Chapitre 8

La mémoire

8.0.4 les mémoires

La mémoire d'un ordinateur se décompose en plusieurs éléments, dont le prix et le temps d'accès sont très variables, cf figure 8.1. Nous développerons dans ce chapitre et le suivant les questions et solutions relatives à la mémoire centrale.

L'importance de la gestion de la mémoire centrale vient de son coût et du coût relatif des autres formes de stockage, la figure 8.2 donne une idée des caractéristiques relatives des différents types de stockage.

8.0.5 La mémoire centrale

La mémoire est un tableau à une dimension de mots machines (ou d'octets), chacun ayant une adresse propre. Les échanges avec l'extérieur se font en général par des lectures ou des écritures à des adresses spécifiques.

Le système Unix est multi-tâche,ceci pour maximiser l'utilisation du cpu. Cette technique pose comme condition obligatoire que la mémoire centrale soit utilisée et/ou partagée entre les différentes tâches.

Les solutions de gestion de la mémoire sont très dépendantes du matériel et ont mis longtemps à évoluer vers les solutions actuelles. Nous allons voir plusieurs approches qui peuvent servir dans des situations particulières .

La mémoire est le point central dans un système d'exploitation, c'est à travers elle que l'unité centrale communique avec l'extérieur.

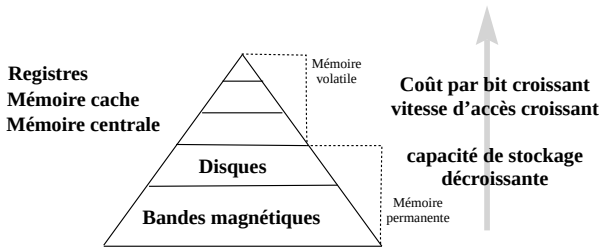

FIG. 8.1 – Hiérarchie de mémoires

CARACTERISTIQUES DES TYPES DE MEMOIRES

TYPE DE MEMOIRE	TAILLE (Octets)	TEMPS D'ACCES (secondes)	COUT RELATIF PAR BIT
CACHE	10^3-10^4	10^{-8}	10
MEMOIRE CENTRALE	10^6-10^7	10^{-7}	1
DISQUE	10^8-10^9	10^{-3}-10^{-2}	10^{-2}-10^{-3}
BANDE	10^8-10^9	10-10^2	10^{-4}

FIG. 8.2 – Caractéristiques relatives des mémoires.

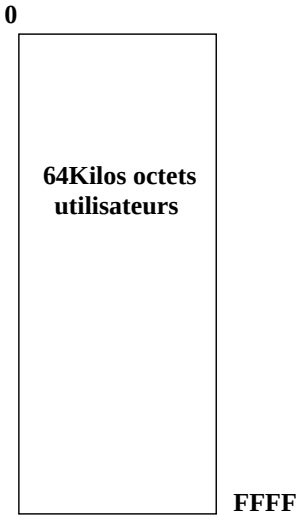

FIG. 8.3 – Une mémoire de 64 Kilo Octets.

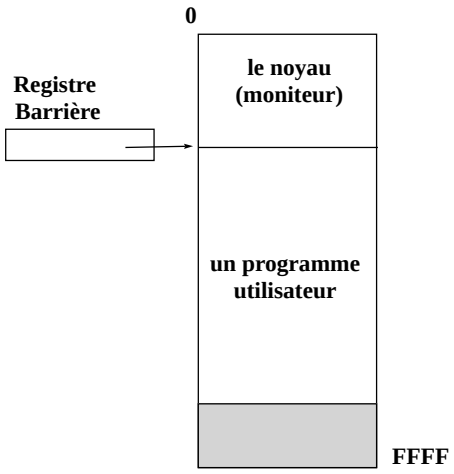

FIG. 8.4 – Protection du moniteur par un registre barrière.

8.1 Allocation contiguë

8.1.1 Pas de gestion de la mémoire

Pas de gestion de la mémoire ! Cette méthode, qui a l'avantage de la simplicité et de la rapidité, permet toute liberté quand à l'utilisation de la mémoire. En effet, toute adresse est accessible, et peut être utilisée pour n'importe quelle tâche. Le désavantage : aucune fonctionnalité, tout doit être reprogrammé, typiquement il n'y pas de système d'exploitation !

8.1.2 Le moniteur résidant

On cherche à protéger le noyau des interférences possibles de la part des utilisateurs. Pour cela, toute adresse d'instruction ou de donnée manipulée par un programme utilisateur est comparée à un registre barrière (fence register).

Tant que l'adresse est supérieure à la barrière, l'adresse est légale, sinon l'adresse est une référence illégale au moniteur et une interruption est émise (invalid adress).

Cette méthode demande que pour tout accès à la mémoire une vérification de la validité de l'adresse soit réalisée. Ceci ralentit toute exécution d'un accès mémoire. (Paterson donne comme exemple de ralentissement des temps de 980 nanosecondes sans vérification et 995 nanosecondes avec vérification). Globalement ce temps supplémentaire peut être oublié.

8.1.3 Le registre barrière

L'implémentation d'un tel mécanisme doit être réalisée de façon matérielle.

La valeur du registre barrière est parfois réalisée de façon fixe sur une machine, ce qui pose des problèmes dès que l'on veut changer le noyau et/ou protéger plus de mémoire (voir DOS).

8.1.4 Le registre base

Le mécanisme suivant est une notion plus utile et plus ergonomique pour décrire la zone d'adressage d'un programme, et utile pour résoudre le problème de déplacement des programmes en mémoire (relocation).

En effet, du fait que l'on utilise un registre barrière, les adresses utilisables de la mémoire ne commencent plus à 0000, alors que l'utilisateur veut continuer à utiliser des adresses logiques qui commencent à 0000.

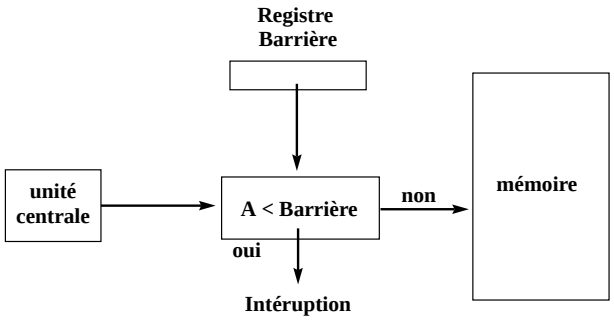

FIG. 8.5 – Implémentation du registre Barrière.

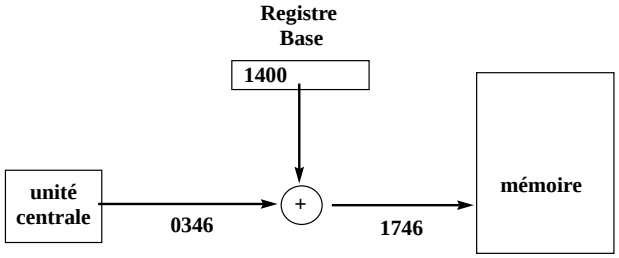

FIG. 8.6 – Implémentation du registre de Base.

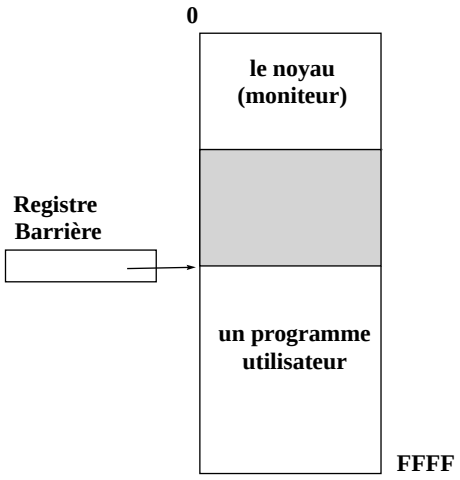

FIG. 8.7 – Positionnement d'un processus par un registre de Base.

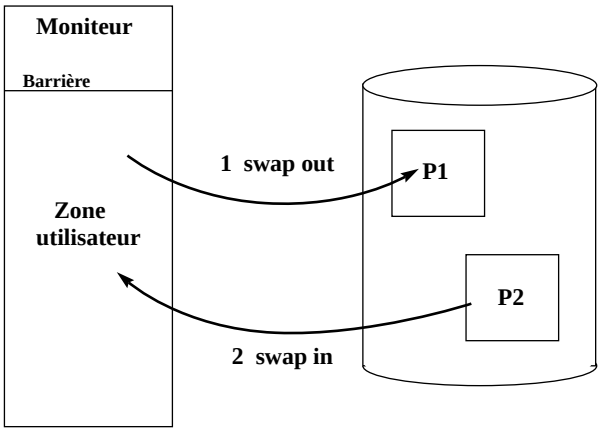

FIG. 8.8 – Un système de swap utilisant uniquement un registre barrière.

Pour continuer à fournir cette possibilité le registre barrière est transformé en registre de base (relocation) . A chaque utilisation d'une adresse logique du programme, on ajoute à cette adresse la valeur du registre de base pour trouver l'adresse physique. L'utilisateur ne connaît plus les adresses physiques. Il travaille uniquement avec des adresses logiques (xdb).

Le moniteur a évidemment une valeur nulle pour son registre de base et donc peut adresser toute la mémoire. Le changement de la valeur du registre de base se fait de façon protégée en mode moniteur.

Ces deux systèmes de protection de la mémoire sont clairement mono-processus. Seul le moniteur peut être protégé par ces mécanismes, il n'est pas possible de protéger les processus entre eux.

8.1.5 Le swap

Il est possible avec les registres barrière ou les registres de base d'écrire des systèmes temps partagé, en utilisant le mécanisme de swap (échange).

Swapper, c'est échanger le contenu de la mémoire centrale avec le contenu d'une mémoire secondaire. Par extension swapper devient l'action de déplacer une zone mémoire de la mémoire vers le support de swap (en général un disque) ou réciproquement du périphérique de swap vers la mémoire.

Le système va réaliser cet échange à chaque changement de contexte. Les systèmes de swap utilisent une mémoire secondaire qui est en général un disque mais on peut utiliser d'autre supports secondaires plus lents ou plus rapides comme des bandes ou mémoires secondaires (non accessibles par l'unité de traitement).

8.1.6 Le coût du swap

Sur un tel système, le temps de commutation de tâches est très important. Il est donc nécessaire que chaque processus reste possesseur de l'unité de traitement un temps suffisamment long pour que le ralentissement dû au swap ne soit pas trop sensible. Que ce passe-t-il sinon ? Le système utilise la majeure partie de ses ressources à déplacer des processus en et hors mémoire centrale. L'unité de traitement n'est plus utilisée au maximum ...

8.1.7 Utilisation de la taille des processus

Pour améliorer les mécanismes de swap, on remarque que le temps de swap est proportionnel à la taille des données à déplacer. Pour améliorer les performances, il faut donc introduire la notion

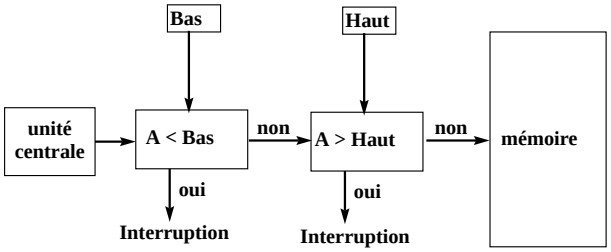

FIG. 8.9 – Double registre barrière.

de taille effective d'un processus, ce qui permet d'améliorer le débit mais cela impose que toutes les augmentations ou réductions de taille d'un processus utilisateur soient réalisée par un appel système (`sbrk`) afin que le noyau connaisse à tout moment la taille réelle de chaque processus.

8.1.8 Swap et exécutions concurrentes

Une autre approche très efficace est de réaliser le swap pendant l'exécution d'un autre processus. Mais avec le système de registres de relocation c'est dangereux. En effet nous ne pouvons pas assurer qu'un processus utilisateur donné ne va pas écrire dans les adresses réservées à un autre processus.

8.1.9 Contraintes

Le swap introduit d'autres contraintes : un processus doit être en préempté actif pour être swappé, c'est à dire n'être en attente d'aucune entrée-sortie. En effet, si P1 demande une E/S et pendant cette demande il y a échange de P1 et P2, alors la lecture demandée par P1 a lieu dans les données de P2.

8.1.10 Deux solutions existent

Soit ne jamais swapper de processus en attente d'entrées-sorties. Soit réaliser toutes les entrées-sorties dans des buffers internes au noyau (solution UNIX), ce qui a pour coût une recopie mémoire à mémoire supplémentaire par E/S. Les transferts entre le noyau et le processus ayant lieu uniquement quand le processus est en mémoire.

8.1.11 Les problèmes de protection

Nous venons d'apercevoir des problèmes de protection entre un processus et le noyau. Si l'on autorise plusieurs processus à résider en mémoire en même temps, il nous faut un mécanisme de protection inter-processus.
Deux méthodes sont couramment utilisées : les extensions du registre barrière et du registre de base (relocation).

8.1.12 Les registres doubles

Deux registres Barrière Bas et Haut
Si Adresse $<$ Bas $\longrightarrow$ lever une exception erreur d'adresse
Si Adresse $>=$ Haut $\longrightarrow$ lever une exception erreur d'adresse
Sinon adresse correcte.

Deux registres de relocation *Base* et *Limit*, on travaille avec des adresses logiques *Limit* donne la valeur maximale d'une adresse logique et *Base* donne la position en mémoire de l'adresse logique

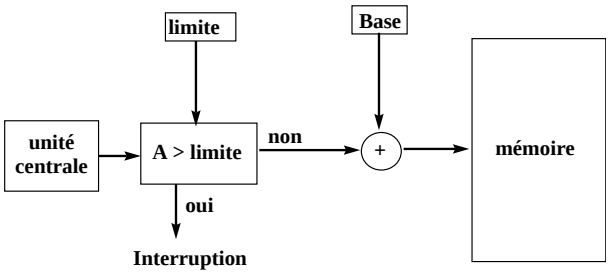

FIG. 8.10 – Base et Limite.

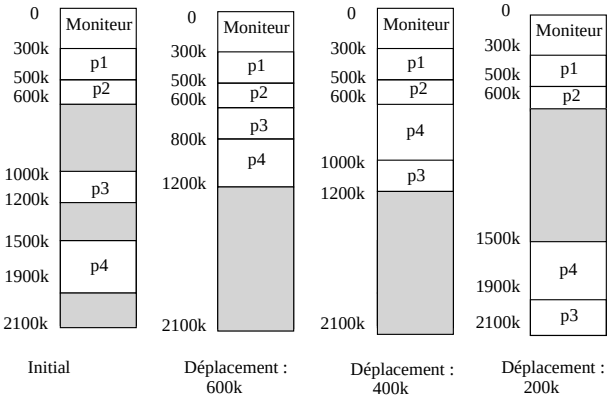

FIG. 8.11 – Une situation d'ordonnancement de processus en mémoire.

zéro.

Si Adresse >= *Limit* ⟶ lever une exception erreur d'adresse
sinon utiliser l'adresse physique Adresse+*Base*.

8.2 Ordonnancement en mémoire des processus

Les choix de l'implémentation des mécanismes d'adressage influence énormément l'ordonnancement des processus.

Nous travaillons dans le cas d'un système de traitement par lots c'est à dire en temps partagé mais les processus restent en mémoire tout le temps de leur exécution. S'il n'y a plus de place le processus est mis en attente (i.e. non chargé en mémoire).

Nous devons résoudre le problème suivant : il nous faut un algorithme pour choisir dynamiquement, parmi les blocs libres de la mémoire centrale, celui qui va recevoir le nouveau processus (algorithme d'allocation de mémoire à un processus). On reconnaît en général trois méthodes :

First-fit Le premier bloc suffisamment grand pour contenir notre processus est choisi.

Best-fit Le plus petit bloc suffisamment grand pour contenir notre processus est choisi.

Worst-fit Le bloc qui nous laisse le plus grand morceau de mémoire libre est choisi (le plus grand bloc).

De nombreuse expériences pratiques et des simulations ont montré que le meilleur est first-fit puis best-fit et que ces deux algorithmes sont beaucoup plus efficaces que worst-fit. **Compactage** On cherche à améliorer ces mécanismes en défragmentant la mémoire c'est à dire en déplaçant les processus en mémoire de façon à rendre contiguës les zones de mémoire libre de façon à pouvoir les utiliser.

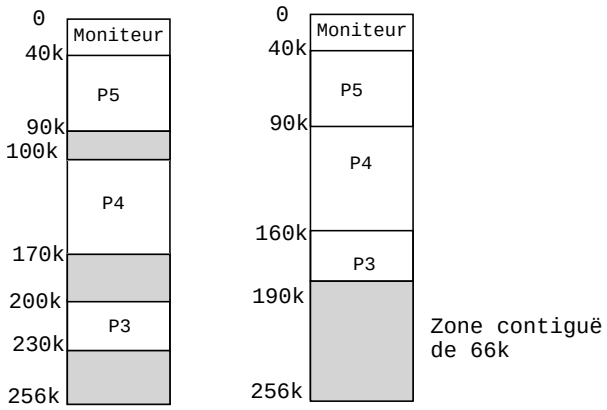

FIG. 8.12 – Compactage

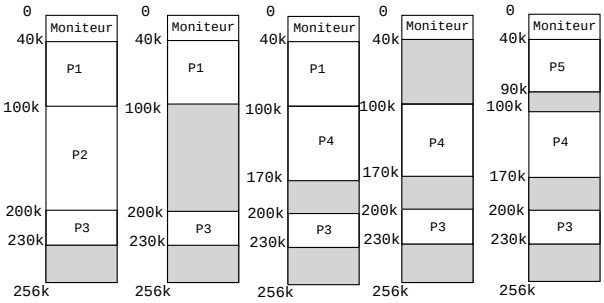

FIG. 8.13 – Plusieurs déplacements possibles.

8.3 Allocation non-contiguë

8.3.1 Les pages et la pagination

Pour accélérer ces mécanismes d'allocation, la notion de page a été introduite.

On va découper la mémoire et les processus en pages. Grâce à ce système, il ne sera plus nécessaire de placer les processus dans une zone contigüe de la mémoire. Il devient possible d'allouer de la mémoire à un processus sans avoir à réaliser de compactage !

Ce principe des page nécessite de nouvelles possibilités matérielles. Toute adresse est maintenant considérée comme un couple

(Numéro de page, Position dans la page)

A : adresse logique, P : taille de page

Numéro de page = A div P

Position = A modulo P

8.3.2 Ordonnancement des processus dans une mémoire paginée

Le choix de l'organisation mémoire a une influence prépondérante sur l'ordonnancement des processus, qui devient beaucoup plus indépendant de la mémoire quand celle-ci est paginée.

Le désavantage de la méthode de gestion de mémoire par un mécanisme de page est le phénomène de **fragmentation interne**. On alloue une page entière alors que le processus ne l'utilise qu'en partie. Mais la taille des mémoires et des processus deviennent tels par rapport aux tailles de page que cette perte devient minime.

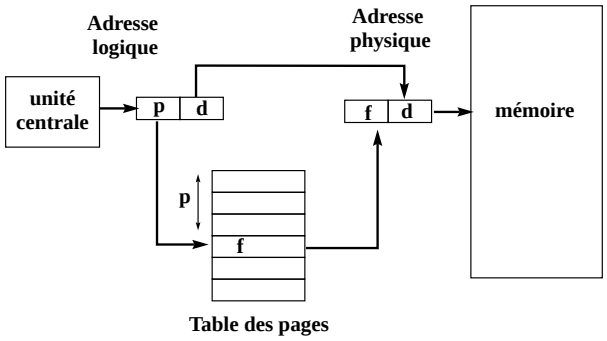

FIG. 8.14 – Calcul d'une adresse avec la table des pages

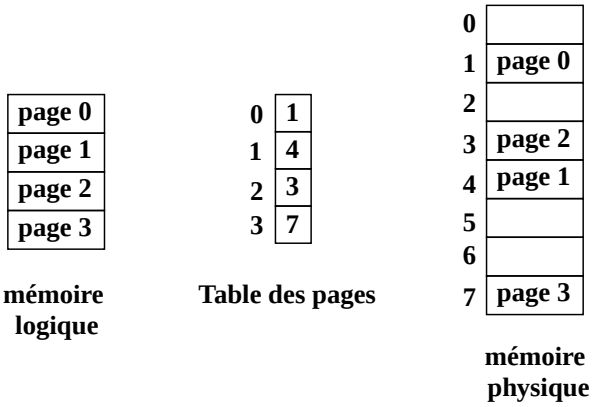

FIG. 8.15 – La mémoire logique et la Table des pages.

Un avantage des pages est une plus grande simplicité du partage de la mémoire entre différents processus. En particulier quand plusieurs processus partagent le même code. La page qui contient du code utilisé par les processus sera partageable et protégée en écriture.

Sous Unix le compilateur produit automatiquement des programmes dont la partie code est partageable.

8.3.3 Comment protéger la mémoire paginée

Les protections d'accès sont faites au niveau de la table des pages.

On a une table des pages globale. C'est donc le système qui alloue les pages à un processus, qui par construction (du système de pagination) ne peut pas écrire en dehors de ses propres pages. De plus, dans la table des pages d'un processus, des drapeaux indiquent le type de page (droits d'accès en lecture/écriture/exécution).

8.3.4 La mémoire segmentée

Nous venons de voir que les adresses logiques utilisées par le programmeur sont différentes des adresses physiques.

La mémoire segmentée est une organisation de la mémoire qui respecte le comportement usuel des programmeurs, qui généralement voient la mémoire comme un ensemble de tableaux distincts contenant des informations de types différents. Un segment pour chaque type : données, code, table des symboles, librairies etc. Ces différentes zones ayant des tailles variées, et parfois variables au cours du temps (le tas par exemple).

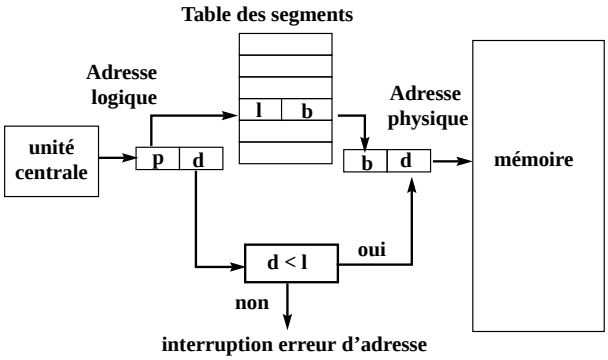

FIG. 8.16 – Mémoire segmentée

La mémoire segmentée non paginée pose des problèmes de compactage (défragmentation). La stratégie idéale est : la mémoire en segments paginés.