Chapitre 4

Gestion de la mémoire

Le **gestionnaire de mémoire** doit tenir à jour la liste des parties libres et des parties occupées de la mémoire, allouer de la mémoire aux processus qui en ont besoin et récupérer la mémoire utilisée par un processus lorsque celui-ci se termine.

4.1 Étude générale de la gestion de la mémoire

4.1.1 Modèles de programmation

On peut distinguer trois modèles de programmation en ce qui concerne la gestion de la mémoire.

Monoprogrammation pure.- La gestion de la mémoire la plus simple consiste à avoir un seul processus en mémoire à un instant donné et à lui permettre d'utiliser toute la mémoire disponible. Cette approche, courante avant 1960, n'est plus utilisée de nos jours parce qu'elle implique, entre autres, que chaque processus contienne les pilotes des périphériques d'entrées-sorties qu'il utilise.

Partage avec le système d'exploitation.- La technique utilisée en général sur les premiers micro-ordinateurs était la suivante : la mémoire est partagée entre le système d'exploitation et un unique processus utilisateur.

Lorsque la mémoire est organisée de cette manière, il ne peut exister qu'un seul processus s'exécutant à un instant donné. L'utilisateur tape une commande et le système d'exploitation charge le programme demandé en mémoire, puis l'exécute. Lorsque le processus se termine, le système d'exploitation affiche une **invite de commande** (en anglais *prompt*) et attend que l'utilisateur lui communique la commande suivante pour charger un nouveau processus, remplaçant le précédent.

Multiprogrammation.- Il est souvent utile d'avoir plus d'un processus en mémoire à la fois. Mais il faut alors organiser la mémoire le plus efficacement possible.

4.1.2 Modèles d'organisation de la mémoire

Dans le cas de la multiprogrammation, on peut distinguer deux modèles d'organisation de la mémoire.

4.1.2.1 Multiprogrammation avec partitions fixes

Introduction.- La méthode la plus simple pour organiser la mémoire, dans le cas de multiprogrammation, est de la diviser en n partitions (éventuellement de tailles inégales), avec n fixé pour toute la session en cours : il faut redémarrer l'ordinateur pour le modifier. Ce partitionnement peut, par exemple, être effectué par l'opérateur au démarrage du système.

Chaque nouvelle tâche est placée dans la file d'attente de la plus petite partition qui peut la contenir. Tout espace inutilisé dans une partition est dès lors perdu.

Ce système, à partitions fixes déterminées par l'opérateur au démarrage, a été utilisé par le OS/360 des grands ordinateurs IBM pendant de nombreuses années. Il s'appelait **MFT** (pour *Multiprogramming with a Fixed number of Tasks*) ou **OS/MFT**.

Code translatable.- La multiprogrammation soulève le problème de la **translation des adresses**. En effet les différentes tâches sont exécutées à diverses adresses. Au moment de l'édition des liens (c'est-à-dire lorsque le programme principal, les procédures de l'utilisateur et les procédures de bibliothèque sont réunies en un seul espace d'adressage), l'éditeur des liens doit connaître l'adresse du début du programme.

Supposons, par exemple, que la première instruction soit un appel à une procédure située à l'adresse (relative) 100 à partir du début du fichier binaire produit par l'éditeur des liens. Si le programme est chargé dans la partition commençant à l'adresse 100 K, cette instruction provoque un saut à l'adresse absolue 100 qui se trouve à l'intérieur du système d'exploitation. En effet, l'adresse devrait être 100 K + 100. Ce problème est celui de la **translation des adresses**.

Une solution consiste à modifier les instructions du programme chargé en mémoire. On ajoute, par exemple, 100 K aux adresses des programmes placés dans la partition 1. Pour effectuer cette opération (à savoir translater les programmes au moment du chargement), l'éditeur des liens doit inclure dans le code binaire la liste des mots mémoire étant des adresses, pour les distinguer des codes d'opérations, des constantes et autres éléments qui ne doivent pas être translatés. OS/MFT fonctionnait de cette manière.

Une autre solution au problème de la translation d'adresse est de doter la machine de deux registres matériels spéciaux : le **registre de base** et le **registre de limite**. Quand on lance un processus, on charge dans le registre de base la limite inférieure de la partition qui lui est attribuée et dans le registre de limite la longueur de la partition. On ajoute la valeur du registre de base à chaque adresse mémoire générée. Ainsi, si le registre de base contient la valeur 100 K, l'instruction `call 100` est remplacée par `call 100 K + 100` sans modifier l'instruction elle-même. L'IBM PC utilise une version simplifiée de ce mécanisme : il possède un registre de base, mais il n'a pas de registre de limite.

Il existe un autre avantage à l'utilisation du registre de base : les programmes peuvent être déplacés en mémoire après le début de leur exécution ; il suffit pour cela de changer la valeur du registre.

4.1.2.2 Multiprogrammation avec partitions variables

Introduction.- On peut aussi choisir des partitions avec tailles variables. Dans ce cas il faut absolument gérer la mémoire.

Gestion de la mémoire par tables de bits.- La mémoire est divisée en **unités d'allocation** (en anglais *cluster*), dont la taille peut varier de quelques mots à plusieurs kilo-octets. On fait correspondre, à chaque unité d'allocation, un bit dans la **table de bits**, égal à 0 si l'unité est libre et à 1 si elle est occupée.

La taille de l'unité d'allocation joue un rôle important. Plus elle est faible, plus la table de bits est importante. Si on prend une unité d'allocation de grande taille, on réduit la taille de la table de bits, mais on perd beaucoup de place mémoire chaque fois que la taille d'un programme n'est pas un multiple de la taille d'une unité d'allocation. Il faut donc trouver un compromis.

La table de bits permet de mémoriser l'occupation de la mémoire dans un espace mémoire de taille fixe car la taille d'une table de bits ne dépend que de la taille de la mémoire principale et de celle de l'unité d'allocation. Le seul vrai problème survient lorsqu'on doit chercher un emplacement de taille k unités d'allocation. Le gestionnaire de la table doit alors parcourir la table de bits à la recherche de k zéros consécutifs. Cette recherche est lente, ce qui fait qu'en pratique on utilise rarement les tables de bits.

Gestion de la mémoire par liste chaînée.- Une deuxième méthode, pour tenir à jour l'occupation de la mémoire, consiste à gérer la liste chaînée des segments libres et des segments occupés. Dans ce cas un **segment** est un processus ou un espace libre entre deux processus.

Un processus a normalement deux voisins (sauf s'il se trouve en haut ou en bas de la mémoire). Ces derniers sont soit des processus, soit des zones libres, ce qui donne les quatre combinaisons de la figure ci-dessous :

PPP
PPV
VPP
VPV

où P est un processus et V un segment vide. Voyons ce qui arrive lorsque le processus se termine :

- Dans le premier cas, il suffit de remplacer le processus par une zone vide pour obtenir :

  PVP.

- Dans le deuxième cas et dans le troisième cas on obtient une zone vide plus importante :

  PVV
  VVP,

  ce qui réduit la liste d'une entrée.
- Dans les dernier cas, on obtient une seule zone vide :

  VVV,

  ce qui réduit la liste de deux entrées.

Comme la table des processus contient des pointeurs sur les positions des différents processus dans cette liste chaînée, il est plus commode d'avoir un double chaînage plutôt qu'une liste chaînée simple. Ceci permet de voir si l'entrée précédente est une zone libre, dans le but de réaliser une fusion.

Quand on mémorise les processus et les zones libres dans une liste triée en fonction des adresses, on peut utiliser plusieurs algorithmes pour allouer de la mémoire aux nouveaux processus. On suppose que le gestionnaire de la mémoire connaît la taille de la mémoire à allouer.

L'algorithme le plus simple est l'**algorithme de la première zone libre** (en anglais *first fit*). Le gestionnaire de la mémoire parcourt la liste des segments à la recherche de la première zone libre pouvant contenir le processus. Cette zone est alors scindée en deux parties : la première contient le processus et la deuxième l'espace mémoire inutilisée (sauf si le processus a exactement la même taille que la zone). Cet algorithme est rapide puisqu'il y a très peu de recherche.

Un autre algorithme est celui du **meilleur ajustement** (en anglais *best fit*). On recherche dans toute la liste la plus petite zone libre qui convient. On évite ainsi de fractionner une grande zone dont on pourrait avoir besoin ultérieurement. L'algorithme du meilleur ajustement est plus lent que celui de la première zone puisqu'il doit, à chaque appel, parcourir l'intégralité de la liste chaînée.

4.2 Mise en place dans MS-DOS

4.2.1 Organisation de la mémoire adressable

Nous avons besoin de connaître l'organisation de la mémoire adressable décidée par IBM pour pouvoir interpréter correctement certains emplacements.

4.2.1.1 Organisation générale de la mémoire en trois secteurs

Le microprocesseur `i8086`, avec ses vingt broches, ne pouvant adresser qu'un MiO, soit 1 024 kiO, d'adresses comprises entre `00000h` et `FFFFFh`. IBM a partagé cet espace adressable en trois parties :

— les 640 kiO compris entre `00000h` et `9FFFFh` sont réservés à la *mémoire vive proprement dite*, située sur des circuits intégrés de mémoire de type physique RAM (pour *Random Access Memory*, il s'agit surtout ici d'une technologie) ; tous les circuits intégrés possibles ne sont pas nécessairement présents (les premiers PC n'étaient livrés qu'avec 64 kiO de mémoire vive).

— les 128 KiO suivants, compris entre `A0000h` et `BFFFFh`, sont réservés à la *mémoire graphique*, contenant ce qui est affiché à l'écran ; on utilise souvent pour cette fonction des circuits intégrés de mémoire plus rapides afin de faciliter le rafraîchissemnt de l'écran.

— les 256 KiO restants, compris entre `C0000h` et `FFFFFh`, sont réservés au BIOS, situé sur des circuits intégrés de mémoire non volatile de type ROM (pour *Read Only Memory*).

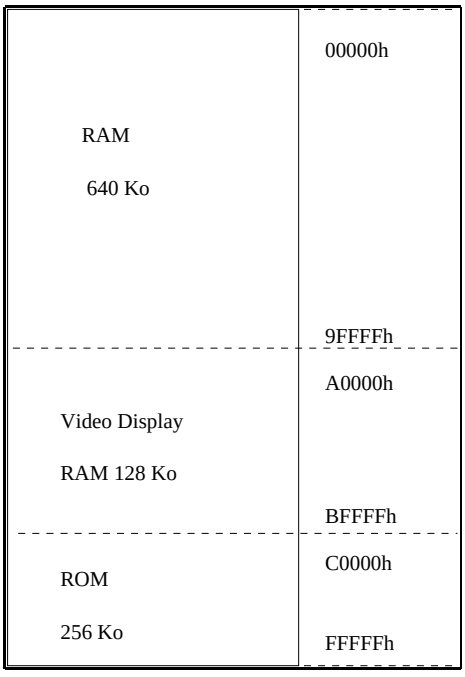

FIGURE 4.1 – Organisation de la mémoire de l'IBM PC

4.2.1.2 Organisation des 640 kiO de RAM de mémoire classique

Définition.- Comme nous venons de le dire, les 640 kiO d'adresses comprises entre `00000h` à `9FFFFh` sont situés sur des circuits intégrés de mémoire RAM. On parle de **mémoire classique** (en anglais *conventional memory* d'où quelquefois la mauvaise traduction *mémoire conventionnelle* (*sic*)).

Constitution de la mémoire classique.- Sur les premiers PC, seuls 64 kiO à 256 kiO étaient situés sur la carte mère, le reste étant éventuellement obtenu en ajoutant une carte d'extension (mémoire). Dans ce cas il fallait positionner des interrupteurs DIP pour indiquer au BIOS et au DOS que de la mémoire avait été ajoutée.

Les adresses doivent nécessairement être consécutives et donc, lorsqu'on ajoute de la mémoire, on doit nécessairement placer un circuit intégré pour les adresses les moins élevées non encore pourvues.

Place réservée au système d'exploitation.- La mémoire classique n'est pas entièrement disponible pour les programmes et les données de l'utilisateur. Une partie en est réservée au système d'exploitation :

- Le premier kiO (`00000h` à `003FFh`) est réservé, par *Intel*, à la *table des vecteurs d'interruption*.
- Les 256 octets suivants, de `00400h` à `004FFh`, sont réservés, par les concepteurs de l'IBM-PC, à la *zone de communication* du BIOS.
- Les 256 octets suivants, d'adresses `00500h` à `005FFh`, sont réservés au stockage de certains *paramètres temporaires* relatifs à MS-DOS et au langage BASIC.
- Enfin un certaine place mémoire est occupée par le système d'exploitation MS-DOS lui-même, la quantité dépendant de la version de celui-ci.

4.2.1.3 La mémoire ROM

Les 256 kiO d'adresses allant de `C0000h` à `FFFFFh` ont été conçu au départ pour être de la mémoire ROM, répartie sur plusieurs barrettes. Il s'agit de quatre barrettes allouant la mémoire conformément au schéma suivant :

Emplacement mémoire	Description
`C000:0000 - C000:7FFF`	extension
`C000:8000 - C000:CFFF`	contrôle du disque dur
`D000:0000 - E000:FFFF`	extension (XT)
`F000:0000 - F000:FFFF`	BIOS ROM

On s'aperçoit que deux emplacements ne sont pas utilisés par le BIOS. On parle de **mémoire haute** (ou UMB pour l'anglais *Upper Memory Block*). Dans certaines versions de PC, on y a placé des circuits intégrés de mémoire RAM et on a utilisé ces emplacements pour la mémoire vive.

4.2.2 Partage de la mémoire

4.2.2.1 Les zones-champs

La mémoire est partagée en blocs contigus appelés **zones-champs** (en anglais *arenas*). Chaque zone débute à une frontière de paragraphe (de 16 octets) et contient un nombre entier de paragraphes.

Chaque zone-champ commence par un **en-tête** (en anglais *arena header* ou *memory control record*) de 16 octets.

4.2.2.2 En-tête d'une zone-champ

L'en-tête d'une zone-champ contient les champs suivants :

— `00-00h` : Le **code-clef**, où `4Dh` ('M' pour *More*) signifie que d'autres blocs suivent et `5Ah` ('Z' pour *Zero*) que zéro bloc suit, c'est-à-dire qu'il s'agit du dernier bloc.

— `01-02h` : Adresse de segment du PSP propriétaire. La valeur `0800h` signifie que le segment appartient à `MSDOS.SYS` et `0000h` que la zone-champ est disponible.

— `03-04h` : Longueur du bloc de mémoire, en paragraphes.

— `05-07h` : Réservé.

— `08-0Fh` : Nom du fichier propriétaire, en format ASCIIZ depuis DOS 4.0 : nom du fichier binaire exécutable propriétaire de cette zone-champ.

4.2.2.3 Liste chaînée des blocs

Techniquement, les zones-champs ne forment pas une liste chaînée car les en-têtes contiennent des tailles plutôt que des pointeurs mais l'effet global reste le même :

— Le *premier* bloc mémoire appartient à `MSDOS.SYS`. Il contient les tampons de fichiers MS-DOS, les FCB utilisés par les fonctions de numéro de fichiers et les pilotes de périphériques chargés par les commandes `device` de `config.sys`.

— Le *second* bloc mémoire est la partie résidente de `command.com` avec son PSP.

— Le *troisième* bloc mémoire est l'environnement maître, contenant la valeur de `comspec`, la valeur de `prompt`, les valeur de `path` et les mots engendrés par `set`.

— Les blocs suivants comprennent les programmes TSR et le programme en train de s'exécuter. Chacun de ces programmes possède deux blocs : le premier est une copie de l'environnement et le second le segment de programme avec son PSP et son module exécutable.

Exercice.- Vérifiez-le en parcourant la mémoire avec `debug`.

4.2.2.4 Adresse du premier bloc : fonction 52h de l'interruption 21h

Introduction.- Il est évidemment très important d'obtenir l'adresse du premier bloc mémoire. Celui-ci dépend de la version de MS-DOS.

Fonction adéquate.- On se sert, pour obtenir cette adresse, de la fonction (non documentée) `52h` de l'interruption `21h`. Cette fonction renvoie dans le registre `ES` l'adresse de segment de la liste des tables de fichier et le décalage dans le registre `BX`. Alors `ES:[BX-4]` pointe sur un double mot de la forme `IP:CS` qui contient l'adresse du premier en-tête de bloc.

Exemple.- Utilisons `debug` pour obtenir ces données :

```
c:> debug
-a
0CC2:0100 mov ah,52
0CC2:0102 int 21
0CC2:0104 int 3
0CC2:0105
-g

AX=5200  BX=0026  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0CC2  ES=00CA  SS=0CC2  CS=0CC2  IP=0104   NV UP EI PL NZ NA PO NC
0CC2:0104 CC            INT     3
-
```

On a ici `ES = 00CA` et `BX = 0026`. L'adresse du premier bloc est donc localisée à `00CA:0022`. Continuons donc :

```
-d 00CA:0022
00CA:0020       00 00 0A 02 46 13-CA 00 CC 00 CA 00 4C 00     ....F.......L.
```

Le double mot est donc `00 00 0A 02` et l'adresse est `020A:0000`.

4.2.2.5 Obtention de la liste chaînée des blocs

Méthode.- Nous venons de voir comment obtenir l'adresse du premier bloc. On a la longueur de ce premier bloc grâce aux décalages `03-04h` de l'en-tête de ce bloc. Pour obtenir l'adresse du bloc suivant, on utilise l'adresse du premier bloc, on lui ajoute 1 (puisque l'en-tête occupe un paragraphe) ainsi que la longueur du bloc (en paragraphes), obtenant ainsi l'adresse du second bloc. On recommence de la même façon pour les blocs suivants.

Exemple.- Continuons l'exemple précédent. Nous avons vu que l'adresse du premier bloc est `020A:0`. Regardons ce qui est contenu dans le premier en-tête :

```
-d 020A:0
020A:0000  4D 08 00 4C 02 02 2E FE-53 44 00 2E 81 06 B8 07   M..L....SD......
```

Le premier octet `4D` ('M') nous dit qu'il ne s'agit pas du dernier bloc. Le mot suivant `08 00`, soit `8000h`, nous dit que ce bloc-mémoire appartient à `MSDOS.SYS`. Le mot suivant `4C 02`, soit `024Ch`, nous donne la longueur du bloc mémoire.

Le bloc mémoire suivant, correspondant à `command.com`, se trouve à l'adresse 20A[0] + 1[0] + 24C[0], soit 457[0]. Regardons ce qui est contenu dans le deuxième en-tête :

```
-d 457:0
0457:0000  4D 08 00 04 00 C2 C2 00-53 43 C2 C2 C2 C2 C2 C2   M.......SC......
```

Le premier octet `4D` ('M') nous dit qu'il ne s'agit pas du dernier bloc. Le mot suivant `08 00`, soit `8000h`, nous dit que ce bloc-mémoire appartient aussi à `MSDOS.SYS`. Le mot suivant `04 00`, soit `0004h`, nous donne la longueur du bloc mémoire.

Le bloc mémoire suivant se trouve à l'adresse 457[0] + 1[0] + 4[0], soit 45C[0]. Regardons ce qui est contenu dans le troisième en-tête :

```
-d 45C:0
045C:0000  4D 60 04 02 00 33 C0 26-8B 0E 34 15 D1 E9 74 06   M‘...3.&..4...t.
```

Il ne s'agit toujours pas du dernier bloc. Le mot `60 04`, soit `0460h`, nous dit à quel processus appartient ce bloc-mémoire. Le mot `02 00` donne la longueur `0002h` du bloc mémoire.

4.2.3 Fonctions 48h, 49h et 4Ah d'allocation dynamique de la mémoire

Ces fonctions de l'interruption `int 21h` de MS-DOS permettent d'allouer, de libérer et de modifier la taille d'une partie de la mémoire.

4.2.3.1 Fonction 48h d'allocation de mémoire

Fonction 48h.- Pour allouer de la mémoire à un programme, on fait appel à la fonction `48h` de `int 21h`, après avoir placer le nombre de paragraphes désirés dans le registre `BX` :

```
mov ah, 48h             ; requete d'allocation de memoire
mov bx, paragraphes     ; nombre de paragraphes
int 21h                 ; appel du service d'interruption
```

Sémantique.- L'opération commence au premier bloc mémoire et parcourt chaque bloc jusqu'à ce qu'il trouve un espace assez grand pour la requête, en général à la fin de la mémoire.

Une opération réussie positionne l'indicateur de retenue `CF` à zéro et renvoie dans le registre `AX` l'adresse de segment du bloc de mémoire alloué. En cas d'échec l'indicateur de retenue `CF` est positionné à 1, un code d'erreur est envoyé dans le registre `AX` (`07h` pour bloc de mémoire détruit, `08h` pour mémoire insuffisante) et la taille, en paragraphes, du plus grand bloc mémoire disponible est envoyé dans le registre `BX`.

Un **bloc de mémoire détruit** est un bloc dont le premier octet n'est ni 'M' ni 'Z'.

4.2.3.2 Fonction 49h de libération de mémoire

Fonction 49h.- La fonction `49h` de `int 21h` libère de la mémoire allouée. On fait appel à elle après avoir placé l'adresse de segment du bloc à libérer dans le registre `ES` :

```
mov ah, 49h               ; requete de liberation de memoire allouee
lea es, seg-address       ; segment d'adresse du bloc
int 21h                   ; appel du service d'interruption
```

Sémantique.- Une opération réussie positionne à zéro l'indicateur de retenue `CF` et place `00h` dans le second et le troisième octet du bloc mémoire en question, signifiant qu'on ne les utilise plus. En cas d'échec, l'indicateur de retenue `CF` est positionné à 1 et un code d'erreur est envoyé dans le registre `AX` (`07h` pour bloc de mémoire détruit et `08h` pour adresse de bloc de mémoire non valide).

4.2.3.3 Fonction 4Ah de modification de taille de mémoire

Fonction 4Ah.- La fonction `4Ah` de `int 21h` permet d'accroître ou de décroître la taille d'un bloc mémoire. On commence par placer dans le registre `BX` le nombre de paragraphes souhaité et dans le registre `ES` l'adresse du PSP :

```
mov ah, 4Ah               ; requete de modification de la memoire
mov bx, paragraphes       ; nombre de paragraphes
lea es, PSP-address       ; adresse du PSP
int 21h                   ; appel du service d'interruption
```

Sémantique.- Une opération réussie positionne à zéro l'indicateur de retenue `CF`. En cas d'échec, l'indicateur de retenue `CF` est positionné à 1, un code d'erreur se trouve dans le registre `AX` (`07h` pour bloc de mémoire détruit, `08h` pour mémoire insuffisante, `09h` pour adresse de bloc de mémoire non valide) et dans le registre `BX` la taille maximum possible (s'il s'agissait d'un essai d'augmentation de la taille). Une fausse adresse du PSP dans `ES` peut causer le code d'erreur `07h`.