Chapitre 4: gestion de la mémoire centrale

1 Introduction

La mémoire physique est un ensemble d'emplacement. Pour qu'un programme puisse s'exécuter il doit être chargé dans la mémoire principale (MP).
Une MP est un ensemble de mots ou d'octets chacun possède sa propre adresse.

2 Le gestionnaire de la mémoire

Un géstionnaire de mémoire est un module du système d'exploitation dont le rôle est la gestion de la mémoire principale. Il doit viser les objectifs suivants: *la réallocation*, *la protection* et *le partage*.

2.1 La réallocation

Le système d'exploitation alloue à chaque programme une zone mémoire, mais tous les programmes n'ont pas la même taille. Chaque fois qu'un utilisateur demande le lancement d'un programme, le système doit trouver une place dans la mémoire principale pour le charger, donc il y a réorganisation des programmes en mémoires.

Exemple Soient les programmes P_1, P_2 et P_3 en mémoire. Si un 4ème programme de taille 150 demandait à être exéctué, ce sera impossible de charger ce programme. Le programme P_4 peut théoriquement être chargé. Malheureusement, dans cet exemple, ce sera encore impossible, car les 150 unités disponibles ne forment pas un espace contigu dans lequel on peut charger le programme P_4. Un réarrangement de l'espace mémoire permettra de charger P_4 comme suit:

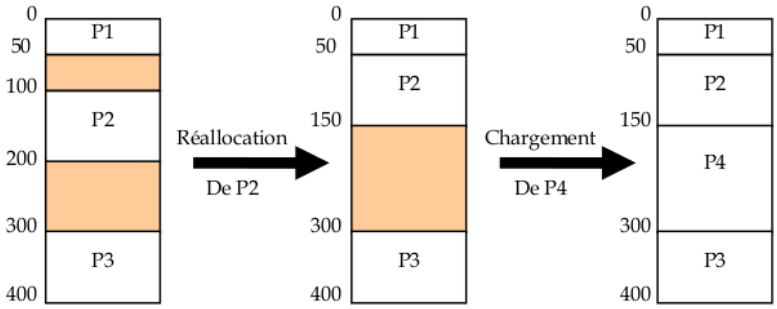

Figure 1: Principe de réallocation

2.2 La protection

Un processus P_i ne peut accéder à l'espace mémoire d'un autre processus P_j que s'il est autorisé.

2.3 Le partage

Parfois, il est utile de partager un espace mémoire entre plusieurs processus. Ainsi, le gestionnaire de mémoire doit autoriser des accès contrôlés sans compromettre la protection.

3 Représentation des adresses d'un objet

- **Adresse symbolique** représente les noms des objets (données et instructions) contenus dans le code source. **Exemple:** int compteur;
- **Adresse logique** correpond à la traduction des adresses symboliques lors de la phase de compilation;
- **Adresse physique** représente l'emplacement physique des adresses logiques d'un programme lors de son exécution.

4 Swapping

Un programme doit être chargé dans la mémoire principale pour être exécuté. Cependant, un processus peut être transféré temporairement de la mémoire principale à une mémoire secondaire (*swap out*) et être ensuite ramené en mémoire principale pour continuer son exécution (*swap in*).

5 Allocation de la mémoire

Un programme pour s'exécuter doit être chargé en mémoire principale. Le système d'exploitation doit lui réserver une zone mémoire.

5.1 Partition unique

La mémoire principale est partitionnée en deux partitions. Une partition réservée pour le système d'exploitation et l'autre partition est réservée aux processus utilisateurs. Un seul processus peut se retrouver à l'instant t dans la mémoire principale.

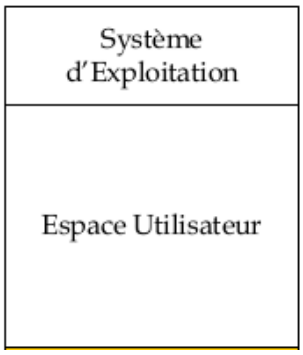

Figure 2: Partition unique

5.2 Partitions multiples

L'espace utilisateur est divisé en plusieurs partitions.

5.2.1 Partitions multiples statiques

L'espace utilisateur est découpé en partition de taille fixe (FIGURE 3).

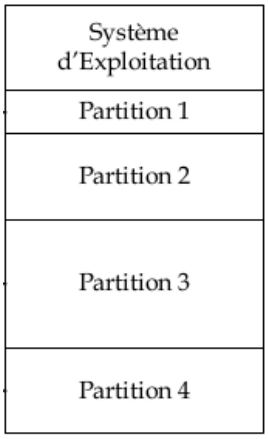

Figure 3: Partitions multiples statiques

Fragmentation interne Soient un programme de taille M et une partition de taille N. Si la partition P est allouée au programme avec $N > M$ alors, la partie non occupée par le programme est appelée *fragmentation interne*.

Fragmentation externe Soit un programme de taille M. Si toutes les partitions libre de taille N_i avec $N_i < M$ alors, le programme ne peut être chargé dans aucune partition libre, c'est la *fragmentation externe*.

5.2.2 Partitions multiples dynamiques

On partitionne la mémoire principale dynamiquement selon la demande. À chaque processus on alloue une partition exactement égale à sa taille.

Stratégies de placement Dans un partitionnement multiple dynamique, le placement des programmes dans les partitions se fait selon un certain nombre de stratégies possibles. On distingue les stratégies suivantes:

- First fit (premier qui convient);
- Best fit (le meilleur qui convient);
- Worst fit (la pire qui convient).

Remarques

- La stratégie du premier qui convient est rapide, mais provoque des pertes importantes en fragmentation de la mémoire. Les deux autres stratégies sont lourdes à mettre en oeuvre;
- Un problème de fragmentation externe se pose et la solution c'est le *compactage*.

5.3 Compactage

Le compactage est une solution pour la fragmentation externe qui permet de regrouper les espaces inutilisés. L'opératioini de compactage est effectuée quand un programme qui demande d'être exécuté ne trouve pas une partition assez grande, mais sa taille est plus petite que la fragmentation externe existante (FIGURE 4).

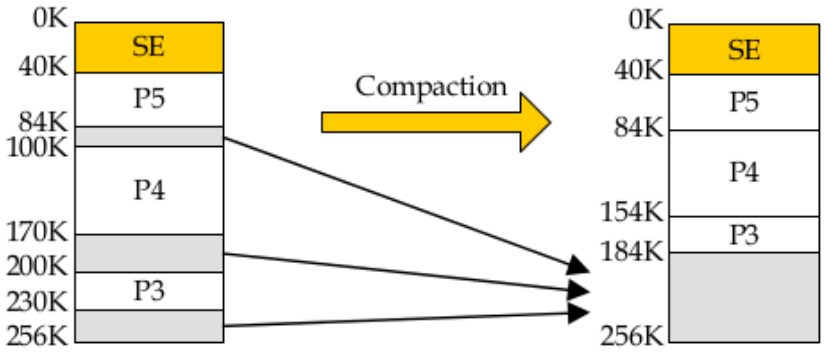

Figure 4: Compactage

5.4 Pagination

C'est une autre solution qui pallie au problème de la fragmentation externe. Elle permet aussi l'allocation d'espace non nécessairement contigu pour l'exécution d'un processus. Dans ce cas, l'espace d'adressage d'un processus est divisé en partitions égales appelées *pages*.

Chaque adresse générée par le processeur central est divisée en deux parties: un numéro de page p et un déplacement page d. La table des pages contient l'adresse de base de chaque page dans la mémoire physique.

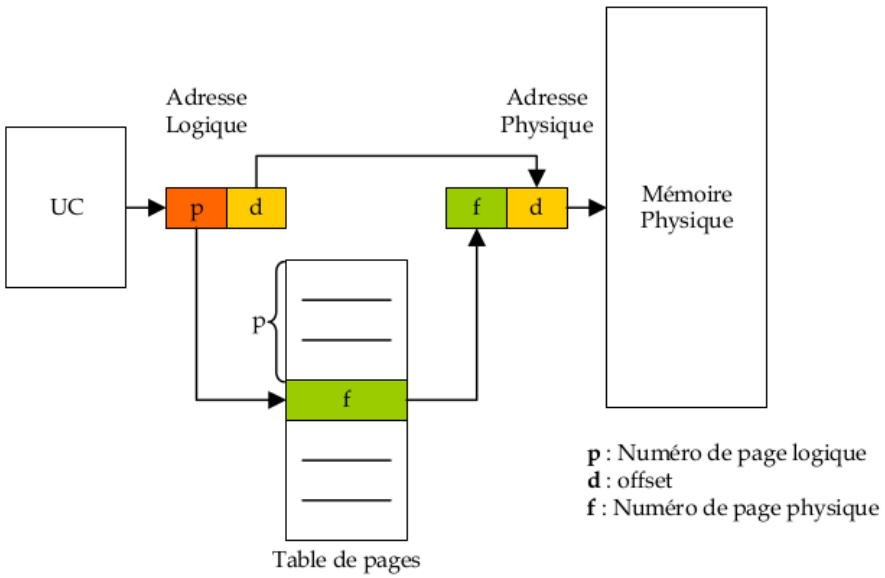

Figure 5: Correspondance d'adresses

Remarque Si un processus nécessite n pages et 1 octet, on doit lui allouer $(n+1)$ pages.

5.5 Segmentation

Chaque processus est constitué d'un ensemble de segments. Chaque segment est un espace linéaire. Les segments sont des espaces d'adressages indépendants de différentes longueurs et qui peuvent

même varier en cours d'utilisation. Ils correspondent à des subdivisions logiques déterminées par le programmeur ou par le compilateur.

Dans une mémoire segmentée, chaque unité logique d'un programme usager est stockée dans un bloc mémoire, appelé *segment* à l'intérieur duquel les adresses sont relatives au début du segment. Ces segments sont de tailles différentes. Un programme sera donc constitué d'un ensemble de segments de code et de données, pouvant être dispersés en mémoire principale.

La segmentation facilite l'édition de liens, ainsi que le partage entre processus de segments de données ou de codes.

Contrairement à la pagination, la segmentation permet d'éliminer la fragmentation interne car l'espace mémoire alloué à un segment est de taille égale exactement à la taille du segment. Cependant, une fragmentation externe peut se produire due au fait qu'on fait une allocation dynamique de l'espace mémoire.

5.5.1 Schéma de translation d'adresses

Chaque segment est repéré par son numéro S et sa longueur variable L. Un segment est un ensemble d'adresse logique contiguës.
Une adresse logique est donnée par un couple (S, d), où S est le numéro du segment et d le déplacement dans le segment.
L'association d'une adresse logique à une adresse physique est décrite dans une table appelée *table de segments*.
Chaque entrée de la table de segments possède un *segment de base* et un *segment limite*. Le segment de base contient l'adresse physique de début où le segment réside en mémoire, tandis que le registre limite spécifie la longueur du segment.

La transition des adresses logiques en adresses physiques est montrée dans la FIGURE 6

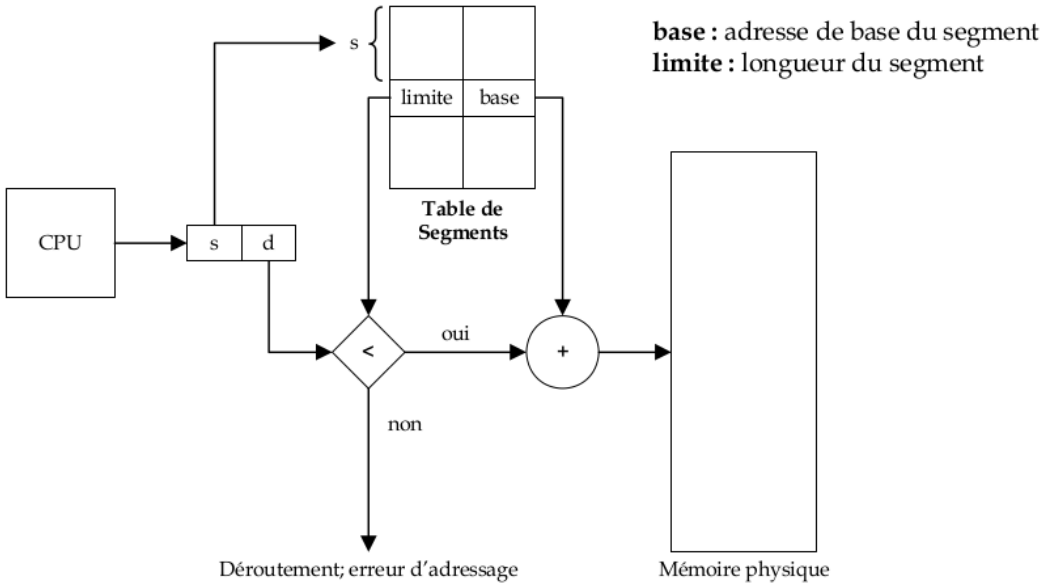

Figure 6: Matériel pour la segmentation