de la durée prévisible des diverses tâches que chaque process (accès mémoire, accès disque, ...) requiert.

Les principales stratégies sont :

- Ordonnancement sans réquisition.
  On laisse les process se terminer avant de lancer le suivant. Très grand rendement pour les process de calcul mais capacité multi-tâche très limité. Les temps d'entrée-sortie sont par contre perdus. Les OS CRAY fonctionnent partiellement comme cela.
- Ordonnancement circulaire.
  Chaque processus a un quantum de temps et ils sont rangés dans une file circulaire. C'est donc chacun son tour. La durée de commutation est réduit au minimum, mais on ne tient pas compte de ce que fait chaque process. D'autre part, plus le temps entre commutation est court, plus on perd de temps dans les changements de contexte et plus il est long, plus les capacités multitâches sont limitées.
- ordonnancement avec priorité.
  Chaque processus se voit doté d'une priorité $P_i$ et est exécuté quand elle tombe à zéro. Il se voit doté également d'un facteur de nice $N_i$. A chaque changement de contexte, on diminue toutes

[2] faire man umask pour avoir plus de détails
[3] Access Control List

les priorités $P_i$ de 1 et on exécute celui qui a la plus petite valeur, on ajoute ensuite à celui qui vient d'être exécuté la valeur $N_i$. Les facteurs $N_i$ peuvent être alloués par l'utilisateur ou par le système suivant des critères qui lui sont propres et en fonction de ce que fait le process. C'est la technique utilisée par la majorité des UNIX.

- Ordonnancement avec files multiples,
  Il s'agit d'un ordonnancement avec priorité, mais avec plusieurs classes de priorité impliquant des fréquences et durées de fonctionnement différentes. Les processus reçoivent de plus des limites différentes suivant leur classe. Les process changent de classe en fonction de paramètres de fonctionnement du système. C'est la technique de fonctionnement privilégiée de VMS.
- Ordonnancement le plus court d'abord
  Cette technique est très adaptée à la mise en oeuvre de système accomplissant toujours les mêmes tâches dont on connait à l'avance la durée. Ce n'est en général jamais le cas, sauf en automatique, mais ce domaine exige en général le temps réel. Cette technique est par contre celle qui garantit le temps minimum d'exécution d'un ensemble de tâches. Elle est employée dans quelques systèmes sous forme de l'adging qui consiste à utiliser des priorités décroissantes en fonction de la durée des process.
- Ordonnancement garanti
  Le temps processeur est répartie entre chaque processus de facon constante. C'est -à-dire que le temps d'exécution est fixe quand le nombre de process est constant. Cela se heurte à la durée des opérations d'entrée-sortie et à la gestion des interruptions matérielles. Cette répartition peut aussi se faire par utilisateur. Pour être réellement efficace, cette technique doit être employée avec un nombre maximum de processus acceptables sinon elle devient un simple ordonnancement circulaire.

Les systèmes temps réel restent des bêtes à part dans la gestion de leurs process. Ces systèmes ne visent pas, contrairement aux autres, une efficacité maximum mais une garantie de la date d'arrivée de certains évènements. La technique la plus efficace dans ce domaine est l'ordonnancement garantie sans interruption possible des créneaux par le matériel. Le système GESPAC implémente cette stratégie et permet de gérer des appareils de tracés (flasheuse SECMA) et de découpes de façon optimale. Ce type de système apporte de plus une très grande sécurité du fait du cloisonnement total entre les tâches. Le seul point que les systèmes temps réel ont du mal à gérer sont les E/S et là, rien n'est bien clair.

Toutes ces belles stratégies se heurtent à un phénomène malheureusement trop répandu à notre époque qui est la surconsommation de ressources du fait de la piètre qualité des développements logiciels. Les coûts du matériel ayant considérablement baissés, le coût du développement logiciel est tiré vers le bas et donc l'optimisation de l'applicatif est devenue une priorité très secondaire. Nous ne pouvons donc que déplorer :

- La fantastique puissance des processeurs actuels n'a pas été explorée dans le sens de rendre temps réel les tâches interactives des systèmes. L'utilisateur se trouve donc toujours confronté aux mêmes temps de latence.
- La baisse du coût de la mémoire vive n'a pas conduit à développer des systèmes où les process commettant de nombreux défauts de page soient fortement pénalisés par les systèmes. Cela aurait moins le mérite de forcer les développeurs à n'utiliser cette ressource qu'avec parcimonie.
- La baisse de coût des processeurs bas de gamme (486) n'a pas conduit au développement de systèmes utilisant des processeurs dédiés à des tâches systèmes.

Force est de constater que les fonctionnalités de fond n'ont pas connues une croissance faramineuse. Par exemple, le temps de connexion sur un système SUNOS SPARC2 à 40 Mhz est de l'ordre d'une minute alors que celui d'une SOLARIS ULTRA 5 à 300 MHz est de 1 minute 30s. Il est vrai que le bureau offert par le deuxième est un peu plus joli :( .

### 3.4.3 Communication interprocessus

Les systèmes de communication entre processus permettent de faire circuler les informations dans la machine par utilisation de la mémoire au lieu de limiter les échanges à l'utilisation de fichiers.

Il y a cinq types de communication intéressants : les signaux, les IPC, les streams, les sockets et les RPC. Seuls les deux premiers sont des mécanismes bas niveaux et feront donc l'objet d'un développement ouvert :

- les signaux, ce sont des interruptions logicielles que peut déclencher un processus dans un autre. Leur mécanisme d'utilisation sous UNIX est très simple et consiste simplement à armer l'interruption à l'aide de l'ordre *signal* et d'envoyer ensuite le signal à l'aide de l'ordre *kill*.
  Il existe deux types de signaux : ceux à destination d'un process traités par lui-même et ceux traités par le noyau. Les premiers servent à effectuer des traitements sur interruptif souvent urgents ou à forcer la synchronisation de processus. Les deuxièmes permettent d'agir sur un process via le scheduleur pour qu'il arrête, relance ou détruise un process.
  Il faut noter que les signaux sont interruptifs et donc un process ne peut pas traiter un signal quand il est en cours de traitement d'un autre. Ignorer ceci peut conduire à des situations de deadlock désagréables.
- les IPC
  On regroupe sous l'appellation IPC (Inter Processus Communication) trois types d'objets : les sémaphores, les messages et les shared memory (mémoire partagée).
  * Les sémaphores. Il s'agit d'entiers, que maintient le système, sur lesquels les process ont le droit d'espionner, d'augmenter ou de diminuer leurs valeurs. Ce que garantit le système est que deux process ne peuvent changer la valeur du sémaphore simultanément. Certains systèmes proposent une fonctionnalité dite spinlock qui sont des sémaphores très bas niveau permettant au process de prévenir les interruptions.
  * Les messages. Il s'agit de files d'attente sous forme de files FIFO maintenues par le système permettant une implémentation facile de tous les systèmes nécessitant des traitements ordonnancés (requêtes d'impression). Les process peuvent ajouter des messages à la file, les supprimer et les reçoivent séquentiellement. Ils peuvent les lire de façon bloquante ou à la volée.
  * Les shared memory. Il s'agit de zones mémoires que maintient le système et sur lequel il garantit que personne n'écrira simultanément. Les process peuvent en créer, les écrire et les lire. Elles sont caractérisées par un numéro qui les différencie et une taille maximum. Il faut prendre garde au fait que les shared memory sont bidirectionnelles mais ne fournissent aucun système de bufferisation et que la même zone sert à communiquer dans les deux sens. Il faut donc absolument synchroniser les processus, avec des sémaphores par exemple, lors de leur utilisation ou utiliser des sockets locales.

  Les IPC sont présents dans les normes POSIX et System V. Ils sont donc de fait exploitables sous WNT et sous UNIX. Dans ce dernier cas, il faut juste vérifier que cette fonctionnalité est

activée dans le noyau.

Les IPC sont en général protégés par un système tenant compte de l'utilisateur du groupe et d'ACL dans le cas des systèmes qui en ont. Deux choses posent problème quand on les utilise : qui les crée et qui les détruit. il faut donc implémenter des modes de numérotation efficaces. Man *ipc* vous en dira plus.

Actuellement la plupart des applications implémente la communication entre leurs propres processus en utilisant des sockets. Cette approche s'explique par la possibilté de déporter facilement sur le réseau, mais se paye par de moindres performances sauf sur les systèmes BSD où les sockets sont le seul mécanisme de communication bas niveau du système.

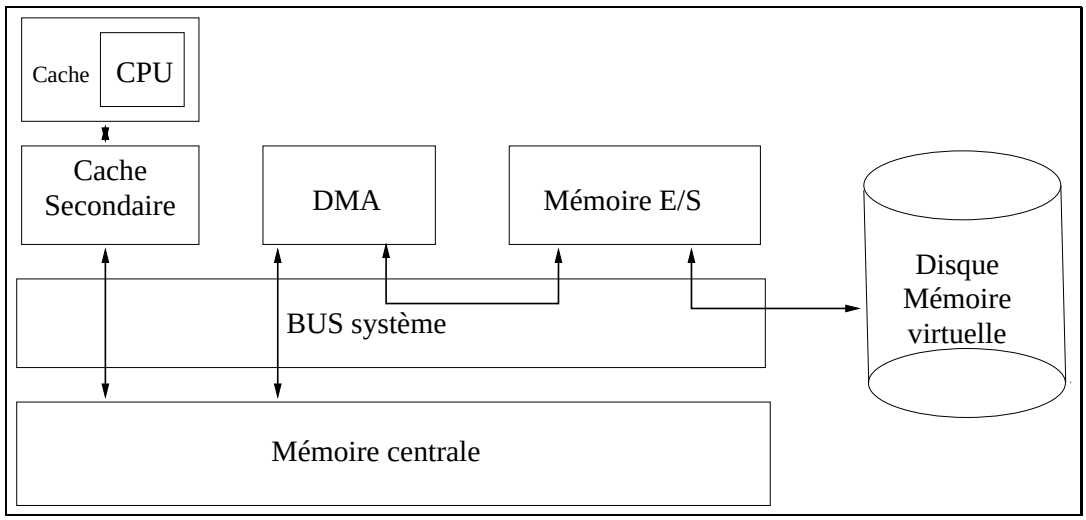


FIG. 10 – Les mémoires d'un système.

# 4 gestion de la mémoire

Sur les systèmes mono-tâches mono-utilisateurs, cette gestion consiste juste à indiquer à l'unique processus le début et la fin de la mémoire physique. Dans le cas des systèmes multi-tâches, les process doivent se partager cette ressource en conservant l'intégrité des données du système et de chaque process.

## 4.1 Type de mémoires

[4]

Les différents types de circuits apparaissant dans les mémoires et leurs gestions sont représentés sur la figure 10. Ces différents systèmes ne sont pas tous présents sur un même système mais leur vocation est toujours la même : stocker les informations de fonctionnement des processus du système à un instant donné et les rendre disponibles, le plus rapidement possible, à l'unité de calcul (CPU).

La description de ces divers éléments est la suivante :

- Bus système : c'est sur ce canal que va circuler l'ensemble des informations de la machine. Les débits actuels sont de l'ordre de la centaine de Mega bits par seconde. De leur nombre et de leur caractéristique dépendent en grande partie les performances du système.
- Cache interne : mémoire cache très rapide de petite taille (32 Ko sur un ULTRA SPARC) contenant les instructions et les données en cours. Elle est propre à chaque processeur. Dans les bons processeurs, elle est cadencée à la vitesse du processeur, ce qui signifie qu'elle est capable de s'écrire ou de se lire à la même vitesse que le processeur peut fournir ou avaler les instructions.
- Cache secondaire : mémoire cache beaucoup plus rapide que la mémoire centrale contenant les données d'exécution des process en cours. Elle peut être partagée par plusieurs processeurs. Elle doit, dans l'idéal, être cadencée à la vitesse du bus du processeur ou à un de ses sous

[4] cette partie n'est là que pour mémoire et a davantage sa place dans un cours d'architecture des ordinateurs.

multiples.

- Mémoire centrale : c'est la mémoire principale de l'ordinateur. Elle contient le code et les données des processus en cours de fonctionnement dans le système ainsi que de ceux qui sont en attente d'exécution. Cette mémoire est, dans le cas idéal, cadencée à la vitesse du bus système.
- Mémoire d'entrée sortie. Elle contient les espaces de communication avec les périphériques. Elle est normalement cadencée à la vitesse du bus système et de sa capacité et de ses mécanismes d'attribution dépend le nombre de périphériques que le système pourra gérer sans conflit. Elle est donc segmentée en partie contenant pour chaque périphérique des registres de commandes et des buffers permettant de lire et d'écrire des données. On appelle ces zones des ports. Sa limitation à 64 ko est le principal défaut de l'architecture PC, d'autres systèmes mieux conçus la logent en mémoire centrale.
- Mémoire virtuelle. Appelée souvent zone d'échange, espace de pagination ou zone de swap, c'est une partie de disque contenant les parties de mémoire inutilisée à un instant donné par les processus actifs. Elle est cadencée à la vitesse du disque, c'est-à-dire à 10 ou 15 ms dans les meilleurs cas.
- Le DMA est un circuit chargé de soulager le processeur en réalisant des transferts depuis ou à destination des périphériques. Cela signifie que le processeur a juste à demander le transfert des données et peut se consacrer à autre chose pendant ce temps. Le débit et le nombre de canaux de ce circuit conditionnent la capacité du système à exploiter rapidement la mémoire virtuelle en réalisant des entrées-sorties rapides.

Deux choses définissent vraiment la qualité de la mémoire : ce que l'on appelle la vitesse, et la capacité.

La vitesse d'une mémoire n'est en fait que son débit. Elle est de l'ordre de 5 $10^{-9}$ secondes pour les circuits les plus rapides. Dans le cas où le système possède un processeur à $N$ Mhz, il est capable d'accepter les données en $\frac{1}{N}10^{-9}$ s, soit 0.00142 $10^{-9}$ dans le cas d'un processeur 700 Mhz. Ceci démontre l'intérêt des mémoires cache de toute sorte. Le débit d'une mémoire est donc donné en b/s et dépend de la qualité des circuits et de la fréquence du bus qui l'exploite. Dans la plupart des cas, les processeurs acceptent les octets par groupe et le débit doit donc être divisé d'autant.

La capacité est définie en kilo-octets car le processeur n'exploite les bits que par groupes de 8/16/32. Ce qui la caractérise est qu'elle n'est jamais suffisante. C'est un composant stratégique des ordinateurs et un des éléments dont le prix est sujet à d'importantes variations.

A ces types de mémoire, il faut ajouter les types d'adressage :

- Adressage physique : les adresses correspondent à un circuit électronique identifié. c'est l'adressage qu'utilise le système à bas niveau, ce qui se comprend dans la mesure où il adresse les buffers des cartes de la machine.
- Adressage virtuel : les adresses correspondent à des adresses physiques via une table de translation que maintient le système. C'est en général l'adressage qu'utilise les process utilisateurs ce qui leur permet d'être facilement déplaçables.

Les programmes ont eux deux modes de désignation de cases mémoires qui sont l'adressage absolu et l'adressage relatif. Ce dernier, qui spécifie les adresses sous forme d'un offset par rapport à la position courante est plutôt utilisé pour les sauts en mémoire alors que le premier est plutôt utilisé pour l'adressage des données.

## 4.2 Allocation de mémoires

Dans les systèmes mono-tâches, l'allocation de la mémoire est simplement faite en indiquant au programme qui se lance l'adresse minimum et l'adresse maximum qu'il peut utiliser. A lui d'en faire bon usage ! Ce mode d'allocation est aussi celui utilisé par les systèmes à multi-tâches coopératifs (MAC-OS) sous des formes légèrement raffinées.

Dans les systèmes multi-tâches multi-utilisateurs, l'allocation de la mémoire doit tenir compte de plusieurs éléments par ordre d'importance :

- permettre le cloisonnement des processus
- garantir sa disponibilité pour les process en cours et démarrant
- garantir un accès rapide aux informations

Il apparaît donc nécessaire que le système dispose d'une technique d'allocation permettant de :

- Savoir donner de la mémoire au process
- Savoir récupérer les zones qui ne sont plus utilisées
- Savoir détecter les accès indus

Ceci implique que le système doit constituer une table d'occupation de la mémoire et que la structure de cette table sera déterminante sur la capacité multi-tâche du système.

Il existe deux modes de construction de cette table d'occupation : la table de bits et la liste chaînée de fragments.

- La liste chaînée.
  Cela consiste à allouer à chaque process des fragments de mémoire en fonction de leur demande en constituant une liste chaînée au fur et à mesure. On a donc une liste de blocs alloués qu'il suffit de parcourir quand on cherche à libérer ou à allouer une zone.
  La liste ressemble à ceci :

```
struct {
Mem_liste *last,* next; /* Fragment suivant et précédent    */
ulong bot,size;         /* début et longueur du fragment     */
ushort proc;            /* Pid du process qui utilise 0=libre */
} Mem_liste;
```

  Pour ce mode de réservation, la libération est extrêmement simple : il suffit de trouver le fragment à libérer et le marquer libre en mettant l'attribut proc à zéro. Si le fragment libéré a pour suivant le dernier de la liste, on le supprime et corrige les valeurs bot et size du dernier. L'allocation est par contre plus problèmatique. Au début, on initialise la liste avec tout l'espace d'adressage du système, puis on construit la liste en ajoutant un élément à la fin sur lequel on a décalé bot et size. Lors d'allocation ultérieure, on cherche le fragment le plus proche de la taille dont on a besoin et s'il n'en existe pas, on l'ajoute à la fin.
  Nous constatons donc que ce système risque de conduire à une fragmentation importante de la mémoire ou de nombreux trous de petite taille vont fatalement apparaître. La vitesse d'allocation va, de plus, dépendre du nombre de fragments et donc l'utilisation de nombreux process fera baisser les performances du système.
  Cette méthode est par contre très économe (18 octets par fragment sur un système 32 bits) en mémoire si on utilise des fragments de taille raisonnable.
- Table de bits (bloc bitmap)

L'allocation par table de bits consiste à noter dans un tableau l'occupation par les process de zones de tailles fixes appelées pages.
Dans la pratique, il est nécessaire de noter pour chaque page le process propriétaire. La table de bits est donc un tableau d' unsigned short et à chaque couple d'octets correspond une zone dont on calcule l'emplacement en effectuant un simple décalage de bits. Pour faciliter ce calcul, les pages ont toujours une taille de $2^n$ octets. Les process peuvent facilement utiliser un mécanisme d'adressage virtuelle en associant à cette table leur propre valeur de translation.
L'allocation de mémoire à un process se fait par une simple recherche de la première page libre et la libération par mise à zéro des pages non utilisées. La défragmentation de la mémoire peut se faire par recopie des pages par le système durant les périodes où il n'a rien à exécuter. Le cloisonnement en mémoire des process est aussi extrêmement simple à réaliser, vu qu'il suffit lors des demandes d'accès de vérifier le propriétaire de la page.
Le seul défaut de cette méthode est de consommer une quantité importante de mémoire. Soit on choisit de petites pages et la table est de taille importante, soit on choisit des grandes pages et le gaspillage se fera dans les pages elles-mêmes. La taille de pages en vogue à notre époque est de 8ko, mais cela est plus dû à des considérations de matériel que d'optimisation logicielle. En effet, sur de nombreuses architectures, un circuit spécialisé du chipset prend en charge une grande partie de cette gestion et donc conditionne la taille des pages. De plus pour être efficace en associations avec de la mémoire virtuelle sur disque, ce système doit employer des pages de taille compatible avec les buffers des principaux contrôleurs disques. D'où ce choix de 8 ko qui satisfait un peu toutes ces contraintes.

Dans tous les cas et comme cela est dit plus haut, la mémoire manque toujours, d'où les mécanismes de mémoire virtuelle décrits ci-dessous.

## 4.3 Gestion de la mémoire virtuelle

Le principe de ce type de mémoire est d'utiliser le disque quand la mémoire vive électronique vient à manquer. C'est une méthode très répandue et aucun système multitâche n'y échappe.

Cette technique est loin d'être une panacée, du fait que les disques restent quand même extrêmement lents. Un accès disque prend un temps de l'ordre de $10ms = 10^{-2}s$ alors qu'un accès en mémoire prend environ $\frac{1}{100}10^{-9}s$ la pagination (ou swaping) est donc un goulot d'étranglement redoutable pour tout système qui y fait appel.

Cette utilisation du disque peut conduire à une situation de blocage apparent du système dans plusieurs cas :

- allocation sauvage de mémoire par un utilisateur avec une fonction de type calloc, c'est-à-dire, que le dit sauvage non seulement s'alloue de la mémoire, mais force en plus le système à aller mettre des zéros dans tous les blocks disque correspondants.
- Surconsommation générale de mémoire et d'E/S qui conduit le système à ne faire plus que des E/S disque.

De nombreux algorithmes essaient de pallier ces inconvénients. La difficulté qu'ils rencontrent tous est que le process choisissant les pages à mémoriser sur disque peut se retrouver lui-même sur disque et donc provoquer lui-même la catastrophe. Une solution peu consommatrice de CPU et fiable reste donc à inventer.

L'optimisation la plus souvent rencontrée est que l'on écrit les pages qui ne sont jamais modifiées qu'une fois sur le disque et non à chaque fois qu'elles ont besoin d'être effacées de la mémoire.

## 4.4 Memory mapping

Cette technique permet d'utiliser un fichier comme partie ajoutée à l'espace de pagination. Elle permet donc d'associer un pointeur dans un langage de haut niveau à un fichier sur un périphérique. La réalisation pratique de cela se fait en utilisant les capacités de mémoire virtuelle de la machine et en rajoutant donc des pages à la liste existante.

Cette fonctionnalité permet une manipulation aisée des fichiers. Bien evidemment, les pages disques ne seront lues et écrites que si nécessaire. 10 minutes de temps consacrées à la lecture du man de $mmap$ vous convaincrons de la simplicité de l'emploi de cette technique.

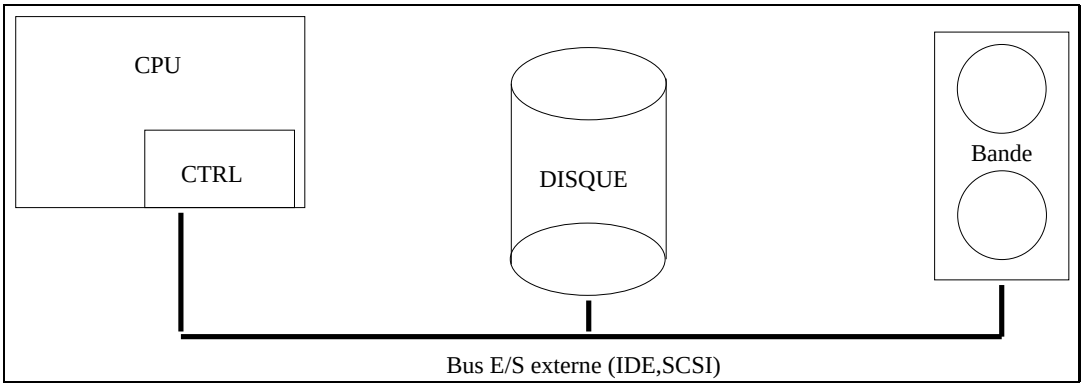


FIG. 11 – Système de stockage d'informations

# 5 Système de fichiers

## 5.1 Aspect hardware du stockage d'information.

L'informatique est la technique de manipulation automatique de l'information, il est donc naturel de mettre en place des mécanismes de stockage d'information, avant, pendant et après leur traitement.

### 5.1.1 Appareillage

Un périphérique de stockage de fichier est constitué d'au moins un média et d'un contrôleur.

Le système devra donc être capable d'attaquer le contrôleur, de transmettre à celui-ci les ordres de lecture et d'écriture et éventuellement d'agir directement sur le média, effacement, formatage bas niveau. Nous constatons également qu'un contrôleur peut gérer plusieurs périphériques de même type, le driver devra donc être capable de distinguer les différents média. Il devra aussi être capable de gérer les changements de media lors de l'utilisation de lecteur de disque amovible (lecteur de disquette).

Les périphériques actuels de stockage des données peuvent être répartis en deux groupes : ceux à accès séquentiel et ceux à accès direct.

- accès direct
  Il s'agit de tous ceux comportant un disque et une tête de lecture tangentielle. Il s'agit également des disques sous forme de mémoire flash. Ils sont en théorie isotrope du point de vue de la recherche d'information. C'est-à-dire que la tête de lecture peut atteindre n'importe quel point du disque dans un temps très inférieur à celui nécessaire à lire quantité d'information séparant le point courant du point visé.
- accès séquentiel
  Il s'agit principalement des lecteurs de bande : CCT, Streamer, Dat, Exabyte, DLT sont les principales utilisées dans les systèmes informatiques actuels. Ces périphériques nécessitent un parcours de toute la surface du média pour se rendre sur un point. Ils sont donc beaucoup moins performants en accès que les disques mais souvent plus rapides en débit.

Les capacités vont de 1 Mo à 72 Go pour les disques et de 2 à 80 Go pour les bandes. Plusieurs choses conditionnent l'utilisation que l'on a d'un média. Il s'agit de déterminer entre sa capacité, son prix, sa capacité à être réinscrit et l'aspect séquentiel/non séquentiel.

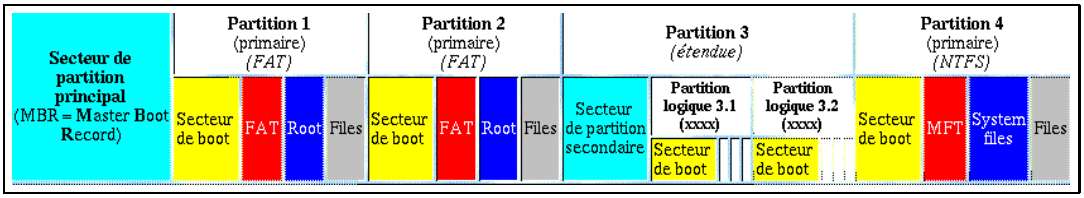


FIG. 12 – Structure d'un disque de PC.

Les OS sont donc, suivant leur utilisation réelle, appelés à gérer des situations très diverses. Ils doivent donc mettre en oeuvre des mécanismes de haut et de bas niveau pour chaque type de périphérique.

### 5.1.2 Partition disque et bande

Les medias fournissent de grands volumes de stockage. Les développeurs de système ont mis en place des sous-divisions appelées partition. Ceci permet des accès plus fins et ciblés que le simple repérage sur une énorme suite d'octets. Le partionnement permet aussi de cloisonner des informations de façon à garantir leur intégrité. Ceci permet aussi de garantir l'espace dévolu à une certaine classe d'information.

L'unité de stockage minimum sur un disque ou une bande est le block. C'est une suite d'octets de taille calculée pour que le contrôleur de périphérique puisse la lire ou l'écrire en une seule opération. Il correspond physiquement à une zone contigue du média dont on a fixé magnétiquement les limites. Ces limites sont fixées lors du formatage pour les disques et lors de l'écriture pour les bandes ou les CDrom. Les blocks sont en général séparés d'un petit espace qui permet de faciliter la lecture en insérant des marques de début et de fin de block.

Si le disque possède plusieurs plateaux, les blocks sont multiplexés sur les toutes les faces. On utilise alors autant de têtes de lecture que le disque comporte de joues pour coder un octet. Ces têtes sont liées mécaniquement entre elles et de fait se déplacent toutes en même temps. On peut donc lire plus vite les données du fait que l'angle de rotation des plateaux nécessaire au codage d'un octet est beaucoup plus petit.

Sur les disques, les blocks sont regroupés en pistes, suite de blocks contigus, et les groupes de piste sont regroupés en partitions. Pour réduire la taille des nombres nécessaires à l'adressage, on utilise aussi la notion de secteur (Cluster) qui regroupe un nombre donné de blocs et permet donc un adressage par indirection.

Sur les bandes, les informations sont regroupées en blocks puis en fichiers. Les fichiers et les blocks se distinguent par des marques magnétiques particulières qui sont, soit enregistrées avec les données, soit sur une piste spéciale appelée piste de synchro. Cette piste a en général une densité d'écriture inférieure aux autres et permet un parcours à grande vitesse pour sauter les blocks et les fichiers que l'on n'utilise pas.

Les blocks sont, comme sur les disques, marqués magnétiquement dans un espace interblock. Les fins de fichiers sont marquées magnétiquement (EOF) avec un block particulier et une marque redondante sur la piste de synchro. La fin de la zone utilisée sur la bande est signalée par une marque EOM et la fin physique par une marque EOT. Les bandes ne se formatent en général pas mais il est quelquefois nécessaire de les effacer complètement pour, par exemple, changer leur densité d'enregistrement.

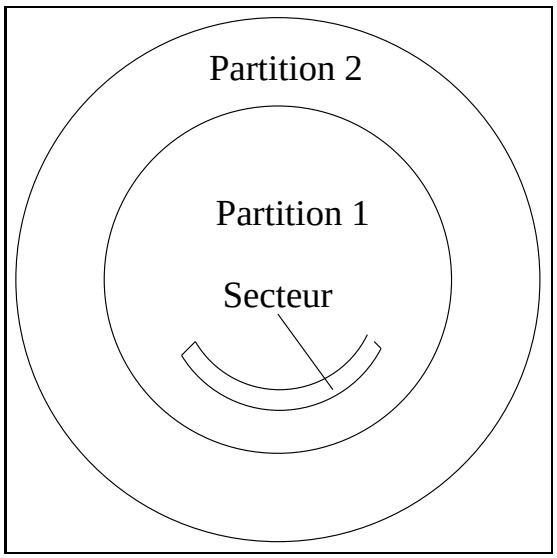


FIG. 13 – Disque,partition,secteur.

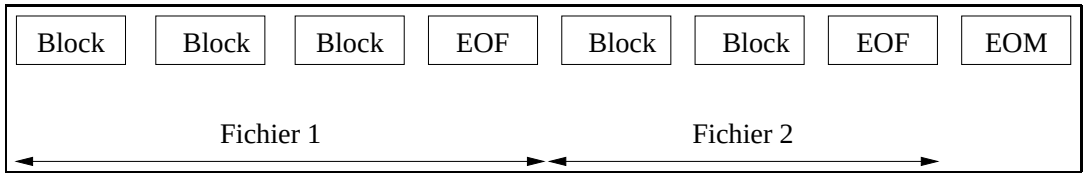


FIG. 14 – Répartions des informations sur bande.

Cette dernière est exprimée en bit per inch (bpi).

Dans tous les cas, plus les blocs sont petits, plus on perd d'espace sur le média du fait de plus nombreux espaces interblocks. La taille des blocks a un effet également sur le débit mais plus complexe. En effet, plus les blocks sont petits, plus il faut parcourir d'espace pour lire, mais s'ils sont trop grands, le contrôleur n'arrive pas à les accepter[5] en une fois et donc une relecture peut devenir nécessaire. La taille des blocks a aussi un impact important sur la fiabilité du média car plus ils sont grands, plus une destruction d'une zone peut avoir des conséquences importantes en terme de perte de données. Actuellement tout le monde s'accorde sur des blocks de 512 octets sur disque et de 1024 ou 2048 sur bande.

Le partitionnement s'avère utile au moins dans deux cas :

- Protection des données en cas de destruction (partielle) d'une partie de la surface magnétique. Dans ce cas, on ne perdra qu'une partie des fichiers.
- Facilité de manipulation et rapidité.

Les inconvénients du partionnement sont dus à l'espace qui lui est nécessaire et donc à la perte de capacité d'enregistrement qu'il induit. Cet inconvénient est de moins en moins problématique de nos jours. Dans un système, on ne doit jamais perdre de vue que les tailles de partition de l'ensemble du système doivent être suffisantes pour accueillir les données qui leur sont destinées, mais également pouvoir être sauvegardées sur un seul support.

[5] Du fait d'une largeur de bus ou d'une taille de buffer limitée.

| *Adresse (hexa)* | *Contenu* | *Type* |
|---|---|---|
| **000** | Programme de partition | Code (446 octets au maximum) |
| **1BE** | **1**ère entrée dans la table de partition | 16 octets |
| **1CE** | **2**ème entrée dans la table de partition | 16 octets |
| **1DE** | **3**ème entrée dans la table de partition | 16 octets |
| **1EE** | **4**ème entrée dans la table de partition | 16 octets |
| **1FE** | **AA55** (code d'identification) | 2 octets |

FIG. 15 – Structure du MBR PC.

Les informations concernant la taille des blocks sont définies lors du formatage bas niveau (dit physique) des disques. Les informations de géométrie du disque sont stockées sur la première piste (dite 0) magnétique qui est inaccessible en mode normal de fonctionnement du disque. Sur la deuxième (dite 1) piste magnétique on va stocker la géométrie des partitions physiques du disque de façon que le contrôleur ou le système arrive à les distinguer.

Cette information est stockée dans le block appelé MBR sur PC dont la figure 15 donne un aperçu.

## 5.2 Aspect logique des systèmes de fichiers.

Les données informatiques sont regroupées en général en fichiers afin d'en faciliter la gestion par l'utilisateur. Cette structure logique permet à l'utilisateur de les désigner facilement sans avoir à causer en blocks avec le contrôleur.

Les fichiers sont désignés par un nom et tout le problème est la liaison entre ceux-ci et le matériel.

### 5.2.1 Notion de volume

Un volume est un indentificateur que l'on associe à un groupe de fichiers. Historiquement, cela correspond à un périphérique (disque ou bande). Cela a deux avantages :

– On désigne le média dans les programmes.
– On indique à l'OS d'utiliser le bon sous programme du noyau pour gérer l'appareil.

Cette notion est toujours présente dans les systèmes mais parfois masquée à l'utilisateur par un autre système logique :

| | |
|---|---|
| MSDOS-NT | A :,C :,D :,C : jusqu'au Z : |
| VMS | $3$DRA0 c'est-à-dire $contrôleur$disquePartition |
| Mac | Nom quelconque sans . |

C'est une notion intéressante dans la mesure où elle introduit une séparation entre les systèmes de fichiers qui facilite la prise en main du système par des utilisateurs non avertis. Elle pose par contre le problème de l'allocation des noms qui peut conduire à planter des applications qui ne trouvent plus leurs fichiers du fait de l'ajout d'un nouveau volume (Application sur CDRom par exemple).

Il ne faut pas confondre la notion de volume et le label. Le label est une inscription logique que l'on met sur le média qui permet de les retrouver automatiquement ou de vérifier leur présence dans un lecteur. Ce label est parfois utilisé comme nom de volume (MAC).

### 5.2.2 Notion de catalogue

Face à l'augmentation du nombre de fichiers dû à la croissance de la taille des médias, il a fallu trouver une méthode pour les regrouper par centres d'intérêt de l'utilisateur de façon que celui-ci ne perde pas trop de temps à rechercher ses fichiers dans une liste trop longue. Ce regroupement est appelé catalogue ou directory.

Au départ, chaque catalogue était indépendant mais l'utilité d'une structure arborescente est rapidement apparue :

- Elle facilite l'exploration et la recherche de fichiers.
- Elle permet des sous-regroupements pratiques à utiliser (ex : distribution de logiciels).

Elle nécessite par contre de savoir désigner un père pour chaque sous directory. Certains systèmes ont choisi une convention **..** pour le père, mais ce n'est pas toujours le cas. VMS ne permet que l'utilisation des noms de directory qu'en utilisant toute leur lignée.

Les catalogues peuvent avoir une réalité physique sur le disque, mais ce n'est pas toujours le cas. Dans ce dernier cas, l'aborescence n'est qu'apparente et est simplement une convention qu'adoptent les appels systèmes pour désigner les fichiers.

### 5.2.3 Fichiers, API et masquages

Si le programmeur ou l'utilisateur devait désigner les blocks de disque qui contiennent ses données, sa vie serait particulièrement dure. L'OS met à sa disposition une batterie d'outils qui masque la structure physique de l'organisation des fichiers. Les contraintes de tels mécanismes sont :

- Pouvoir créer des fichiers
- Pouvoir allouer de la place à ces fichiers
- Garantir le respect de chaque fichier lors de l'écriture sur le disque.
- Pouvoir lire et écrire chaque fichier par morceaux[6]
- Pouvoir supprimer chaque fichier.

Du fait de ces contraintes, les médias à accès direct ont toujours une table d'allocation des fichiers dont la gestion est réalisée par le système d'exploitation. Pour que cette table puisse remplir son rôle, elle doit comporter un mécanisme de mémorisation de chaque block occupé par chaque fichier.

Il y a deux techniques pour cela :

- Le chaînage. On marque le block suivant en fin de blocks
- La table de secteurs. On garde une trace des blocks alloués à chaque fichier

La première est peu coûteuse en place, mais la destruction d'un block conduit à la perte de la fin du fichier. Elle permet par contre de rajouter facilement des données au milieu d'un fichier en modifiant les blocks comme on modifie une liste chaînée ordinaire. Cette méthode ne conduit pas à des pertes de données à condition d'avoir une unité d'enregistrement de données dans le fichier compatible avec la taille des blocks.

La table de secteurs (groupe de blocks) occupe plus de place mais offre une sécurité importante sauf en cas destruction totale. Elle ne permet par contre que l'ajout d'information à la fin du fichier. Avec des secteurs de 2048 octets et une table de longueur 128 ko, on ne peut adresser que 131 Mega.

[6] sinon on ne pourrait pas travailler avec des fichiers de taille supérieure à la mémoire vive de la machine

Dans la pratique, quelques systèmes utilisent ces deux méthodes pour sécuriser leurs écritures ou les accélérer. Dans tous les cas, les systèmes font toujours plusieurs copies des tables de secteurs pour éviter les pertes de données. Des utilitaires permettent de rechercher les blocks perdus et les chaînes sans débuts de façon à réparer les fichiers endommagés.

Une API de gestion de fichiers doit masquer ces problèmes en fournissant des fonctions :

- création de fichier à partir du nom,
- ouverture du fichier, c'est-à-dire initialisation d'un descripteur de fichier simple à manipuler pour le programmeur,
- une fonction de manipulation des attributs. Le rôle de la fonction de manipulation des attributs est de permettre d'ajouter des informations (DATE /PROPRIETAIRE /TYPE /DROIT /VISIBILITE) sur le disque en plus du contenu du fichier,
- d'écriture,
- de lecture,
- de suppression,
- de fermeture, c'est-à-dire de mise du fichier dans un état standard dans lequel toutes les demandes d'écriture ont été réalisées. Cette fonction réalise de plus la libération des buffers d'entrée-sortie.

De la facilité d'utilisation et de l'optimisation de cette API repose en grande partie le succès et la pérénité d'un système.

### 5.2.4 Protection, concurrence et verrouillage

Comme nous l'avons vu plus haut, un système doit assurer le cloisonnement des fichiers entre eux. Les systèmes multi-utilisateurs et multitâches doivent de plus gérer la concurrence d'accès aux fichiers par les différents process et la confidentialité des informations des différents utilisateurs.

- Pour la protection du point de vue utilisateurs les mécanismes de protection sont basé sur des reconnaissance de droit propre à chaque processus. Dans les systèmes multiutilsateur les processus fonctionne au nom d'un utilisateur, quand il créés un fichier, celui-ci hérite de leur nom d'utilisateur et en général de droit par défault. On dit que l'on donne un propriétaire au fichier. Les droits d'accès des autres utilisateurs du système sont défini par un système d'héritage par apport à des droits minimaux fixé par le système combinné à des régles d'héritage des droits du répertoire dans lequel le fichier à été créer. On distingue le plus souvent les droits de : lecture, écriture, ajout, suppression, exécution et destruction. Suivant le type de fichier (répertoire/données/programme) ces droits ont bien évidemment une traduction différente. Ces droit peuvent être fixé via un tableau (propriétaire/groupe/monde) ou via des ACL (liste de controle d'accès). Dans le premier cas la vérification se fait par apport à un propriétaire et à un groupe de référence, dans le deumième cas par apport à couple (OBJET/ACTION) qui peuvent être mulitiple.
- Pour les accès concurrents, on introduit deux types de verrouillage, un sur l'allocation des blocks disques et descripteurs de fichiers, un sur les fichiers eux-mêmes.
  Le premier sert à éviter que le système de fichiers devienne incohérent par attribution d'une même zone physique à plusieurs fichiers logiques. Il repose le plus souvent sur une bonne gestion des interruptions par le noyau. Il est particulièrement critique à mettre en place sur les systèmes SMP.

Le deuxième est destiné à garantir au processus utilisateur l'exclusivité d'écriture ou de lecture sur le fichier logique. Ce système est soit géré en mode noyau, soit en mode utilisateur. Il est donc soit obligatoire (NT/VMS) soit laissé à la discrétion du programmeur. Le verrouillage systématique est très efficace mais devient rapidement la cause principale de cauchemar de l'administrateur du système. En effet, tout process coincé après l'ouverture du fichier bloque l'accès au fichier en question.

## 5.3 Description de systèmes existants

### 5.3.1 MSDOS-FAT

Ce système repose sur deux mécanismes, la FAT (File Allocation Table) et les directories. La FAT est une liste chaînée de groupes de blocks de disque (cluster), elle recense donc quels secteurs sont libres et quels secteurs sont alloués sur le disque. Si ce chaînage disparait, le disque est perdu. Les disques en possèdent donc une copie accessible suivant des procédures spéciales.

Les directories associent le nom du fichier, leur date et une référence sur les premiers blocks les concernant dans la FAT.

L'avantage de ce système est que toutes les informations sont sur le disque, ce qui permet des arrêts à chaud sans risque. Le défaut est par contre la lenteur de l'ensemble qui oblige la mécanique du disque à de fréquents allés et retours. Il a aussi l'avantage d'être le système de fichiers le plus répandu et donc d'être reconnu par tous les systèmes d'exploitation.

Une extension dite FAT32 permet d'adresser des disques plus grands grâce à une structure de FAT où les adresses de blocks sont écrites sur quatre octets au lieu de deux.

### 5.3.2 UNIX

Le système de fichiers UNIX est défini par le cahier des charges du système, mais variable dans les implémentations. Il n'est en général pas journalisé et les tables de fichiers sont manipulé en mémoire durant le fonctionnement du système.

Les règles qui les régissent sont les suivantes :

- Chaque disque peut être découpé en partitions.
- Chaque partition contient des fichiers qui sont des suites de blocks disques sans organisation particulière.
- Chaque fichier possède un inode, qui est un numéro d'identification interne au système.

La structure d'une partition disque est donc découpée en groupes de blocks dont la structure est la suivante :

| Block | contenu |
|---|---|
| 0 | Mbr |
| 1 | Superblock |
| 2 | Block bitmap |
| 3 | Inodes 1 |
| | ... |
| N+1 | Inodes N |
| N+2 | block de donnée 1 |
| | etc |

Mbr : il contient la table de partitions du disque
Superblock : il contient les adresses des tables des espaces disponibles
Block bitmap : il contient une table des blocks de données libres dans le groupe
Iliste : la liste des inodes
Fichiers : cette zone contient les données des fichiers proprement dits.

un inode a la structure suivante :

```
struct  stat {
        dev_t   st_dev;                 /* designe le device physique */
        ino_t   st_ino;                 /* le numero du noeud         */
        mode_t  st_mode;                /* le mode d'acces au fichier  */
        short   st_nlink;               /* le nombre de references      */
        uid_t   st_uid;                 /* le proprietaire             */
        gid_t   st_gid;                 /* le groupe */
        dev_t   st_rdev;
        off_t   st_size;                /* la taille du fichier en octets */
        time_t  st_atime;               /* la date du dernier acces */
        int     st_spare1;
        time_t  st_mtime;               /* la date de modif */
        int     st_spare2;
        time_t  st_ctime;                /* la date de creation */
        int     st_spare3;
        long    st_blksize;              /* la taille des blocks */
        long    st_blocks;               /* le premier block */
        long    st_spare4[2];            /* les blocks d'indirection double
                                         /* ou triple */
};
```

Nous constatons donc que l'organisation physique du disque n'impose aucune organisation a priori des fichiers. La structure arborescente est donc notée dans des fichiers ordinaires qui sont chaînés entre eux par les informations qu'ils contiennent. Un directory connait son père (éventuellement lui même) comme **..** et lui même comme **.** , les inodes des fichiers qu'il contient et ses fils. La structure de directory est décrite dans **dirent.h**.

L'organisation n'est donc qu'apparente à l'utilisateur. Cette méthode a l'avantage de permettre des suppressions très rapides de fichiers et de directories. Elle permet aussi de référencer un fichier à plusieurs endroits en inscrivant son inode dans plusieurs directories.

Le système crée les fichiers, par recherche du premier inode libre. La suppression d'un fichier intervient en mettant la valeur **st_nlink** à zéro. Quand on référence un fichier d'un répertoire, cette

valeur est incrémentée et quand un process ouvre le fichier, elle l'est également. Elle est décrémentée quand un process ferme le fichier ou quand l'utilisateur efface le fichier d'un directory. Quand la référence tombe à zéro, le fichier est simplement oublié et l'inode sera réutilisé lors de la prochaine création.

Les opérations de contrôles de droits d'accès sont réalisées par masquage binaire de l'UID et du GID de l'utilisateur par les droits d'accès stockés dans l'inode comparé au même masquage réalisé sur l'UID et le GID du fichier. C'est un mécanisme simpliste mais extrêmement rapide.

Le superblock et la i-liste sont stockés en mémoire vive pendant le fonctionnement du système. C'est très rapide, mais une écriture périodique **sync** et une procédure particulière d'arrêt de la machine **halt** sont nécessaires.

L'allocation d'espace à un fichier se fait d'abord dans le groupe de blocks courants **st_blocks**, puis par indirection double (c'est-à-dire que l'on donne l'adresse d'un block qui contient l'adresse d'autres blocks) puis par indirection triple si cela s'avère nécessaire. Nous pouvons ainsi adresser un espace considérable à un fichier ($2^3 2^3$ octets).

Du point de vue de l'utilisateur, deux interfaces apparaissent, les fichiers contenus dans le répertoire **/dev** donnant accès aux périphériques et le système de fichiers arborescent.

– Les devices.
  Il y en a un par disque ou lecteur de bande et 1 par partition de disque. Nous disposons pour chaque périphérique d'un device text (sans formatage) et d'un device block (structuré). On peut réaliser directement les opérations d'entrées-sorties sur ces devices mais, dans ce cas, ils sont vus comme des suites d'octets ou de blocks sans organisation particulière.
– Le système de fichiers
  Celui-ci possède une racine qui est le répertoire / qui est au moment du boot de la machine le seul apparent. Il est ensuite possible de raccorder les autres disques ou partitions sur des directories de ce file system. Cela s'appelle monter un système de fichiers. L'administrateur système a donc tout le loisir de raccorder les disques où bon lui semble.

Le système de fichiers UNIX permet donc un grand nombre de manipulations en fournissant des performances d'accès très honorables. Il est à noter qu'il ne fournit aucun système de verrouillage de fichiers, cela étant bien evidemment pris en charge par le noyau.

### 5.3.3 WinNT

WinNT tolère trois types de système de fichiers : FAT, NTFS et HPFS.

Le premier est d'origine DOS, le deuxième est le système de fichier spécifique à NT et HPFS est une survivance d'OS2.

Le démarrage de NT ne peut-être que dans un de ces trois types de partition du fait de l'utilisation de l'interruption 13H du DOS pour accéder au bootblock.

Merci à Simon ANDRIEU (F1) pour la frappe de cette partie !

NTFS doit permettre de gérer des fichiers pour différents types de standard. Ces standards sont au nombre de trois : FAT HPFS(0S/2)é POSIX. NTFS doit donc reprendre les caractéristiques des trois.

Au dire du constructeur les exigences de conception de NTFS sont :

– Récupération : NTFS permet une récupération des données endommagées. Ceci est fait en

implémentant un modèle transactionnel (style SGBD) c'est-à-dire un système atomique permettant le groupage d'opérateurs et le roll-back de toutes les actions si une partie de l'atome ne peut être réalisé.
- Sécurité : Chaque fichier reçoit des ACL qui permettent de vérifier les droits d'accès pour chaque opération possible pour tous les utilisateurs. Seul l'administrateur peut fixer les droits pour tous.
- redondance des données et tolérance de panne : Grâce au modèle de pilote en couche, NT peut insérer une couche de type RAID 1 à RAID 5 permettant une récupération complète des données.
- grand disque et fichier : Un cluster a une taille variant de 512 octets à 64 Koctets. Ils sont numérotés sur $2^{64}$ bits. ? introduit la "sur" unité d'allocation de fichier. les fichiers peuvent donc avoir une longueur de $2^{64}$ octets et les disques gérant $2^{64} * 2^{16}$ octets.

D'autre part, NTFS implémente quelques fonctions intéressantes :

- Streams de données : Un fichier NTFS peut avoir plusieurs enregistrements distincts appelés "Streams". Les streams ont un numéro propre venant après le nom de fichier et le séparateur : cela permet, par exemple, de gérer facilement les fichiers associés à des ressources dans le style MAC/OS . Il s'agit du grand retour du fichier séquentiel à enregistrement variable très aimé sur VMS.
- Nom de fichier constitué de caractères 16 bits UNICODE.
- Utilitaire d'indexation générale sur tous les attributs de fichier.
- Recoupage dynamique des clusters défectueux.
- Support des liens matériels POSIX.

Au niveau structure des données sur disque, NTFS reprend les idées de MSDOS pour l'organisation en cluster et pas mal de notions UNIX pour le reste. La structure s'appuie sur la notion de MFT (Master File Table) qui contiendra une *ligne* pour chaque fichier du volume ou une *ligne* par fragment de fichier quand ceux-ci sont trop gros.

Les autres attributs du volume sont stockés dans des fichiers de ? commençant par $ ($MFT, $BOOT, $LOGFILE) dont ? ($MFT) sont ? sur le volume lui-même. Les secteurs défectueux sont mappés dans $VOLUME.

Comme sous UNIX, les répertoires sont des fichiers ordinaires et recouvrent un numéro de fichier doublé d'un numéro de séquence qui est l'index du fichier dans la MFT.

Un numéro de fichier est de la forme :

| Numéro de fichier (48 bits) | No de séquence (16 bits) |
|---|---|

soit $2^{64}$ numéro par volume.

un enregistrement MFT est du type :

| Info standart | Filename | Sécu | Data | HPFS |
|---|---|---|---|---|

ou Data contient les données d'un fichier ou les listes de cluster contenant les données du fichier. Cela est également le cas pour les répertoires et si on manque de place on alloue un deuxième enregistrement dans la MFT. Tout cela fait terriblement référence à des notions UNIX.

L'enregistrement de MFT étant limité à 1024 octets, le descripteur de sécurité peut être un trop petit il est donc nécessaire de les stocker dans un fichier central. Windows 2000 aporte encore d'autres améliorations de NTFS ce qui introduira probablement quelques problème de compatibilité.

## 5.4 Sécurité des systèmes de fichiers.

A ne pas confondre avec la sécurité des fichiers dans les systèmes de fichiers.

### 5.4.1 Sauvegarde et réparation

Pour être sûr, un système de fichiers doit être sauvegardable, c'est-à-dire que les systèmes d'exploitation fournissent un ensemble d'outils permettant de recopier les fichiers sur un autre support.

Cela pose divers problèmes, concernant notamment les fichiers ouverts ou, sur les systèmes multi-utilisateurs, les fichiers qui disparaissent ou s'étendent durant l'opération de sauvegarde. Un autre problème est la charge CPU induite par ce type d'opération. On doit donc pouvoir procéder de façon incrémentale, c'est-à-dire ne sauvegarder que ce qui a été modifié depuis la dernière sauvegarde.

Un système correct doit donc fournir un outil de backup tenant compte de ce cahier des charges, tout en permettant une restauration à chaud des fichiers. Dans la pratique, UNIX et VMS fournissent tout ce qu'il faut pour cela (**dump et restore**), mais vous laissent assurer la cohérence entre ce que vous récupérez sur les bandes et l'état actuel de votre machine.

Les systèmes WinNT/2000/XP ne fournissent par contre pas de système satisfaisant de sauvegarde des fichiers dans la mesure où leur système de verrouillage ne permet pas d'assurer à coup sûr l'accès à un fichier. Ils ne fournissent pas non plus d'outil satisfaisant permettant de réparer un système endommagé, il faut donc être très prudent.

Certains systèmes comme MSDOS permettent de récupérer des fichiers endommagés ou effacés par accident en s'appuyant sur les chaînage restant présents sur le disque. Cela se fait avec l'ordre **recover**, **chkdsk**, **scandisk** ou **unformat** suivant le type d'incident intervenu.

En cas d'arrêt à chaud, les systèmes de type UNIX fournissent un outil de réparation **fsck** qui contrôle l'état et la cohérence des fichiers et permet de supprimer les fichiers pour lesquels la cohérence ne peut être assurée. Afin d'augmenter les chances de réparation, les informations contenues dans le super block et dans la table des inodes sont recopiées plusieurs fois sur le disque, **fsck** peut donc s'appuyer sur ces copies pour assurer la cohérence des fichiers.

### 5.4.2 Redondance

Le problème de la résolution des pannes à chaud sur les systèmes n'échappe pas à la règle absolue de l'informatique qu'est la redondance. Il existe deux techniques dans ce domaine :

– la redondance totale
  On recopie toutes les informations, on fait alors ce que l'on appelle du mirroring. On utilise systématiquement deux disques, et en cas de défaillance partielle ou totale de l'un des disques, on utilise l'autre. Ce système suppose que l'on est capable de detecter une panne. Il présente aussi l'avantage de permettre de répartir les ordres de lecture sur les deux disques et donc d'accélérer les entrées de données.

– la redondance partielle
  C’est une chose qui est mise en oeuvre depuis bien longtemps, c’est-à-dire que l’on duplique les informations vitales (superblock, FAT) mais il est maintenant mis en oeuvre grâce à des systèmes hardwares complets qui masquent l’accès au système, l’accès physique au disque. De tels systèmes sont appelés RAID. Le niveau RAID 3 dédie un disque à stocker des sommes de contrôle de trois autres disques. Le niveau RAID 5 se pratique avec 5 disques plus une roue de secours. Pour ce niveau, chaque disque stocke des sommes de contrôles des autres disques et permet ainsi une reconstruction sur le sixième disque en cas de crash complet de l’un des disques.

En conclusion sur les systèmes de fichiers, je dirais que la seule chose qui compte vraiment est d’avoir des copies de tout ce que contiennent vos disques, surtout si vous êtes admistrateur système. Pour au moins deux raisons, conserver votre travail et par respect pour les milliers d’heures passées par vos utilisateurs à constituer ces fichiers.