Ecole Nationale des Sciences de l'Informatique

Rapport de Stage

Des Systèmes d'Exploitation

Une approche pratique

Par
CHIHI Imed
ELACHTAR Eslah
ROMDHANI Imed

Eté 1996

Remerciements

Nous tenons à remercier le personnel de la bibliothèque: Monsieur Jamel et Mademoiselle Saloua pour la disponibilité et la patience dont ils ont fait preuve pour nous fournir les documents et les références nécessaires à l'élaboration de ce travail.

CHIHI Imed
ELASHTAR Eslah
ROMDHANI Imed

Je voudrais remercier tous ceux qui m'ont enseigné l'informatique auparavant: Mr Lassaad ASMI, Mr Adel NEHDI, Mr Taoufik YAHMADI et Mr Tarek TOUJANI. Je leur dois beaucoup.

Imed CHIHI

ملخص

أو آس–بايس ليس تطبيقا متخصصا رغم أنه يبدو نظام معالجة قواعد بيانات، إذ أنه جعل لكي يبيّن إستعمالات تقنيات أثبتت فعاليّتها في هندسة البرامج الحديثة مثل التخاطب الميسّر مع المستعمل و التصرّف في الذّاكرة الإنتقالية و التصرّف في نظام الملفّات على طريقة "يونِكس".

المفاتيح: نظم التّشغيل، واسطة رسوميّة، نظام ملفّات

Abstract

OS-Base is not designed for a special aim, though it could look like a Data Base Management System, it was made to show the efficiency of some modern operating systems' fundamentals: easy-to-use Computer-Human Interface, virtual memory management and UNIX-like file system support.

Keywords: Operating system, graphical interface, file system, memory management.

Résumé

OS-Base n'est pas une application spécialisée, bien qu'il ressemble de loin à un Système de Gestion de Base de Données. Il fût conçu pour mettre en œuvre des notions qui sont à l'origine du succès des systèmes d'exploitation modernes: interface Homme-Machine intuitive, système de gestion de mémoire virtuelle et support d'un système de fichier semblable à celui de UNIX.

Mots clés: système d'exploitation, interface graphique, système de gestion de fichiers, gestion de le mémoire.

Avant propos

Dans le cadre de leur formation, les élèves ingénieurs de l'Ecole Nationale des Sciences de l'Informatique, ayant réussi leur première année d'études, sont appelés à réaliser un projet d'été qui est présumé être en rapport étroit avec les connaissances qu'ils ont acquises dans la classe II 1. Ce travail, en principe logiciel, doit être réalisé en langage ADA.

Ce travail d'été a pour but d'initier l'élève ingénieur au travail personnel, à la maîtrise du temps, à la recherche bibliographique...Notons là que bien que le Plan de Développement cite que ce stage vise l'ouverture sur des outils et des langages nouveaux, l'administration a jugé bon que ça soit élaboré en ADA, un langage enseigné au cours du deuxième semestre.

Le langage ADA a été conçu pour le développement sur les gros systèmes, sa conception a été faite avec un grand souci de portabilité et d'indépendance du matériel. Ainsi, l'accès aux ressources matérielles, très sollicitées par un système d'exploitation, fût d'une telle difficulté.

Parmi les modules de la classe II 1 nous avons opté pour « Systèmes d'exploitations et utilitaires de bases », des travaux sur ce thème nécessitent à la fois de bonnes connaissances au niveau du matériel ainsi qu'une solide maîtrise d'algorithmique.

Ce travail que nous avons dénommé OS-Base, est à première vue un système de gestion de bases de données avec une belle interface, mais ceci cache tout un mécanisme de *swapping* sur disque et une gestion de fichier avancée qui simule le fonctionnement du système de gestion de fichier de UNIX.

TABLE DE MATIERES

0.
Introduction

A propos de ce chapitre

C'est une présentation des systèmes d'exploitations, de leur principes, de leur fonctionnements et des plates-formes matérielles dont ils dépendent.

"Dans certaines applications, nous constatons que la technologie est la partie facile. L'aspect le plus difficile est le côté social. Il demande beaucoup de réflexion et donc de temps."

Donald Norman(*)

Au début des années cinquante, les ordinateurs étaient vus comme étant des calculateurs, ils le sont encore, mais ce n'est plus ce qu'on en pense. En effet, à voir les capacités des machines actuelles, on se croit devant une merveilleuse création, pourtant il ne s'agit que du même calculateur des années cinquante dans lequel on a substitué les tubes à vides par des puces de Silicium rapides.

Ce qui nous a donné cette impression est une interface logique tellement ingénieuse qu'elle a pu transformer un tas de circuits en un serviteur intelligent et disponible: seul un système d'exploitation peut faire autant.

Le système d'exploitation est devenu l'élément déterminant d'un environnement informatique et il est aujourd'hui plus courant de dire que l'on travaille sous MS/DOS ou sous UNIX plutôt que sous telle ou telle machine particulière.

Un système d'exploitation est censé:

- Savoir gérer les ressources mises à sa disposition d'une manière intelligente, simple et optimale. Sur ce thème, un grand nombre de techniques logicielles (algorithmes) et matérielles fût mis au point.
- Mettre ces ressources à la disposition de l'utilisateur à travers une interface simple et ergonomique.
- Supporter la diversité des équipements matériels des différents constructeurs et offrir une interface d'accès permettant d'en faire abstraction.
- Protéger les ressources de chaque processus dans le cas d'un fonctionnement multitâche.
- Tolérer les erreurs et les défaillances matérielles et humaines.

(*) ***Donald Norman*** est psychologue de formation, créateur du département des sciences cognitives à l'Université de San Diego (Californie) Il est actuellement vice-président "Technologies avancées" chez Apple.

- Protéger les données/programmes d'un usager contre les accès indiscrets dans une architecture multi-utilisateur.

- Etre ouvert sur les autres systèmes et offrir des possibilités des interfaces de dialogue avec des plates-formes non homogènes avec la sienne.

I.
Interfaces Homme-Machine

A propos de ce chapitre

C'est une présentation de l'importance de l'interface Homme-Machine, de son aspect visuel, de son noyau et des travaux effectués sur chaque thème. On présentera des exemples des interfaces les plus courantes au fur et à mesure.

« ..je me suis battu pour qu'Apple comprenne que les ordinateurs devaient être graphiques de la tête au pied. »

Jef Raskin[1]

1. Introduction

Sur les machines anciennes, et dès l'apparition des systèmes d'exploitation en tant que programmes de gestion des ressources, la communication avec l'utilisateur, ou plutôt l'opérateur à l'époque, se faisait par des commandes gravées sur carte perforée ou sur bandes magnétiques ou même par des interventions au niveau de l'électronique.. bref, des outils qui manquaient de souplesse et d'esthétique. Avec l'évolution des technologies sont apparus les terminaux: un moniteur et un clavier avec une interface de connexion, sur ces terminaux les utilisateurs consultaient des banques de données, l'information défile sur l'écran ou s'imprime sur papier sans que l'interrogateur n'y puisse changer quoi que ce soit. Juste après apparurent, les systèmes interactifs: une machine centrale répond aux interrogations et interroge, ce fût le cas des réseaux UNIX par exemple. Même à ce stade, seuls les experts pouvaient tirer profit de telles machines, les interfaces utilisateur véritablement simples ne virent le jour qu'avec les systèmes graphiques avancés: la norme X-Window du côté du monde UNIX et Mac OS et plus tard Windows du côté de la micro personnelle. C'était une sorte d'évolution naturelle due à l'explosion des capacités multimédias des ordinateurs.

Comme les propos de *Jef Raskin* le confirme, une interface graphique est incomparablement plus simple à commander qu'une banale ligne de commande sur un fond noir. Ceci revient à une conception très soignée qui en fait un véritable «interlocuteur» intelligent.

[1] ***Jef Raskin*** est concepteur d'interfaces chez Apple. Il a piloté le fameux projet Macintosh

2. Ce que doit être une interface graphique

2.1. Ergonomique

Une interface graphique qui se respecte renferme beaucoup de génie, en effet il ne s'agit pas d'un tas de routines graphiques, c'est plutôt une multitude d'objets physiquement très différents qui fonctionnent en harmonie, et surtout un comportement qui prend en compte des caractères de la nature humaine tels l'intuition, l'habitude, l'oubli..

A propos du support des imperfections humaines nous aimons citer un résultat d'études menées par un laboratoire de conception de HCI *(Human-Computer Interface)*, en effet ces études affirment que dans une boite de dialogue, le nombre des objets affichés ne dois pas dépasser 7 car la mémoire visuelle humaine est incapable de mémoriser des objets plus nombreux. Pour rester simples et intuitives les interfaces graphiques actuelles essayent tout simplement de simuler le plus parfaitement possible un bureau classique avec des dossiers rangés, ouverts ou fermés.

Notons là qu'il ne faut pas confondre ergonomie et esthétique: ce qui est ergonomique doit se faire sans peine, ce qui est esthétique se fait avec admiration.. un exemple: Supposez que vous désiriez modifier un paramètre de l'affichage, certaines interfaces vous proposeront de voir dans le menu système/affichage/préférences ou plus simplement de cliquer sur le bureau pour faire dérouler les options qui s'y rapportent: ça c'est de ergonomie. Une fois la boîte de dialogue affichée, on peut vous proposer le logo multicolore du constructeur de votre carte graphique par exemple: ça c'est de esthétique. Il y a aussi autre chose: imaginez que vous désiriez détruire un document, sur Mac OS vous aurez à mettre son icône dans la poubelle (*trash*) qui se gonflera pour vous montrer qu'elle n'est plus vide, sur OS/2 vous aurez à mettre cette icône sur le broyeur (*shredder*) et sur NextStep vous la jetterez dans un trou noir (*black hole*) qui se mettra à tourner indiquant qu'il est entrain de détruire le document: ce n'est ni de ergonomie ni de esthétique, c'est juste une question de **philosophie**.

2.2. Visuelle

Une interface graphique met en oeuvre énormément d'algorithmes pour gérer le recouvrement et le rafraîchissement des fenêtres, les entrées sorties d'objets visuels tels que textes, images, vidéos.., les événements générés par l'environnement: clicks et déplacements de souris, caractères saisis au clavier..

"Un graphique vaut mieux qu'un long discours", c'est vraiment le slogan de toutes les interfaces graphiques. La notion icône, empruntée d'un contexte religieux, est très présente, l'utilisateur a de plus en plus tendance à *voir* plutôt qu'à *lire*.

2.3. Rapide

C'est un grand souci des concepteurs qui met en contribution à la fois des programmeurs, des mathématiciens et des constructeurs. En effet, ce type de programmes consomme énormément de temps processeur et de mémoire pour l'affichage et le calcul des images et des courbes, des algorithmes rapides et un matériel puissant sont de nécessité, c'est peut être ce qui explique le retard des ordinateurs personnels dits compatibles IBM PC(TM) à s'habiller de telles interfaces.

A titre d'exemple nous citons l'interface graphique la plus rencontrée sur les PCs: Windows qui utilise un moteur graphique appelé *GDI* (*Graphic Device Interface*), il gère tous les tracés de lignes de courbes et de toute forme géométrique usuelle, il défini aussi ce que Windows appelle *Device Context*, c'est à dire un contexte de périphérique qui peut être une fenêtre à l'écran, une sorite vers l'imprimante ou une connexion réseau. Mac OS, et plus précisément le System 7.0, utilise de son côté un moteur appelé *Quick Draw*, qui est l'analogue du GDI.

3. Aspect visuel de OS-Base

Beaucoup d'éléments de l'interface sont copiés de NextStep, il faut comprendre que la conception de l'aspect de l'interface est généralement confié à des psychologues et des ingénieurs de design spécialisés. Comme on l'a expliquer, il ne s'agit de décorer l'écran pour plaire.

3.1. Le bureau (Desk)

Le bureau est une surface de couleur cyan foncée vide au lancement, c'est dans cette région que les fenêtres des bases ouvertes s'afficheront. A l'extrême droite se trouve une colonne d'icônes qui représente l'ensemble des bases de données crées. Sur NextStep on appelle ceci *Dock*, le *Dock* en anglais veut dire le banc des accusés, les bases de données ouvertes sont assimilées à des accusés qu'on peut appeler à tout moment.

Au coin supérieur droit se trouve un logo de l'ENSI, enfin on l'a dessiné et on l'a appeler comme ça parce que notre école n'a pas malheureusement de logo, il symbolise le contexte du système.

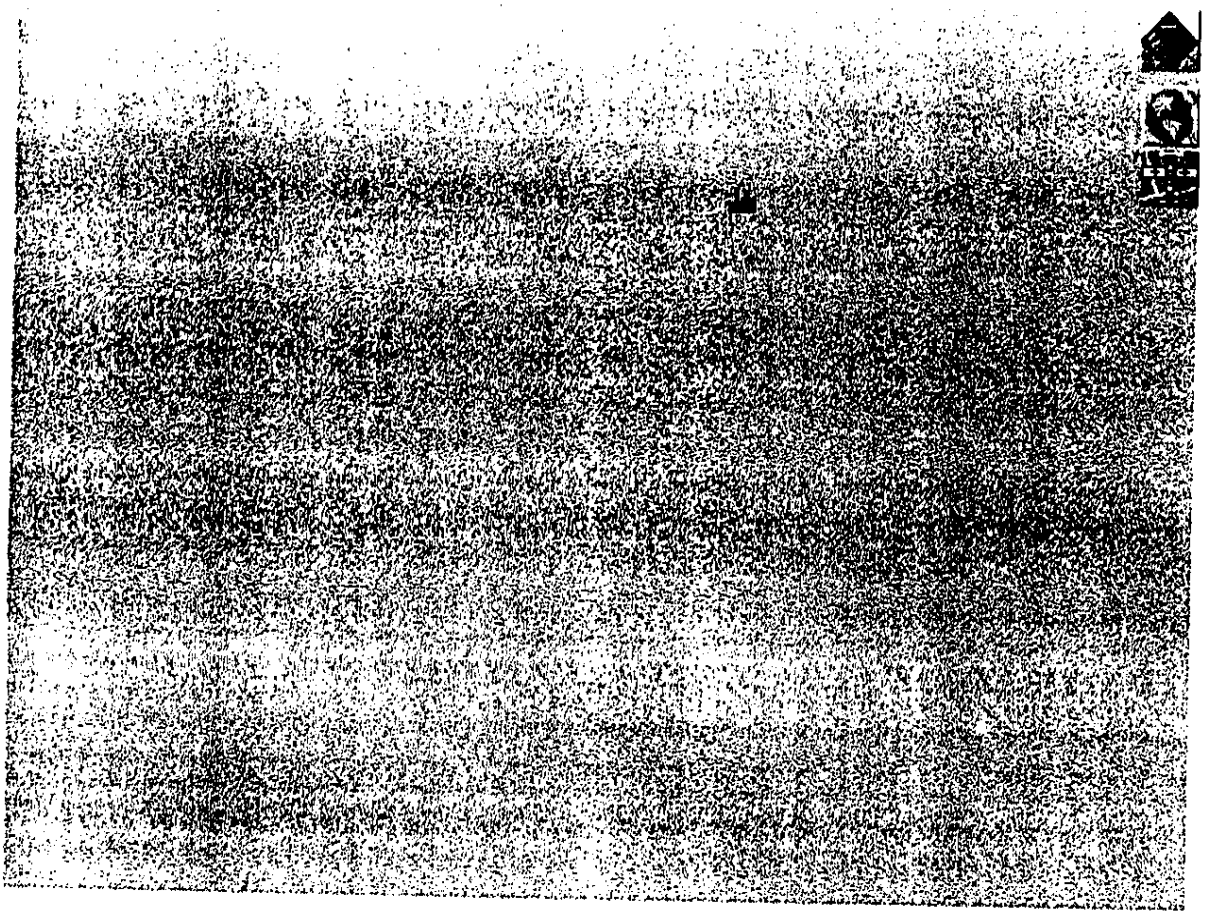


fig. I.1. Bureau de OS-Base

3.2. Les menus

Les menus, encore à la manière de NextStep et d'ailleurs comme toute interface à la norme X-Window sous UNIX, s'activent par un click du bouton doit et s'affichent à l'endroit courant du pointeur de la souris. Ils sont surmontés d'un bandeau noir contenant le titre du menu, et puis les options disponibles. Un click à l'extérieur à pour effet d'ignorer l'appel au menu. Un click sur le bandeau noir permet de déplacer le panneau du menu.

3.3. Les fenêtres

Nous avons pensé à utiliser un gestionnaire de recouvrement de fenêtres primitif: avant d'afficher une fenêtre on sauvegarde la région qui sera recouverte qu'on restituera une fois cette fenêtre est détruite, mais nous en avons renoncer à cause de la lenteur insupportable de routine *GetImage*. Ainsi, nous avons imposer une contrainte: n'afficher des fenêtres que directement sur le bureau.

La fenêtre principale, où les champs de la base de données seront affichés est surmontée d'une barre de titre qui, comme son nom l'indique, contient le titre de la base ouverte.

Au coin supérieur gauche un bouton en X permet la fermeture immédiate de la base ouverte, tout comme Windows 95, de l'aspect du moins.

Au coin supérieur droit sont affichés deux boutons de défilement qui seront grisés si un défilement n'est pas possible. Le fait de griser les options non disponibles est une coutume dans les HCI graphiques.

Du côté droit de la fenêtre principale se trouve une colonne d'icônes qui représentent les actions habituelles dans ce types d'applications: ajout, recherche, suppression ou modification d'un enregistrement.

Comme alternative à la saisie dans une fenêtre fille, désormais non commande pour la raison évoquée plus haut, nous avons réservé une zone de saisie en bas.

La fenêtre *Help on OS-Base*, s'affiche sur demande de l'option Help à partir du bureau. Elle est bien sûr incomparablement plus modeste que les documents d'aide des interfaces professionnelles, dont voici un aperçu.

3.3.1. Les bulles d'aides de Mac OS

Dans le menu Pomme du Mac une option *Activer les bulles d'aide* permet d'activation de l'affichage de petites fenêtres style bande dessinée, contenant une aide rapide sur l'objet couramment pointé par la souris.

3.3.2. Les boutons ? et les contextes d'aides de Windows 95

Chaque fenêtre affichée par Windows 95 contient un bouton « ? » au niveau de sa barre de titre. Presser se bouton, puis cliquer sur n'importe quel objet au sein de cette fenêtre provoque l'affichage d'une rubrique d'aide qui s'y rapporte. De la même façon un click du bouton droit sur un objet peut proposer un option « Qu'est-ce que c'est » qui le définit.

3.4. La souris

C'est le périphérique de pointage le plus populaire, elle fût inventée dans le PARC (*Palo Alto Research Center, Californie, USA*) de Xerox. Le curseur de la souris peut changer de forme selon sa position et l'occupation du système, certaines interfaces offrent des curseurs animés sous forme d'une main qui tapote ou un cheval qui galope..

Nous avons dessiner 4 curseurs que nous n'avons pas tous utilisés : main (*hHand*), sablier (*hSand*), flèche (*hArrow*) et croix (*hCross*).

La modification de la forme du curseur se fait par un appel à `Mouse_LoadCursor` en fournissant en paramètre un handle sur un descripteur de curseur préalablement chargé depuis un fichier *.MCF (Mouse Cursor File)* dont la description est fournie en annexe.

3.5. Les polices de caractère

Les polices sont monnaie courante dans les interfaces graphiques, Windows utilisait des polices BitMap qui sont des matrices de points. A partir de la version 3.1, Windows reconnaît un nouveau format TTF (*TrueType Fonts*) qui supporte les couleurs, les rotations, les déformations.. etc. Avant les polices TrueType, il y avait des outils de polissage comme ATM (*Adobe Type Manager*) très connu dans le monde Macintosh et *FaceLift* de BitStream.

Nous avons utilisés un jeu de polices BitMap qui définit les caractères par une matrice 8x16 pixels. Les styles: small, bold, italic et system sont fournis.

Un fichier FNT contient les 4096 octets définissant le jeu de caractères ASCII (*American Standard Code for Information Iterchange*) étendu. Au démarrage, ces polices sont chargées dans des bloc mémoire, elles sont référencées par le handle de ce bloc. Les procédures évoquant les polices de caractères reçoivent le handle correspondant parmi ceux des polices citées ci-dessus: *hSmallTxt*, *hItalicTxt*, *hBoldTxt* et *hSystem*.

II. Systèmes de Fichiers

A propos de ce chapitre

Dans ce chapitre de tous les aspect de la sauvegarde depuis le support physique jusqu'aux techniques accès évoluées.

"Soulager du travail qui a toujours fatigué l'esprit"

Blaise Pascal

1. Introduction

L'une des tâches les plus ardues accomplies par un système d'exploitation est la gestion des périphériques et encore plus leur organisation logique, cette organisation doit être telle que les applications puissent faire abstraction de l'organe physique avec quoi elles communiquent: c'est al virtualisation du matériel.

On s'intéressera ici aux périphériques de sauvegarde du moment qu'ils présentent l'organe le plus sollicité par les programmes utilisateurs et par le noyau du système d'exploitation lui même. La technologie du support lui même importe peu dans notre étude, on s'acharnera plutôt sur l'organisation logique établie par le système de fichiers.

2. Organisation physique

Les supports les plus fréquents sont les disques magnétiques. Les supports optiques et magnéto-optiques sont en pleine évolution. Un disque dur se présente sous la forme d'un jeu de plateaux entre lesquels se meut un ensemble de têtes tout comme un peigne, d'ailleurs c'est comme ça qu'on les appelle.

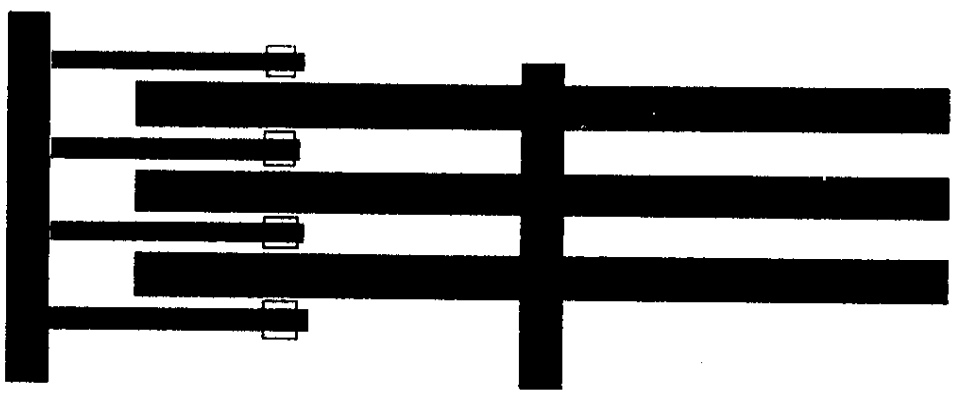


fig. II.1. Disposition des têtes de lecture/écriture.

Chaque plateau est subdivisé en pistes et en secteurs en guise de système de coordonnées pour référencer des zones physiques sur le disque. L'ensemble formé par

une piste sur l'un des plateaux et ses projetées sur les autres plateaux forme un cylindre.

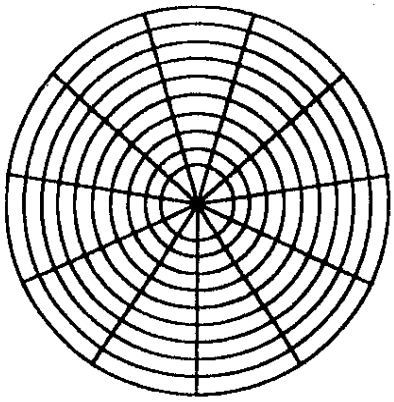


fig. II.2. Organisation en pistes et secteurs

3. Disposition des secteurs

Un secteur physique contient le plus souvent 512 octets, rappelons là qu'il s'agit d'une contrainte imposée par l'électronique du contrôleur. La plupart des systèmes d'exploitations réservent le tout premier secteur (secteur 0) du support pour le démarrage: c'est le secteur de *boot*, sur ce secteur est inscrit un petit programme pour charger le noyau du système à l'initialisation. Par exemple, sur les systèmes DOS les trois premiers octets sont une instruction de saut de la forme:

```
JMP    XXXXh
```

Ce saut s'effectue vers un programme de quelques centaines d'octets au sein du secteur une fois chargé en mémoire, c'est là que les virus dits de boot viennent inscrire leur code viral.

Le reste du secteur de boot contient des informations sur l'organisation physique du support: type, capacité, taille d'un secteur, nombre de secteurs physiques dans un secteurs logique..

Les secteurs suivants contiennent des informations relatives au système de fichiers. Par exemple, les systèmes DOS y inscrivent la FAT (*File Allocation Table*), puis le répertoire racine qui est une liste d'entrées de fichiers et de répertoires de la racine « \ ».

4. Exemples de systèmes de fichiers

Pratiquement tous les systèmes DOS, car il y en a beaucoup (Novell(TM) DOS, Digital Research(TM) DOS, PC DOS, MS DOS..), utilisent une FAT. La FAT est une liste chaînée de clusters (secteurs logiques) dont les entrée est un numéro de cluster, les clusters défectueux et les fins de liste de fichiers sont indiqués par des valeurs spéciales.

En fait, MS-DOS reconnaît aussi les techniques accès aux fichiers à la manière de CP/M: les FCB (*File Control Blocks*), ils n'autorisent pas la structure arborescente du répertoire reconnue sur DOS depuis la version 2.0.

OS/2 reconnaît le système FAT ainsi qu'un système de fichiers HPFS (*High Performance File System*). Windows NT utilise un système de fichiers NTFS (*New Technology File System*), une autre catégorie concerne les systèmes de fichiers répartis ou distribués: le NFS (*Network File System*).

UNIX quant à lui contient une ingéniosité dans son système de fichiers: c'est le fait d'inclure les périphériques dans le système de gestion de fichiers. En effet, les fichiers *device* représentent comme il faut un périphérique duquel et sur lequel on peut lire et écrire. Ainsi le fichier */dev/fd0* représente la première disquette, */dev/hda1* le premier disque dur ..etc. Les DOSs reconnaissent cette notion de représenter les périphériques par des fichiers, ainsi CON est l'entrée/sortie standard, PRN l'imprimante, AUX le port série ..etc. Mais cette représentation n'est par aussi bien intégrée dans le système de fichiers comme sur UNIX.

5. Profil du système de fichiers implanté

5.1. Secteur de boot

C'est pratiquement le même que celui de DOS. Ceci est sa structure:

Adresse	Contenu
0000h	Premier block libre
0003h	Chaîne contenant le nom du fabriquant
000Bh	Octets par secteurs (512)
000Dh	Secteurs par cluster (1)
000Eh	Nombre de secteurs réservés (0)
0010h	Non utilisé
0011h	Nombre d'entrées dans le répertoire racine (26)
0013h	Nombre de secteurs dans le volume 2880 sur les disquettes 3" ½ Haute Densité 1440 sur les disquettes 3" ½ Double Densité
0015h	Descripteur du support F0h sur les disquettes 3" ½ HD F9h sur les disquettes 3" ½ DD
0016h	Non utilisé
0018h	Non utilisé
001Ah	Non utilisé
001Ch	Non utilisé
001Eh-01FFh	Non utilisés

Tab. II.1. Structure du secteur de boot

5.2. Le répertoire

5.2.1. Structure d'une table de descripteurs

La fameuse arborescence des fichiers connue des utilisateurs de DOS et de UNIX est représentée par un arbre de tables de fichiers dont la racine est le secteur numéro 1. Les entrées de cette table sont des structures du type:

```
type FSEntry is record
        fName       : array(1..15) of byte_integer;
        fProperties : byte_integer,
        iNode       : integer;
        eStatus     : byte_integer;
end record;
```

La taille d'une structure est de 15+1+2+1=19 octets, ainsi le répertoire racine contiendra au maximum 26 fichiers et répertoires.

Chaque entrée est un descripteur de fichier qui contient:

- fName : Un nom de fichier sur 14 caractères.
- fProperties : Précise s'il s'agit d'un fichier ordinaire ou d'un répertoire, UNIX reconnaît d'autres types de fichiers tels les liens symboliques ou les *pipes*.
- iNode : Un numéro d'un noeud d'information (voir plus loin) contenant des informations relatives au fichier/répertoire.
- eStatus : Une valeur qui spécifie l'état de l'entrée du répertoire: libre, supprimée, occupée ou défectueuse.

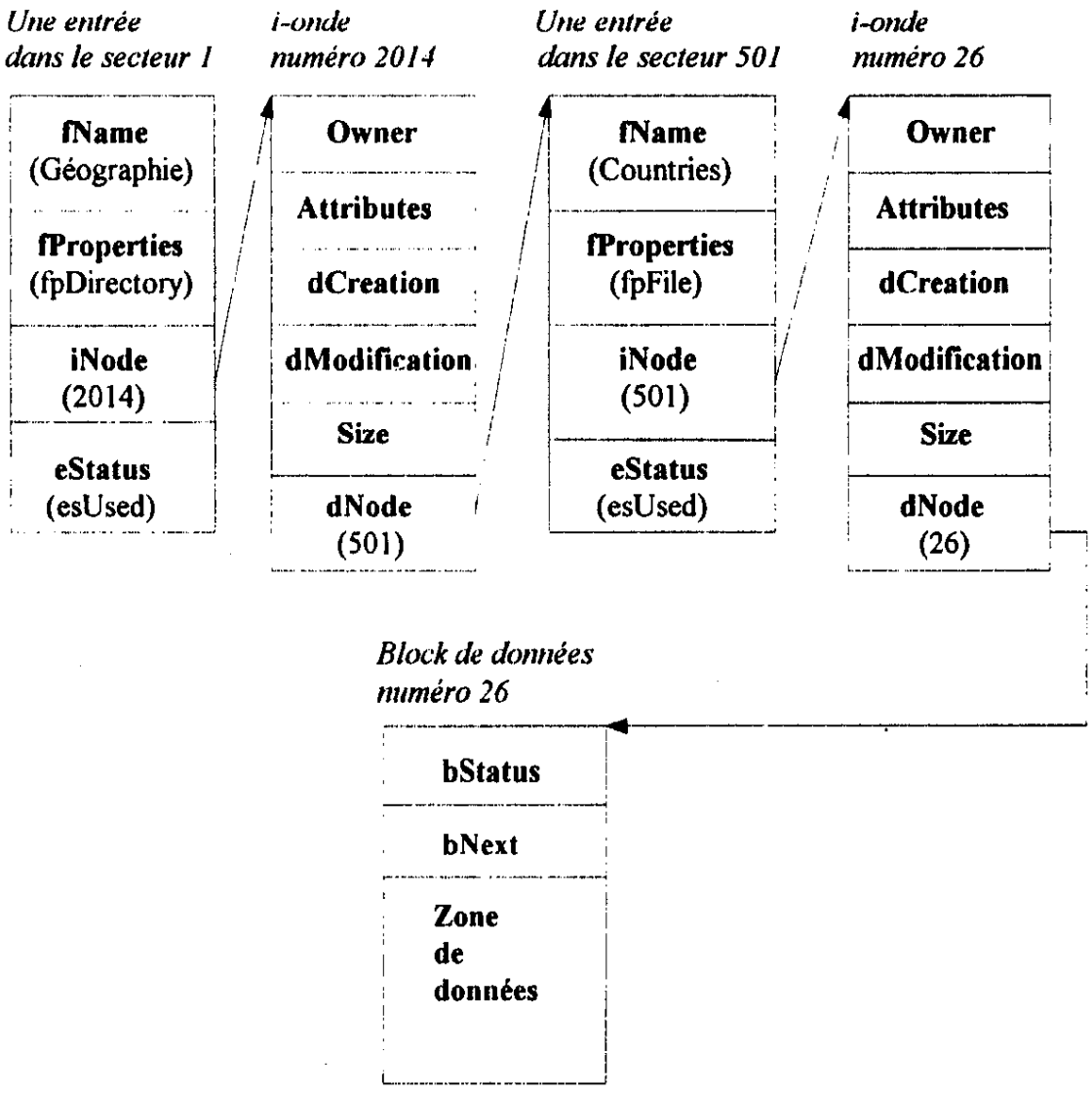


Fig. II. 3. Organisation logique et chaînage des blocks
(ici un accès à /Géographie/Countries*)*

5.2.2. Structure d'un i-onde

Un noeud d'information contient des informations sur le fichier ou le répertoire, c'est une structure du type:

```
type struct_iNode is record
      Owner          : integer;
      Attributes     : integer;
      dCreation      : long_integer;
      dModification  : long_integer;
      size           : long_integer;
      dNode          : integer;
end record;
```

C'est sur cette structure que les contrôles de droits accès et les modifications apportées au fichier sont effectuées.

5.3. Primitives accès

Un système de fichiers doit mettre à la disposition des applications des primitives accès aux fichiers de manière à pouvoir faire abstraction du support physique.

5.3.1. Création (fsCreate)

La procédure de création reçoit un handle (voir la gestion de mémoire) sur un fichier, un nom et un mot de droits à la manière des trois chiffres en base 8 de UNIX. Elle procède à la vérification de la syntaxe du nom et la recherche du répertoire spécifié, ensuite elle cherche une entrée vide dans ce répertoire et la réserve au fichier. A la sortie de la routine, le fichier est ouvert.

5.3.2. Destruction (fsKill)

Un handle de fichier est fourni, la procédure commence par libérer les blocks qu'il a réservé puis elle supprime son entrée dans le répertoire et enchaîne les blocks libérés avec les autres blocks libres. Finalement l'espace mémoire alloué par le handle est libéré. cette procédure suppose que le fichier est ouvert.

5.3.3. Ouverture (fsOpen)

Reçoit un handle et un nom de fichier. A l'appel, elle procède à la recherche du fichier, puis vérifie les droits accès et charge le i-onde du fichier si accès est autorisé.

5.3.4. Fermeture (fsClose)

Reçoit un handle pour libérer les ressources allouées et mettre-à-jour le i-onde correspondant.

5.3.5. Ecriture (fsWrite)

Reçoit un handle de fichier, un handle de block mémoire et la taille du block à écrire. Elle vérifie le droit d'écriture et si l'écriture a lieu à la fin du fichier elle met-à-jour la taille et alloue de nouveaux blocks si besoin est.

5.3.6. Lecture (fsRead)

Reçoit un handle de fichier, un handle de block mémoire et la taille du block à lire. Elle vérifie le droit de lecture et si la lecture est permise, elle commence le transfert vers le tampon spécifié.

5.3.7. Positionnement du pointeur de fichier (fsSeek)

Positionne le pointeur de fichier à une position donnée pour préparer une lecture ou une écriture. Elle reçoit un handle de fichier et un entier long, et modifie les valeurs courantes du secteur et d'offset dans ce secteur en conséquent tout en contrôlant les débordements.

5.4. Structure de données évoluée

Les applications qui sollicitent le plus le système de fichiers sont les SGBD (*Systèmes de Gestion de Bases de Données*). Elles utilisent des techniques très avancées pour gérer des volumes de données gigantesques.

6. Les techniques utilisées par les SGBDs

Dans cette partie du projet on essaie de mettre en œuvre un système de gestion de base de données à la fois simplifié, dans la mesure où il offre peu de possibilités à l'utilisateur, et ambitieux du fait que l'utilisateur peut stoker des images comme il est libre de choisir la taille des enregistrements qu'il souhaite créer. Pour cela on a pensé à associer à chaque base de donnée lors de sa création un fichier .FRM (*Form File*) dans lequel sont stockées les informations sur les différents champs de l'enregistrement de la base ainsi que le type d'accès à ces enregistrements et la clé de recherche.

Le fichier .FRM cité plus haut définit les champs de la base, c'est un fichier de structures du type:

```
type FormFileStructure is record
   FieldType   : byte_integer;
   Designation : array(1..11) of byte_integer;
   Size        : natural;
   xPos, yPos  : natural;
end record;
```

- FieldType : renseigne sur le type de l'information contenue dans le champs correspondant pour prendre les dispositions nécessaires à son affichage. La valeur 0 indique une image, 1 une chaîne.
- Designation : est le nom du champ, il est sur 10 caractères tout comme dBase III.
- Size : taille du champs en octets (caractères)
- xPos, yPos : coordonnées du coin supérieur gauche du lieu d'affichage.

6.1. Les types d'organisation

L'expression "organisation de fichier" désigne généralement la façon dont les enregistrements sont rangés dans un fichier. On en distingue principalement:

6.1.1. Organisation séquentielle

Dans une telle organisation, les enregistrements sont placés les uns à la suite des autres, à partir du début du fichier dans l'ordre où ils sont émis par le programme de création. Dans certaines conditions, les enregistrements pourront être placés, consécutivement, à la fin du fichier.

On a opté pour l'organisation suivante: les deux premiers octets du fichier de données contiennent le nombre d'enregistrements supprimées dans le fichier (zone *SupRecNbr*). Le reste du fichier est une suite d'enregistrements débutés par des marques ('*' ou '$') indiquant s'ils sont supprimés ou non.

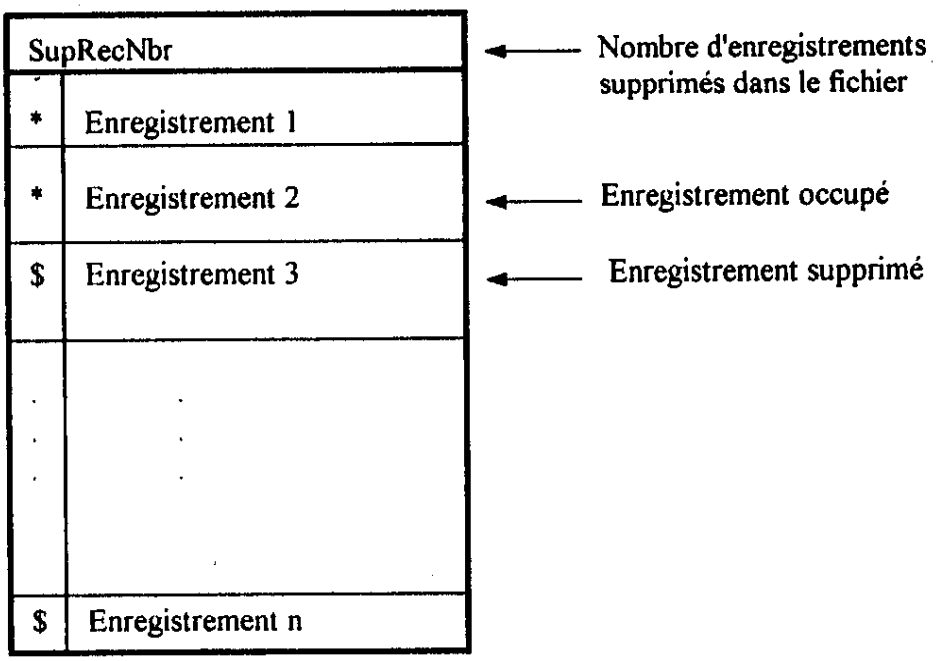


Fig. II.3. Structure logique d'un fichier séquentiel

Pour cette organisation, un ensemble de primitives spécifiques a été implémenté:

- *seqInitialize*
  Comme son nom l'indique, cette procédure met à zéro le compteur du nombre d'enregistrements supprimés et l'écrit au début du fichier (dans les deux premiers octets).

- *seqWrite*
  Cette procédure effectue l'écriture effective des enregistrements et ce à la fin du fichier tout en ajoutant au début de l'enregistrement la marque '*' indiquant qu'il est occupé.

- *seqResearch*
  Cette fonction effectue la recherche d'un enregistrement dans le fichier de données en se basant sur une clé. Elle retourne la position de l'enregistrement par rapport au début du fichier au cas où la recherche est positive, sinon elle retourne -1 indiquant que l'enregistrement désigné n'a pas été trouvé.

- *seqRead*
  Cette procédure appelle la fonction seqResearch qui lui fournit la position de l'enregistrement à lire s'il existe. A partir de cette position, seqRead lit un bloc de taille RecordSize (Taille de l'enregistrement).

- *seqDelete*

  seqDelete appelle la fonction *seqResearch* qui lui fournit l'emplacement de l'enregistrement à supprimer. Une opération de suppression consiste tout simplement au changement de la marque '*' par la marque '$' indiquant que l'enregistrement n'est plus occupé. Après la suppression d'un enregistrement le fichier est mis à jour en incrémentant la valeur de la variable *SupRecNbr* et en la réécrivant au début du fichier. *seqDelete* contrôle la valeur de *SupRecNbr* (nombre d'enregistrement supprimées) qui en atteignant une valeur *MaxSupRec* fixée au moment de la création du fichier (cette valeur dépend de la taille de l'enregistrement) déclenche une procédure d'optimisation de l'espace disque *DiskSpaceOptimizer*.

- *DiskSpaceOptimizer*

  Cette procédure effectue la lecture enregistrement par enregistrement du fichier séquentiel et suivant la marque de suppression ('*' ou '$') elle effectue ou non le transfert de l'enregistrement dans un autre fichier intermédiaire nommé **OS_ENSI.DSO**. Une fois que les enregistrements ayant la marque '*' ont été transférés à **OS_ENSI.DSO** on supprime le fichier de donnée puis on renomme le fichier **OS_ENSI.DSO** en l'ancien nom du fichier de données. La variable *SupRecNbr* est remise à zéro. Ainsi on a effectué une optimisation de l'espace occupé par le fichier séquentiel.

6.1.2. Organisation directe

C'est l'organisation dans laquelle les enregistrements sont placés dans le fichier à des emplacements déterminés par leur rang ou leur numéro par rapport au début du fichier.

L'organisation directe à la quelle on a opté consiste à définir deux zones au début du fichier, l'une *First_Free* et l'autre *First_Disp*:

- La zone *First_Free*: initialisée à 0 à la création du fichier, cette zone fournit le numéro du dernier enregistrement supprimé (devenu libre). Cette zone sera d'abord consultée avant toute décision d'affectation (écriture) d'un nouvel enregistrement.

- La zone *First_Disp*: initialisée à 1 à la création du fichier, cette zone fournit à tout instant le numéro du prochain enregistrement disponible. Lorsque le fichier est vide, First_Disp pointe sur le premier enregistrement du fichier.

Après ces deux zones viennent se succéder les enregistrements de données. Dans chaque enregistrement de données, deux zones sont réservées, l'une *mark*, pour indiquer qu'un enregistrement est libre ('*') ou pas ('$'), et l'autre *Next_Free*, pour chaîner entre eux les différents enregistrements supprimés, donc disponibles pour une réutilisation. Cette caractéristique est jugée capitale puisqu'elle permet d'optimiser l'occupation du fichier par une utilisation systématique des "trous" laissées par les suppressions.

La chaîne des enregistrements supprimés a pour pointeur de tête de chaîne, la zone *First_Free*. Chaque fois qu'un enregistrement est supprimé, son numéro est inséré en tête de chaîne. Ainsi, le dernier enregistrement supprimé sera le premier alloué lors d'une création de nouvel enregistrement.

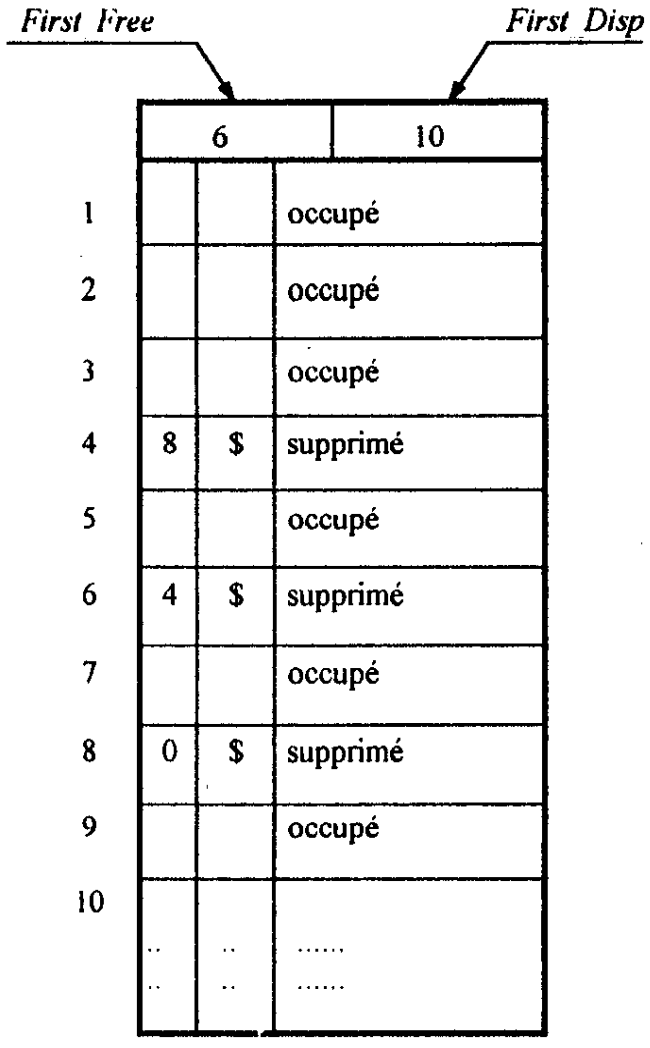


Fig. II.3. Structure logique d'un fichier direct

Ici la zone *First_Free* contient la valeur 6 qui représente l'adresse relative ou numéro du dernier enregistrement supprimé. Si nous nous reportons à l'enregistrement n° 6, nous constatons que celui-ci est supprimé (zone *Mark* = $) et que la zone *Next_Free* contient le numéro de l'enregistrement supprimé suivant (adresse relative 4).

L'enregistrement 4 renvoie quant à lui sur l'enregistrement n° 8 dont la zone *Next_Free* à zéro indique qu'il est le dernier de la chaîne.

Comme pour accès séquentiel, nous avons implémenté des primitives accès à ce type.

- *dInitialize*

  Cette procédure initialise le fichier de donnée sen mettant à zéro la zone *First_Free* et à un la zone *First_Disp*. Comme elle effectue la création d'un fichier d'index dans lequel seront enregistrées la clé et la position dans le fichier de données des enregistrements.

- *dWrite*

  dWrite est une procédure qui effectue l'écriture des enregistrements dans le fichier de donnée et la mise à jour du fichier d'index. Avant une opération d'écriture *dWrite* doit consulter la zone *First_Free*, si elle n'est pas à zéro alors l'écriture sera effectuée à la position que *First_Free* fournit, mais si elle est à zéro alors *dWrite* consulte la zone *First_Disp* et écrit à la position que cette dernière fournit, après l'écriture la zone *First_Disp* est mise à jour.

- *dResearch*

  Cette fonction effectue une recherche séquentielle dans le fichier d'index et retourne la position de l'enregistrement cherché s'il est trouvé, sinon, elle retourne -1.

- *dRead*

  Elle effectue une simple lecture depuis la zone indiquée par la fonction *dResearch*.

- *dDelete*

  Nous savons qu'une suppression consiste à rendre disponible un enregistrement en lui ajoutant la marque de suppression '$'. Le problème est évident, sauf qu'il faut mettre en œuvre une technique permettant de rendre cet enregistrement accessible lors de futures demandes de création. On a opté pour la suivante: la zone *First_Free* est mise à zéro chaque fois qu'il n'y à aucun enregistrement supprimé. Sinon, cette zone contient le numéro du dernier enregistrement supprimé qui dans sa zone *Next_Free* contient le numéro de l'enregistrement supprimé avant lui et ainsi on construit une chaîne des enregistrements supprimés dont le numéro de la tête est dans *First_Free*. On remarque que l'insertion se fait en tête de chaîne.

6.1.3. Organisation indexée

Dans l'organisation indexée, les enregistrements sont écrits les uns à la suite des autres. Simultanément, il est créé un index ou répertoire contenant, pour chaque enregistrement, l'indicatif ou clé de celui-ci, ainsi que sa position relative par rapport au début du fichier. A l'intérieur de l'index les indicatifs sont maintenues classées par ordre croissant des valeurs.

6.2. Les types d'accès

Une fois créé sous l'une des trois organisations, les différents enregistrements qui le composent peuvent être exploités par un programme au moyen d'une des trois méthodes d'accès ci-dessous.

6.2.1. Accès consécutif

Les enregistrements sont accessible dans l'ordre physique où ils ont été écrits, du début du fichier jusqu'à la fin. Un enregistrement i ne peut être atteint que si les (i-1) enregistrements l'ont été.

6.2.2. Accès sélectif

Les enregistrements sont disponibles dans un ordre au choix de l'utilisateur. Formellement, l'enregistrement demandé est défini par son numéro ou sa position par rapport au début du fichier :

- soit directement en fournissant ce numéro;
- soit indirectement en fournissant une clé et en consultant un répertoire ou indexe.

6.2.3. Accès indexé

Conceptuellement, dans l'accès indexé, les enregistrements sont accessible en ordre croissant ou décroissant, sur l'indicatif par l'intermédiaire d'un indexe ou répertoire.

III.
Gestion de la Mémoire

A propos de ce chapitre

C'est aussi un parcours de la hiérarchie logique de la mémoire depuis le composant jusqu'aux techniques d'allocation avancées.

« ..because MS-DOS is a piece of grabs »

Jim Manzi [*]

1. Introduction

La mémoire est le composant le plus sollicité par le processeur, sur plusieurs architectures matérielles l'organisation de la mémoire dépend très étroitement du processeur installé. C'est aussi le point de rencontre de tous les processus, aucun ne peut s'en passer. Avec le temps processeur elle présente la ressource système la plus «vitale».

2. Présentation du composant

Un chip mémoire se présente souvent sous la forme d'une barrette SIMM *(Single Inline Memory Module)*, on peut en distinguer plusieurs classes selon la technologie utilisée, le domaine d'utilisation, l'opérabilité..

La plupart des systèmes PC utilisent des mémoires DRAM (*Dynamic Random Access Memory*), c'est-à-dire une mémoire à accès aléatoire dynamique, sa "dynamicité" réside dans le fait qu'elle nécessite d'être rafraîchie (réécrite) cycliquement. Une autre technologie est dite Statique, elle est très rapide (temps d'accès de 5 à 20ns contre 60 à 80ns pour la DRAM) et ne nécessite pas de rafraîchissement, de tels composants servent en tant que mémoire cache dans les PCs ou en tant que mémoire centrale dans les stations de travail.

3. Organisation de la mémoire sur PC

L'espace d'adressage des processeurs intel 80x86 en mode réel est de 1 Mo duquel il faut soustraire l'espace réservé au matériel : fenêtres XMS (zone accès à la mémoire étendue au format *eXpanded Memory Specifications*), segment réservé à la carte graphique, zone de routines BIOS (*Basic Input/Output System*)..etc.

[*] ***Jim Manzi*** est président de Lotus Developpement Corp., il répliqua ainsi lorsqu'on lui demanda la raison du choix de OS/2 comme plate-forme de Lotus Notes.

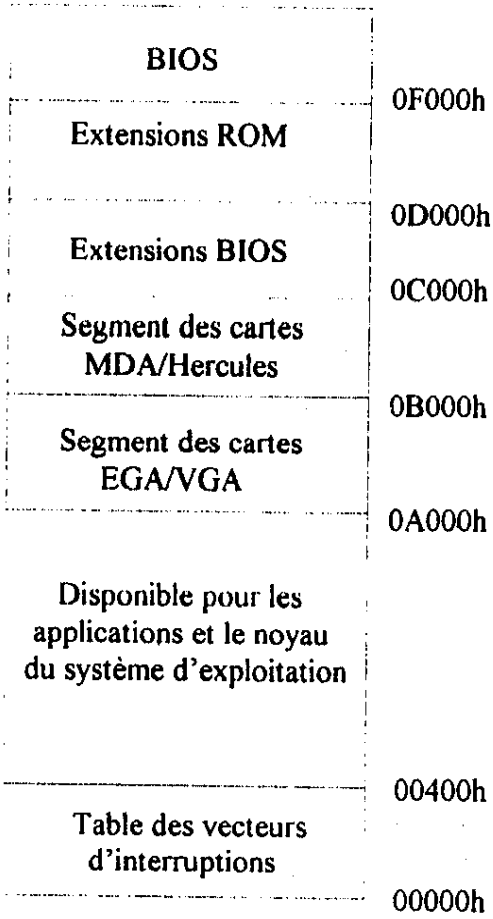


Fig. III.1. Organisation de la mémoire sur PC

4. Techniques de gestion de la mémoire sur PC

En mode réel, les processeurs intel de la famille 80x86 utilisent un procédé de segmentation de la mémoire, le principe est de séparer le code, les données et la pile. En mode protégé, ils utilisent des sélecteurs et des descripteurs de blocs avec des protections contre les accès non autorisés.

4.1. Segmentation en mode réel

Les segments ont 64Ko de taille et sont alignés sur des paragraphes (un segment commence toujours à une adresse multiple de 16). Pour référencer une position mémoire ils utilisent des adresses (logiques) au format SSSSh:OOOOh, le calcul de l'adresse physique se fait comme suit:

Addr_Physical = Segment x 10h + Offset

Une adresse physique sur 20 bits ne permet d'adresser que 1 Mo duquel il faut soustraire l'espace réservé au matériel : fenêtres XMS (zone accès à la mémoire étendue au format *eXpanded Memory Specifications*), segment réservé à la carte graphique, zone de routines BIOS (*Basic Input/Output System*)..etc.

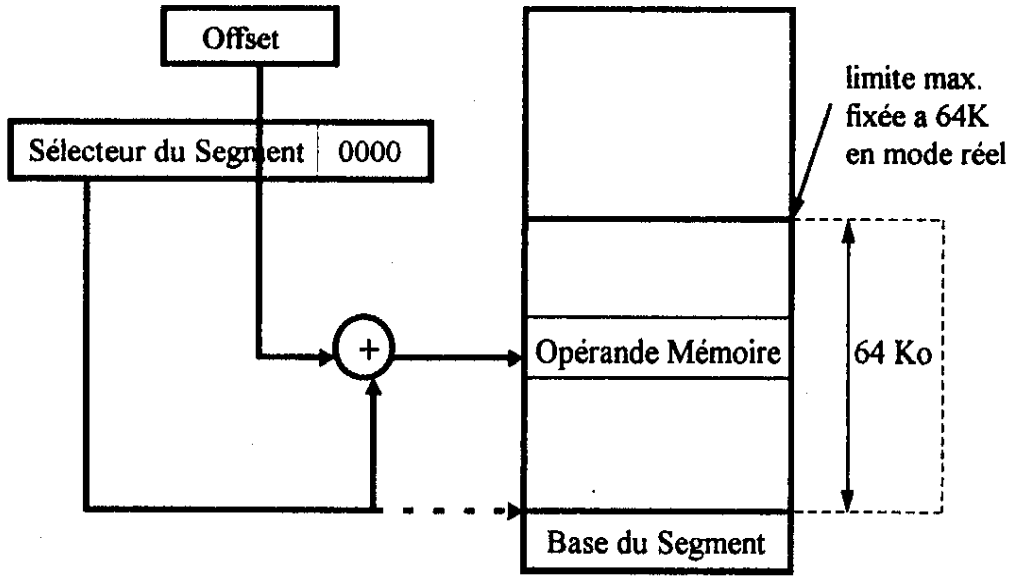


Fig. III.2. Adressage en mode réel (Document intel 240950-011)

4.2. Segmentation en mode protégé

Le mode protégé n'est disponible qu'à partir du processeur i80286, mais les véritables progrès sont apparus avec le i80386 qui supporte en plus l'adressage 32 bits.

En mode protégé, une valeur sur deux bit dite droit de privilège ou PL (*Privilige Level*) est indiquée dans tous les segments et les accès aux périphériques. Les registres de segments (sur 16 ou 32 bits) sont utilisés pour indiquer une entrée dans une table de sélecteurs, les deux bits de poids faible servent pour indiquer le niveau de privilège de l'émetteur de la requête accès au segment, c'est le RPL (*Requested Privilige Level*). Chaque entrée dans la table des descripteurs contient les droits accès (access rights), la taille (limit) qui peut atteindre 4 Go et le début du segment (base address).

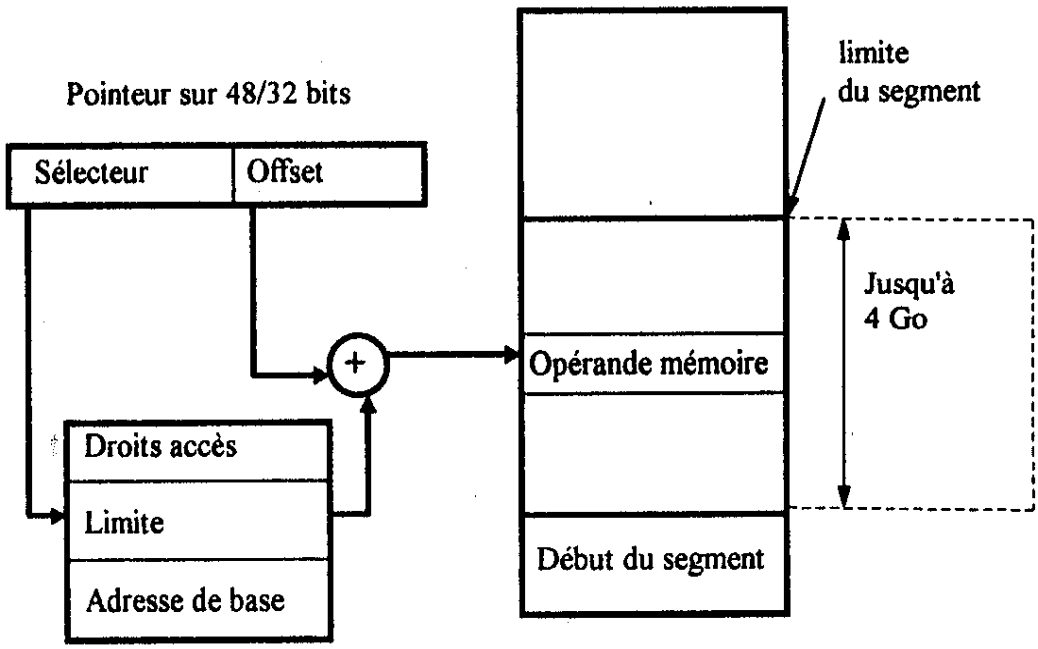


Fig. III.3. Adressage en mode protégé (Document intel *241944-010)*

4.3. Stratégie d'allocation de mémoire sur les systèmes DOS

DOS utilise une méthode très simple pour gérer l'allocation de la mémoire: les blocks sont chaînés et contiennent des entêtes assurant ce chaînage et renseignant sur le bloc. Dans la terminologie du DOS, ce bloc s'appelle un MCB (*Memory Control Block*) et a la structure suivante:

Adresse	**Contenu**
0	'Z' si dernier MCB 'M' sinon
1	Adresse du segment PSP (*Program Segment Prefix*)
3	Taille des paragraphes en mémoire (jusqu'au prochain MCB)
5	Non utilisé
16	Début de l'espace mémoire alloué

Fig. III.4. Structure d'un MCB

5. Notre gestionnaire de mémoire

Nous l'avons inspiré de l'organisation de la mémoire dans le mode protégé des processeurs 386 et supérieurs. En effet, ces processeurs offrent un support pour un gestionnaire de mémoire virtuelle. En effet, dans l'octet des droits accès, le bit numéro 7 est dit bit de présence. Il est à 1 si le segment existe effectivement(*), sinon le segment devra être rechargé depuis un périphérique d'échange (*swap device*).

L'allocation/libération proprement dite se fait par les routines classiques de MS-DOS, ce sont des services classiques de l'interruption 21h:

Fonction	Rôle
48h	Alloue de la mémoire
49h	Libère de la mémoire
4Ah	Modifie la taille d'un bloc mémoire
58h	Lit/fixe la stratégie d'allocation

Fig. III.5. Fonctions d'allocation de mémoire sous DOS

Notre gestionnaire de mémoire utilise ce qu'on appelle dans la terminologie des systèmes d'exploitation un algorithme LRU (*Least Reacently Used*).

5.1. Organisation des blocs

Une table de descripteurs contient des entrées associées à chaque bloc, une entrée est une structure du type

```
type TBlkDescriptor is record
            addr   : system.address;
            size   : natural;
            status : byte_integer;
            age    : integer;
            swapID : Long_integer;
end record;
```

Ainsi chaque bloc est définit par un déscripteur de la façon suivante:

- *addr* est l'adresse du bloc en mémoire.
- *size* est sa taille en paragraphes (16 octets)
- *status* est son état
  bit 0 : Bit d'utilisation, vaut 1 si l'entrée est utilisée
  bit 1 : Bit *IsStatic*, permet d'exclure ce bloc du processus de *swapping* s'il est à 1.
  bit 2 : Bit de présence, vaut 1 si le segment est présent en mémoire.

(*) *No memory mapping* dans document original.

- *age* est l'"age" du segment, c'est une valeur qui renseigne sur la fréquence accès à ce segment. Il est utile pour l'optimisation du choix du bloc à transférer sur le support d'échange

5.2. Primitives d'accès à la mémoire

Les applications utilisant le gestionnaire de mémoire que nous sommes en train de décrire peuvent se servir d'un jeu de primitives pour effectuer les opérations habituelles sur un bloc en mémoire.

Notons là que les tailles des blocs sont exprimées en paragraphes de 16 octets, nous avons préféré laisser l'expression de la taille en paragraphes dans le but de rappeler l'utilisateur du mécanisme d'allocation de DOS qui se déroule "en coulisses".

Les index des entrées dans la tables de descripteurs sont des valeurs de type *long_integer*.

5.2.1. Allocation (malloc)

Tout comme en C, cette fonction est appelée en fournissant la taille du bloc requis en paramètre, elle retourne un index d'une entrée dans la table des descripteurs.

Si l'espace mémoire libre disponible n'est pas suffisant, *malloc* demande le descripteur le plus "vieux" à *GetOldest* pour le transférer sur disque et libérer l'espace qu'il occupait, elle boucle ainsi jusqu'à ce que assez d'espace libre soit disponible.

5.2.2. Libération (FreeMem)

Cette procédure prend en paramètre un déscripteur de bloc; s'il n'est pas présent en mémoire, elle le charge et libère son espace mémoire et l'entrée qu'il occupait. Ceci peut paraître absurde, mais il faut penser au bloc sur disque qui occuperait de l'espace inutilement si *FreeMem* ne le recharge pas.

5.2.3. Lecture/écriture d'un octet (GetByte/SetByte)

GetByte est une fonction qui reçoit un handle de bloc et un offset, elle renvoi l'octet se trouvant à l'offset fourni dans le bloc décrit par le handle.

SetByte modifie un octet donné dans le bloc.

5.2.4. Lecture de l'adresse physique (GetAddress)

Renvoi l'adresse de début du bloc en mémoire, l'appel de cette fonction déclenche automatiquement le chargement du bloc dans la mémoire vive.

Cette primitive, ainsi que *GetByte* et *SetByte* constituent les accès à un bloc, elles sont responsables de la gestion du paramètre *age*. Lors de accès à un bloc, l'age des tous les autres blocs est incrémenté et celui du bloc concerné est remis à zéro: il devient par conséquent le bloc "le plus jeune".

5.2.5. *Routines internes*

- *SwapToDisk*
  Appelée par malloc pour transférer un block sur disque. Le périphérique d'échange reconnaît deux tailles limites: 1Ko (64 paragraphes) et 32Ko (2048 paragraphes), cette contrainte revient à la limite imposée par le type integer de ADA, s'il reconnaissait le type UINT (*Unsigned Integer*) on aurait pu pousser cette limite à la taille d'un segment entier.

- *UnSwapFromDisk*
  Restitue un bloc vers la mémoire en utilisant la même primitive d'allocation (*malloc*).

- *IsFree*
  Renvoi une information sur l'occupation ou non de l'entrée fournie en paramètre.

- *IsPresent*
  Renvoi une information sur la présence ou non du bloc fourni en paramètre.

Conclusion

Nous avons travaillé, et nous travaillons encore, sur beaucoup de thèmes assez diversifiés embrassant la discipline des systèmes d'exploitation. Ceci nous a permis de faire face à une multitude de problèmes et de difficultés d'ordre matériel et logiciel revenant aux restrictions du langage ADA entre autres.

Nous jugeons que les solutions proposées furent assez adaptées aux besoins.

Bien sûr un tel travail fût très pénible surtout sous les contraintes de temps imposées, pourtant nous estimons avoir accompli avec succès 90% de notre objectif prévu en mi-juillet.

Le reste du travail, qui concerne surtout des mise en forme des procédures (le noyau est fonctionnel), se déroule encore et doit s'achever dans la semaine en cours.

Nous jugeons que le présent travail répond aux exigences d'un stage de programmation pour des élèves ingénieurs de la classe II 1. Nous ne prétendons pas avoir réaliser un noyau d'un système d'exploitation mais, du moins on a en touché les concepts et les fonctionnalités de prés.

Bien que OS-Base fonctionne correctement, des améliorations au niveau des contrôles d'erreurs, une gestion plus poussée des entrées/sorties et des optimisations générales sont nécessaires pour satisfaire une utilisation professionnelle.

Achevé à Tunis, le 16 septembre 1996.

Annexe I

Notes sur la loi du Copyright

Marques commerciales citées

Microsoft, Windows et MS-DOS sont des marques commerciales de Microsoft Corp.
Redmond, WA 98052

Aldus PhotoStyler est une marque commerciale de Aldus Corp.

UNIX est une marque déposée de AT&T (Bell laboratories).
SPARC est une marque déposée de Sun Microsystems.
Macintosh est une marque déposée de Apple Computer, Inc.

Intel est une marque déposée de Intel Corporation.

Le gestionnaire de souris fourni

Nous rappelons que nous fournissons le gestionnaire de souris "*Microsoft Mouse driver*" en tant que copie de celui installé sur les ordinateurs de l'*ENSI*. Du fait, nul n'a le droit de l'utiliser hors de l'école.

Le programme lui même

La copie intégrale ou partielle du code source est permise pour tout étudiant de l'*ENSI*. Tout étranger à l'école doit mentionner notre propriété avant chaque usage.

Annexe II

Conditions du Travail

Environnement matériel

Ce travail a été réaliser sur un PC utilisant un processeur int$_{el}$ 486 cadencé à 66 Mhz avec 8 Mo de mémoire vive et 256 Ko de mémoire cache. Pour l'affichage, on a utilisé une carte graphique à base d'un chip Cirrus Logic 5424 avec 1 Mo de mémoire vidéo sur bus Vesa.

Le présent rapport fût tiré sur une imprimante Hewlett Packard(TM) DeskJet 600.

Environnement logiciel

Pour la génération de code nous avons utilisé un compilateur ADA de chez Meridian Softaware, le *adAvantage ada compiler* sous DOS.

Les images ont été fabriquées et traitées sous Aldus PhotoStyler 2.0 et MS-PaintBrush tournant tous les deux sous Microsoft Windows 3.1

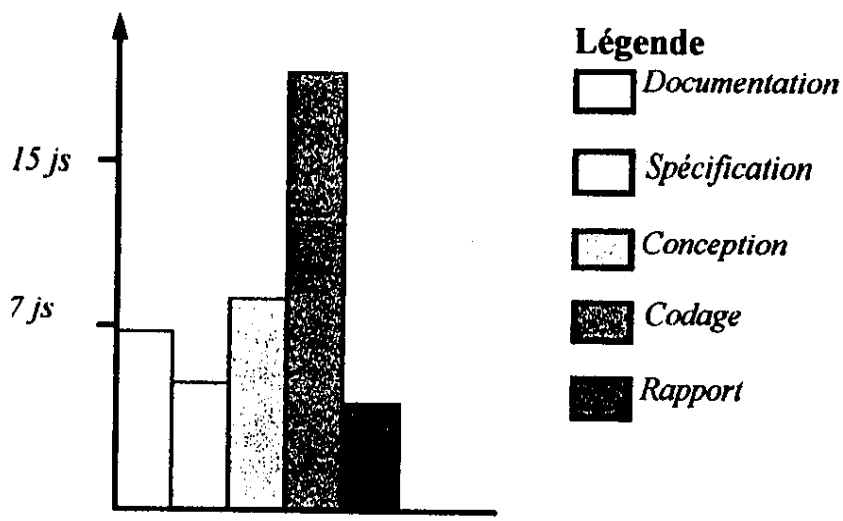
Les icônes ont été fabriquées et traitées sous Borland Resource Workshop sous Windows.

Le présent rapport fût rédigé et imprimé sur MS Word pour Windows 6.0 édition arabe.

Volume temporel

Les travaux ont commencés le 1 août 1996 après tant d'ennuis rencontrés lors de la recherche d'un logement à la capitale. Ils se sont déroulés à raison de 10 heures par jour (ce rythme s'est établi dés le 16 août).

Chronogramme



Annexe II.1. Chronogramme des travaux

Annexe III

Le moteur graphique *Graphics Booster*

1. De la nécessité de l'accélération

Pour mettre en évidence les raisons d'un outil d'accélération, on se contentera d'évoquer un résultat comparatif entre les performances de la routine du BIOS (fonction 0Ch de l'interruption 10h) et notre routine.

Le test en question concerne le remplissage de l'écran en mode 12h, c'est-à-dire l'écriture de 307200 points. Les résultats sont flagrants:

Interruption 10h : 5 secondes 49 centièmes.

Notre routine : 6 centièmes de secondes !

=> Une accélération de plus de 90 fois.

Même avec ces améliorations, l'affichage laisse à désirer. En effet, sur Windows par exemple, les routines du GDI atteignent quelques millions de pixels par seconde dans les mêmes conditions.

2. Technique utilisée

2.1. Le mode VGA 12h

Dans ce mode la mémoire vidéo commence au segment A000h, elle est organisée comme suit:

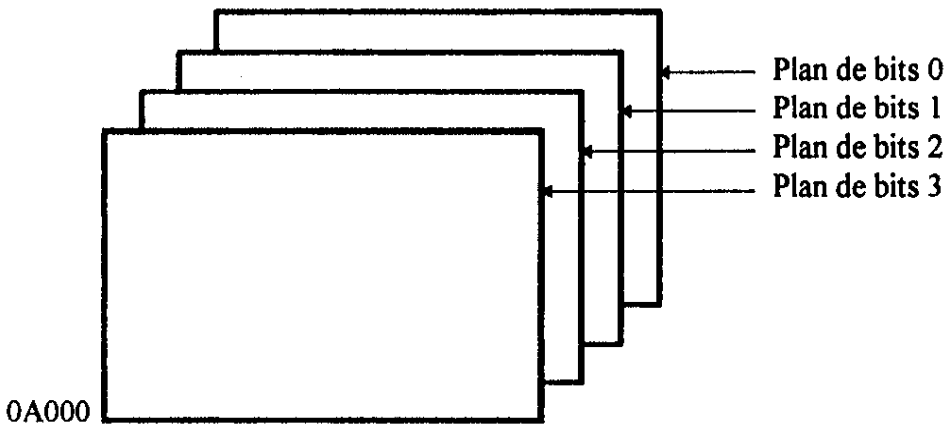


Fig. Annexe III.1. Organisation en plans de bits de la mémoire vidéo

Ces plans sont accessibles a partir de la même adresse (A000h) mais un seul est actif à la fois, on le choisi en adressant le port 03CEh du contrôleur graphique. Une couleur d'un pixel dans ce mode (affichant 16 couleurs) est codée sur 4 bits, chaque bit est inscrit dans le plan correspondant à son poids.

Par exemple, la couleur 5_{10} (0101_2) au point (x, y) sera représentée ainsi:
BitPos = *(640x + y)* MOD *8*
BytePos = *(640x + y)* DIV *8*

0 dans le bit *BitPos* de l'octet à l'offset *BytePos* du plan de bits 3
1 dans le bit *BitPos* de l'octet à l'offset *BytePos* du plan de bits 2
0 dans le bit *BitPos* de l'octet à l'offset *BytePos* du plan de bits 1
1 dans le bit *BitPos* de l'octet à l'offset *BytePos* du plan de bits 0

2.2. Mise en service des routines

Nous avons choisi d'installer le moteur graphique Graphics Booster en tant que gestionnaire de l'interruption 32h.

Le choix de cette interruption revient au fait qu'aucun programme système n'en fait usage, même les applications les plus courantes ne l'utilisent pas. En fait, quelques rares applications en font usage, mais il s'agit d'applications virus ! et plus précisément les virus Tiny, Plovdiv 1.3 et Damage 1.3.

Le programme Graphics Booster 1.5 s'installe en tant que TSR (*Terminate and Stay Resident*) et prend en charge la gestion de l'interruption 32h.
A chaque démarrage il s'autodétecte, en cas de détection d'une instance en service, l'installation est avortée.

3. Format d'appel

Fonction	Entrée	Sortie
Test de présence	AH : 12h	AX : 2100h
Affiche un pixel	AH : 00 AL : couleur (0..15) CX : abscisse DX : ordonnée	Un pixel est dessiné à la position et avec la couleur indiquées
Affiche un pixel en mode XOR put	AH : 01 AL : couleur (0..15) CX : abscisse DX : ordonnée	Un pixel est dessiné à la position indiquée avec la couleur: AL XOR couleur courante
Lire la couleur d'un point donné	AH : 02 CX : abscisse DX : ordonnée	AX : couleur du point graphique
Dessiner une barre pleine	AH : 03 AL : couleur (CX, DX) : coordonnées du coin supérieur gauche (SI, DI) : coordonnées du coin inférieur droit	Une barre est dessinée dans la position et avec la couleur choisies.

Table Annexe III.1. Format d'appel des services de Graphics Booster

A chaque appel il suffit de charger les registres avec les valeurs indiquées ci-dessus puis de déclencher l'interruption 32h.

4. Code

```
CODE segment
      ORG    100h
      assume CS:CODE

 start:  JMP   install

      CR         EQU  0Dh
      LF         EQU  0Ah

Welcome   db CR, LF
          db 'Graphics Booster 1.5'
          db CR, LF
          db '(c) Copyright 1996, E. Elachtar, I. Chihi,'
          db 'I. Romdhani', LF, CR
          db 'National Computer Science School','$'


AlreadyIn  db CR, LF, 'Graphics Booster is already
           db 'installed !', LF, CR, '$'

 InstSucc  db LF, CR, 'Successfuly installed', LF, CR, '$'

       X1  dw 0
       X2  dw 0
       Y1  dw 0
       Y2  dw 0
    Color  db 0

WinPalette db  0, 4, 2, 6, 1, 5, 3, 8, 7, 12, 10, 14, 9,
              13, 11, 15


TSRint32  proc far        ; .The new entry to INT 32

            JMP    CheckEntry

       PutPixelProc   Proc NEAR
                PUSH   CX
                PUSH   ES
                PUSH   DX

                MOV    AX, CX  ; load x coordinate
                MOV    DX, 80  ; line length in bytes, here 80
                MUL    DX
                POP    BX      ; BX <- y
                PUSH   BX
                               ; loads y coordinate
```

```
        MOV     CL, BL  ; saves low byte for shift

        SHR     BX, 1   ; divide x by 8
        SHR     BX, 1
        SHR     BX, 1
        ADD     BX, AX  ; adds the offset resulting
                        ; from the MUL

        AND     CL, 7   ; calculates the binary mask
                        ; from x
        XOR     CL, 7
        MOV     AH, 1
        SHL     AH, CL

        MOV     DX, 3CEh ; accessing video
                         ; controller
        MOV     AL, 8    ; loads binary mask into
                         ; mask register
        OUT     DX, AX

        MOV     AX, (02h shl 8) + 5  ; write mode 2 &
        OUT     DX, AX               ; read mode 0

        MOV     AX, 0A000h     ; loads video segment
        MOV     ES, AX

        MOV     AL, ES:[BX]  ; loads the latch register


        MOV     AL, Color       ; set the color
        MOV     ES:[BX], AL     ; rewrites the latch
                                ; register

            ; Resets the default values
            ; into the graphic controller registers

        MOV     AX, (0FFh shl 8 ) + 8
        OUT     DX, AX

        MOV     AX, (00h shl 8) + 5
        OUT     DX, AX

        POP     DX
        POP     ES
        POP     CX

        RET

PutPixelProc  EndP

GetPixelProc  Proc NEAR
        PUSH    DX
        PUSH    CX
```

```
                MOV     AX, DX      ; loads y coordinate
                MOV     DX, 80      ; Multiplies by line width
                MUL     DX
                MOV     SI, CX      ; loads X coordinate

                SHR     SI, 1       ; Divides X by eight (2^3)
                SHR     SI, 1
                SHR     SI, 1
                ADD     SI, AX      ; Adds offset from
                                    ; multiplication

                AND     CL, 7       ; Calculates the binary mask
                                    ; from X
                XOR     CL, 7
                MOV     CH, 1
                SHL     CH, CL

                MOV     AX, 0A000h  ; loads ES with the video
                                    ; segment
                MOV     ES, AX

                MOV     DX, 03CEh   ; Access the graphical
                                    ; controller
                MOV     AX, (3 shl 8)+ 4  ; reads plane #3
                XOR     BL, BL

GP1   :         OUT     DX, AX      ; Sets the plan to read
                MOV     BH, ES:[SI] ; reads the latch register
                AND     BH, CH   ; masks non interesting pixels
                NEG     BH
                ROL     BX, 1    ; rotates bit 7 of BH into bit
                                 ; 1 of BL

                DEC     AH     ; continues on the next plane
                JGE     GP1    ; is it zero, if no ---> carry on

                MOV     AL, BL  ; sends back the result into AL
                MOV     AH, 00  ; makes sure the whole AX
                                ; contains the color


                POP     CX
                POP     DX
                RET

    GetPixelProc   EndP


CheckEntry: PUSHF       ; saves flags to keep them
                        ; away from CMP effects
            MOV     color, AL
            PUSH    SI
            MOV     BH, 0
            MOV     BL, AL
            MOV     SI, BX
```

```
                MOV     AL, WinPalette[SI]
                POP     SI


                CMP     AH, 12h      ; is this a presence test ?
                JE      Pres_Test

                CMP     AH, 00h ; may be it is a Put Pixel request?
                JE      Put_Pixel

                CMP     AH, 01h
                JE      xorPut_Pixel

                CMP     AH, 02h
                JE      Get_Pixel    ; or a Get Pixel request

                CMP     AH, 03h      ; or even a Draw Box request !
                JE      DrawBox

                POPF         ; but what the hell could it be else ?!
                RET     2


Pres_Test :     POPF
                STI
                MOV     AX, 2100h       ; Send back gboost signature
                RET     2               ; done.

Put_Pixel :     POPF
                MOV     Color, AL
                CALL    PutPixelProc
                RET     2


Get_Pixel :     POPF
                XCHG    CX, DX
                CALL    GetPixelProc


                PUSH    SI
                MOV     BH, 0                   ; converts the requested
                MOV     BL, AL                  ; color into Windows
                MOV     SI, BX                  ; palette
                MOV     AL, WinPalette[SI]
                POP     SI

                RET     2

xorPut_Pixel:
                POPF
                XCHG    CX, DX
                CALL    GetPixelProc  ; gets the current pixel
```

```
                                    ; color
                XCHG    CX, DX

                MOV     BL, AL
                XOR     Color, BL   ; XORs it with the requested
                                    ; color

                CALL    PutPixelProc  ; draws the pixel with the
                                      ; resulting color
                RET     2

DrawBox :       MOV     X2, SI
                MOV     Y2, DI
                MOV     Color, AL
                MOV     X1, DX
                MOV     Y1, CX

Continue :      PUSH    CX
                PUSH    ES
                PUSH    DX

                MOV     AX, CX        ; load x coordinate
                MOV     DX, 80        ; line length in bytes, here 80
                MUL     DX
                POP     BX            ; BX <- y
                PUSH    BX
                                      ; loads y coordinate
                MOV     CL, BL        ; saves low byte for shift

                SHR     BX, 1         ; divide x by 8
                SHR     BX, 1
                SHR     BX, 1
                ADD     BX, AX        ; adds the offset resulting
                                      ; from the MUL

                AND     CL, 7         ; calculates the binary mask
                                      ; from x
                XOR     CL, 7
                MOV     AH, 1
                SHL     AH, CL

                MOV     DX, 3CEh        ; accessing video controller
                MOV     AL, 8           ; loads binary mask into
                                        ; masking register
                OUT     DX, AX

                MOV     AX, (02h shl 8) + 5  ; write mode 2 &
                OUT     DX, AX               ; read mode 0

                MOV     AX, 0A000h           ; loads video segment
                MOV     ES, AX

                MOV     AL, ES:[BX]  ; loads the latch register
```

```
                MOV     AL, Color       ; set the color
                MOV     ES:[BX], AL     ; rewrites the latch register


                 ; Resets the default values
                 ; into the graphic controller registers

                MOV     AX, (0FFh shl 8 ) + 8
                OUT     DX, AX

                MOV     AX, (00h shl 8) + 5
                OUT     DX, AX

                POP     DX
                POP     ES
                POP     CX

                CMP     DX, X2          ; is this the end of line
                JE      NextLine        ; if yes, go to next line
                INC     DX              ; if not process the next pixel
                JMP     Continue        ; on the same line
NextLine :      MOV     DX, X1          ; restart from line beginning
                CMP     CX, Y2          ; is this the last line
                JE      EndBox          ; if yes, done
                INC     CX              ; if not go on the next line
                JMP     Continue

EndBox :        POPF

                RET     2

      TSRint32  endp


 EndOfTSRCode:          ; this marks where the resident code ends

Install :       MOV     AX, CS
                MOV     DS, AX

                MOV     DX, offset Welcome  ; .Welcoming message
                MOV     AH, 09h
                INT     21h


                MOV     AH, 12h             ; .is it installed ?
                INT     32h
                CMP     AH, 21h
                JE      AlreadyInstErr

                MOV     AX, 2532h  ; Revector INT 32 to our gBoost
                MOV     DX, offset TSRint32
                INT     21h
```

```
OkGoAhead : MOV     DX, offset InstSucc    ; Install Ok message
            MOV     AH, 09h
            INT     21h

            MOV     DX, EndOfTSRCode-Start+100h+15; Calculation
                                                  ;   of the
                                                  ; bytes number
            MOV     Cl, 4
            SHR     DX, CL          ; Converts into paragraphs

            MOV     AL, 0           ; Exit code, 0 => No errors.
            MOV     AH, 31h         ; Leave it resident  fn. 31h
            INT     21h


AlreadyInstErr:
            MOV     DX, offset AlreadyIn ; Already installed
                                         ; message
            MOV     AH, 09h
            INT     21h

            MOV     AH, 4Ch
            MOV     AL, 0FFh ; Exit code, -1 => errors occured
            INT     21h

code ENDS
 end Start
```

Annexe IV

Le format BMP

1. Compression des images

Plusieurs algorithmes sont utilisés pour compresser les images.

1.1. Le format GIF

Il est apparut sur le réseau *CompuServe* pour le transfert des images, il utilise un algorithme LZW (Lempel-Ziff-Welsh) qui repose sur le principe de remplacer les occurrences déjà rencontrées par des indexes dans un dictionnaire.

1.2. Le format JPEG

C'est le format *Join Photographic Experts Group*, il utilise un algorithme avec perte d'information. C'est le principe d'éliminer les détails imperceptibles par l'oeil humain, ceci permet d'atteindre des taux de compression très élevés.

1.3. Le format TIFF

C'est le Tagged Image File Format. Il fût introduit par la technologie PAO pour compresser les images scannées.

1.4. Autres formats

Il y a plusieurs autres formats tels que TIGA, LBM, PCX, PIC ..etc

2. Le format de Windows : BMP

Microsoft Windows enregistre les images dans un format indépendant du matériel, c'est le DIB (*Device-Independent BitMap*). Cette indépendance est garantie par le fait que ce format d'image définit les couleurs qu'il utilise d'une manière absolue par indication des composantes RGB (Red Green Blue) pour chacune.

2.1. Structure d'un fichier BMP

Un fichier image BMP est structuré comme suit:

```
type BMPFile is record
 bmfh        : BITMAPFILEHEADER;
 bmih        : BITMAPINFOHEADER;
 aColors     : array(long_integer range <>) of RGBQUAD;
 aBitMapBits:array(long_integer range <>) of byte_integer;
end record;
```

Les types BITMAPINFOHEADER, BITMAPFILEHEADER et RGBQUAD sont définis comme suit:

```
type BITMAPFILEHEADER is record

        -- Specifies the file type, should be "BM"
        bfType      :integer;

        -- Specifies the size of the file, in bytes.
        bfSize      : long_integer;

        -- Reserved; must be set to zero.
        bfReserved1 : integer;

        -- Reserved; must be set to zero.
        bfReserved2 : integer;

        -- Specifies the byte offset from
        -- the BITMAPFILEHEADER structure to the
        -- actual bitmap data in the file.
        bfOffBits   : long_integer;

end record;
```

La structure BITMAPINFOHEADER contient des informations sur les dimensions et le format de couleurs du DIB.

```
type BITMAPINFOHEADER is record

    -- Specifies the number of bytes required
    -- by the BITMAPINFOHEADER structure.
    biSize,
    biWidth,
    biHeight          : long_integer;

    -- Specifies the number of planes for the
    -- target device. This member must be set to 1.
    biPlanes,

    -- Specifies the number of bits per pixel.
    -- This value must be 1, 4, 8, or 24.
    biBitCount        : integer ;
```

```
    -- Specifies the type of compression for a compressed
    -- bitmap. It can be one of the following values:
    biCompression,
    biSizeImage,
    biXPelsPerMeter,
    biYPelsPerMeter,
    biClrUsed,
    biClrImportant    : long_integer;

end record;
```

Une table de définition des couleurs contient des entrées de la forme:

```
type RGBQUAD is record

    -- intensity of the blue component
    rgbBlue     : byte_integer;

    -- intensity of the green component
    rgbGreen    : byte_integer;

    -- intensity of the red component
    rgbRed      : byte_integer;

    rgbReserved : byte_integer;
end record;
```

Si vous désirer écrire des routines de manipulation de ce format, voici des conseils utiles:

- Une ligne est alignée sur un mot(*), alignez là en ajoutant des points noirs s'il le faut.
- Les données décrivant les points de l'image commencent à l'offset 118 dans le fichier, ils décrivent l'image en commençant par le coin inférieur gauche.
- Pratiquement toutes les images utilisées par Windows ne sont pas compressées. En mode 16 couleurs, chaque octet décrit deux pixels consécutifs dans ses 4 bits de poids faible et fort.

(*) *Aligned on a word boundary* dans le document original.

Bibliographie