Chapitre 3 GESTION DE LA MEMOIRE

Pour pouvoir utiliser un ordinateur en multiprogrammation, le SE charge plusieurs processus en mémoire centrale (MC). La façon la plus simple consiste à affecter à chaque processus un ensemble d'adresses contiguës.

Quand le nombre de tâches devient élevé, pour satisfaire au principe d'équitabilité et pour minimiser le temps de réponse des processus, il faut pouvoir **simuler** la présence simultanée en MC de tous les processus. D'où la technique de "va et vient" ou **recouvrement (swapping)**, qui consiste à stocker temporairement sur disque l'image d'un processus, afin de libérer de la place en MC pour d'autres processus.

D'autre part, la taille d'un processus doit pouvoir dépasser la taille de la mémoire disponible, même si l'on enlève tous les autres processus. L'utilisation de pages (mécanisme de **pagination**) ou de segments (mécanisme de **segmentation**) permet au système de conserver en MC les parties utilisées des processus et de stocker, si nécessaire, le reste sur disque.

Le rôle du gestionnaire de la mémoire est de connaître les parties libres et occupées, d'allouer de la mémoire aux processus qui en ont besoin, de récupérer de la mémoire à la fin de l'exécution d'un processus et de traiter le recouvrement entre le disque et la mémoire centrale, lorsqu'elle ne peut pas contenir tous les processus actifs.

1.GESTION SANS RECOUVREMENT, NI PAGINATION

1.1 La monoprogrammation

Il n'y a en MC que :

- un seul processus utilisateur,
- le processus système (pour partie en RAM, pour partie en ROM; la partie en ROM étant appelée BIOS [*Basic Input Output System*])
- les pilotes de périphériques

Cette technique en voie de disparition est limitée à quelques micro-ordinateurs . Elle n'autorise qu'un seul processus actif en mémoire à un instant donné.

1.2 La multiprogrammation

La multiprogrammation est utilisée sur la plupart des ordinateurs : elle permet de diviser un programme en plusieurs processus et à plusieurs utilisateurs de travailler en temps partagé avec la même machine.

Toutefois, on remarque que plus le nombre n de processus en MC est élevé, plus le taux d'utilisation du processeur est élevé : on a donc intérêt à augmenter la taille de la MC.

Une solution simple consiste à diviser la mémoire en **n partitions fixes**, de tailles pas nécessairement égales. Il existe deux méthodes de gestion :

- on crée une file d'attente par partition. Chaque nouveau processus est placé dans la file d'attente de la plus petite partition pouvant le contenir. Inconvénients :
  * on perd en général de la place au sein de chaque partition
  * il peut y avoir des partitions inutilisées (leur file d'attente est vide)

- on crée une seule file d'attente globale. Il existe deux stratégies :
  * dès qu'une partition se libère, on lui affecte la première tâche de la file qui peut y tenir. Inconvénient : on peut ainsi affecter une partition de grande taille à une petite tâche et perdre beaucoup de place

  * dès qu'une partition se libère, on lui affecte la plus grande tâche de la file qui peut y tenir. Inconvénient : on pénalise les processus de petite taille.

3. GESTION AVEC RECOUVREMENT AVEC PAGINATION OU SEGMENTATION

La taille d'un processus doit pouvoir dépasser la taille de la mémoire physique disponible, même si l'on enlève tous les autres processus. Le principe de la **mémoire virtuelle** : le SE conserve en mémoire centrale les parties utilisées des processus et stocke, si nécessaire, le reste sur disque. Mémoire virtuelle et multiprogrammation se complètent bien : un processus en attente d'une ressource n'est plus conservé en MC, si cela s'avère nécessaire.

La mémoire virtuelle fait appel à deux mécanismes : segmentation ou pagination. La mémoire est divisée en segments ou pages. Sans recours à la mémoire virtuelle, un processus est entièrement chargé à des adresses contiguës ; avec le recours à la mémoire virtuelle, un processus peut être chargé dans des pages ou des segments non contigus.

3.1 La pagination

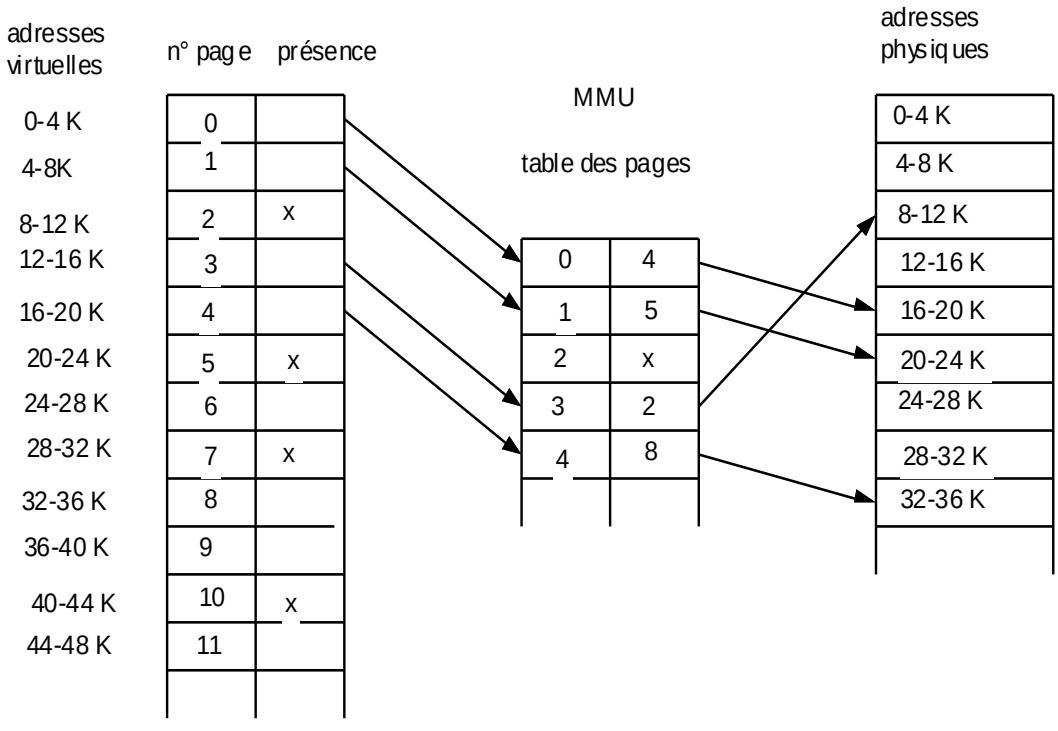
L'espace d'adressage d'un processus est divisé en petites unités de taille fixe appelées **pages**. La MC est elle aussi découpée en unités physiques de même taille appelées **cadres**. Les échanges entre MC et disques ne portent que sur des pages entières. De ce fait, l'espace d'adressage d'un processus est potentiellement illimité (limité à l'espace mémoire total de la machine). On parle alors d'**adressage virtuel**.

Pour un processus, le système ne chargera que les pages utilisées. Mais la demande de pages à charger peut être plus élevée que le nombre de cadres disponibles. Une gestion de l'allocation des cadres libres est nécessaire.

Dans un SE sans mémoire virtuelle, la machine calcule les adresses physiques en ajoutant le contenu d'un registre de base aux adresses relatives contenues dans les instructions du processus. Dans un SE à pagination, un sous-ensemble inséré entre l'UC et la MC, **la MMU** (Memory Management Unit ou unité de gestion de la mémoire) traduit les adresses virtuelles en adresses physiques.

La MMU mémorise :
- les cadres physiques alloués à des processus (sous forme d'une table de bits de présence)
- les cadres mémoire alloués à chaque page d'un processus (sous forme d'une table des pages)

On dira qu'une page est **mappée** ou **chargée** si elle est physiquement présente en mémoire.



Dans l'exemple précédent, les pages ont une taille de 4 Ko. L'adresse virtuelle 12292 correspond à un déplacement de 4 octets dans la page virtuelle 3 (car 12292 = 12288 + 4 et 12288 = 12*1024). La page virtuelle 3 correspond à la page physique 2. L'adresse physique correspond donc à un déplacement de 4 octets dans la page physique 2, soit : (8*1024) + 4 = 8196 .

Par contre, la page virtuelle 2 n'est pas mappée. Une adresse virtuelle comprise entre 8192 et 12287 donnera lieu à **un défaut de page**. Il y a défaut de page quand il y a un accès à une adresse virtuelle correspondant à une page non mappée. En cas de défaut de page, un déroutement se produit (trap) et le processeur est rendu au SE. Le système doit alors effectuer les opérations suivantes :

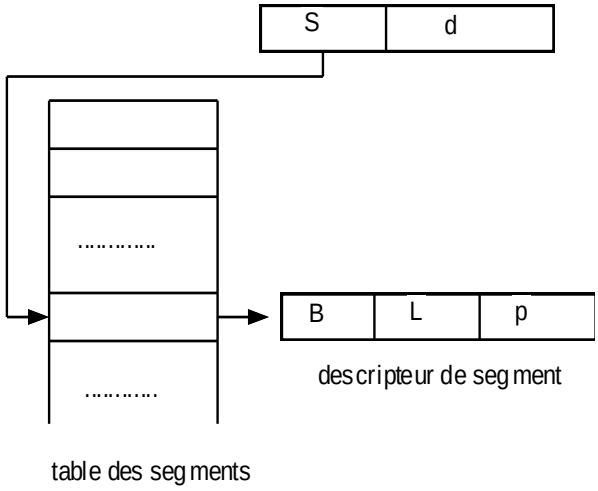
- déterminer la page à charger
- déterminer la page à décharger sur le disque pour libérer un cadre
- lire sur le disque la page à charger
- modifier la table de bits et la table de pages

3.2 La segmentation

Dans cette solution, l'espace d'adressage d'un processus est divisé en **segments**, générés à la compilation. Chaque segment est repéré par son numéro S et sa longueur variable L. Un segment est un ensemble d'adresses virtuelles contiguës.

Contrairement à la pagination, la segmentation est "connue" du processus : une adresse n'est plus donnée de façon absolue par rapport au début de l'adressage virtuel; une adresse est donnée par un couple (S , d), où S est le n° du segment et d le déplacement dans le segment, $d \in [0 , L [$.

Pour calculer l'adresse physique, on utilise une **table des segments** :



B : adresse de base (adresse physique de début du segment)
L : longueur du segment ou limite
p : protection du segment

L'adresse physique correspondant à l'adresse virtuelle (S , d) sera donc B + d, si d <= L

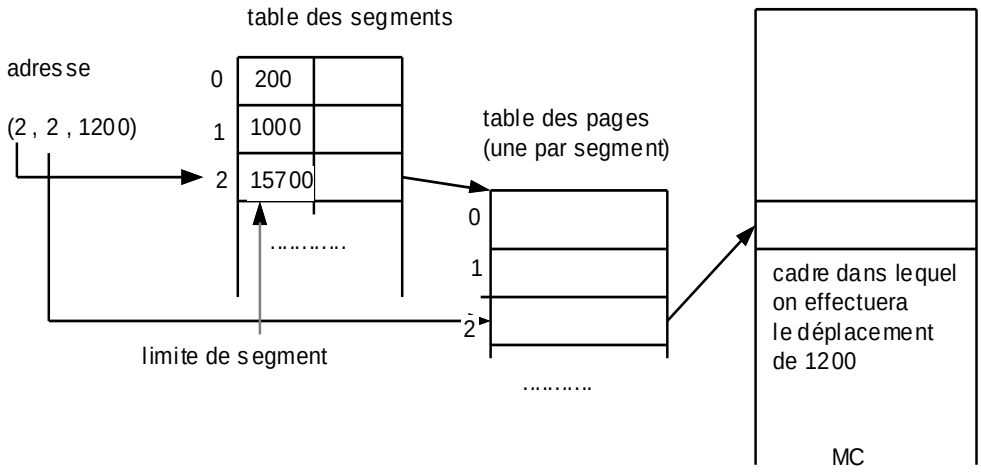
La segmentation simplifie la gestion des objets communs (rangés chacun dans un segment), notamment si leur taille évolue dynamiquement.

3.3 La segmentation paginée

La taille d'un segment peut être importante, d'où un temps de chargement long qui peut en résulter. La pagination des segments peut être une solution.

Une adresse virtuelle (S , d), avec S n° de segment et d déplacement dans le segment, est transformée en (S , P , d'), où P est un n° de page et d' un déplacement dans la page P.

Exemple : avec l'hypothèse de pages de 4 Ko, l'adresse logique (2 , 9200) est transformée en (2 , 2, 1200)



Dans tous les cas, l'utilisation d'une mémoire associative, stockant des triplets (S, P, adresse de cadre), gérée comme fenêtre au voisinage du dernier cadre accédé, peut accélérer la recherche des cadres.

4. ALGORITHMES DE REMPLACEMENT DE PAGES

Les méthodes reposent sur 3 stratégies :

- une stratégie de **chargement** qui choisit les pages à charger et l'instant de chargement
- une stratégie de **placement** qui choisit un cadre libre pour chaque page à charger
- une stratégie de r**emplacement** destinée à libérer certains cadres. La recherche se fait soit sur l'ensemble des pages (recherche globale), soit sur les pages "appartenant" au processus (recherche locale). Les stratégies de placement sont dépendantes des stratégies de remplacement : on charge nécessairement une page dans le cadre qui vient d'être libéré.

On peut distinguer deux catégories de méthodes :

- **pagination anticipée (préchargement)** : on charge les pages à l'avance; mais comment prévoir efficacement ?
- **pagination à la demande** : le chargement n'a lieu qu'en cas de défaut de page et on ne charge que la page manquante.

On appelle **suite (chaine) de références** $w = p_1\ p_2\ ...\ p_n$ une suite de numéros de pages correspondant à des accès à ces pages. On dira qu'un algorithme A de pagination sur m cadres est **stable** si :

$$\forall\ m\ ,\ \forall\ w\ ,\ \text{on a : Coût}\ (A\ ,\ m\ ,\ w) <= \text{Coût}\ (\ A\ ,\ m+1\ ,\ w)$$

Cette notion est importante pour éviter l'écroulement du SE (thrashing) par une génération excessive de défauts de pages.

4.1 Remplacement de page optimal

Le SE indexe chaque page par le nombre d'instructions qui seront exécutées avant qu'elle ne soit référencée. En cas de nécessité, le SE retire la page d'indice le plus élevé, c'est à dire la page qui sera référencée dans le futur le plus lointain.
Cet algorithme est pratiquement impossible à appliquer (comment calculer les indices des pages ?). Toutefois, avec un simulateur, on peut évaluer les performances de cet algorithme et s'en servir comme référence pour les suivants. En outre, cet algorithme est stable.

4.2 NRU (not recently used)

Le SE référence chaque page par deux bits R (le plus à gauche) et M initialisés à 0. A chaque accès en lecture à une page, R est mis à 1. A chaque accès en écriture, M est mis à 1. A chaque interruption d'horloge, le SE remet R à 0.

Les bits R-M forment le code d'un index de valeur 0 à 3 en base 10. En cas de défaut de page, on retire une page au hasard dans la catégorie non vide de plus petit index. On retire donc préférentiellement une page modifiée non référencée (index 1) qu'une page très utilisée en consultation (index 2). Cet algorithme est assez efficace.

4.3 FIFO (first in, first out)

Le SE indexe chaque page par sa date de chargement et constitue une liste chaînée, la première page de la liste étant la plus ancienne ment chargée et la dernière la plus récemment chargée. Le SE replacera en cas de nécessité la page en tête de la liste et chargera la nouvelle page en fin de liste.

Deux critiques à cet algorithme :

- ce n'est pas parce qu'une page est la plus ancienne en mémoire qu'elle est celle dont on se sert le moins

- l'algorithme n'est pas stable : quand le nombre de cadres augmente, le nombre de défauts de pages ne diminue pas nécessairement (on parle de l'anomalie de BELADY).

Amélioration 1 : le SE examine les bits R et M (gérés comme ci-dessus en 4.2) de la page la plus ancienne en MC (en tête de la liste). Si cette page est de catégorie 0, il l'ôte. Sinon, il teste la page un peu moins ancienne, etc... S'il n'y a aucune page de catégorie 0, il recommence avec la catégorie 1, puis éventuellement avec les catégories 2 et 3.

Amélioration 2 : **Algorithme de la seconde chance** : R et M sont gérés comme ci-dessus en 4.2. Le SE examine le bit R de la page la plus ancienne (tête de la liste). Si R = 0, la page est remplacée, sinon R est mis à 0, la page est placée en fin de liste et on recommence avec la nouvelle page en tête. Si toutes les pages ont été référencées depuis la dernière RAZ, on revient à FIFO, mais avec un surcoût.

4.4 LRU (least recently used)

En cas de nécessité, le SE retire la page la moins récemment référencée. Pour cela, il indexe chaque page par le temps écoulé depuis sa dernière référence et il constitue une liste chaînée des pages par ordre décroissant de temps depuis la dernière référence.

L'algorithme est stable. Mais il nécessite une gestion coûteuse de la liste qui est modifiée à chaque accès à une page.

Variantes :

NFU (not frequently used) ou **LFU** (least frequently used) : le SE gère pour chaque page un compteur de nombre de références . Il cumule dans ce compteur le bit R juste avant chaque RAZ de R au top d'horloge. Il remplacera la page qui aura la plus petite valeur de compteur. Inconvénient : une page qui a été fortement référencée, mais qui n'est plus utilisée ne sera pas remplacée avant longtemps.

Vieillisement (aging) ou **NFU modifié** : avant chaque RAZ de R, le SE décale le compteur de 1 bit à droite (division entière par 2) et additionne R au bit de poids **fort**. En cas de nécessité, on ôtera la page de compteur le plus petit. Mais en cas d'égalité des compteurs, on ne sait pas quelle est la page la moins récemment utilisée.

Matrice : s'il y a N pages, le SE gère une matrice (N , N+1). A chaque référence à la page p, le SE positionne à 1 tous les bits de la ligne p et à 0 tous les bits de la colonne p. En outre, le 1er bit de la ligne p vaut 0 si la page est chargée, 1 sinon. On ôtera la page dont la ligne a la plus petite valeur.

4.5 Améliorations

Le dérobeur de pages : dès qu'un seuil minimal de cadres libres est atteint, le SE réveille un dérobeur de pages. Celui-ci supprime les pages les moins récemment utilisées, par algorithme d'aging, et les range dans la liste des cadres libres, ceci jusqu'à un seuil donné. Le SE vérifie qu'une page référencée sur défaut de page n'est pas dans la liste.

Problèmes d'entrée/sortie : on a intérêt, soit à verrouiller les pages concernées par les E/S pour éviter de les ôter, soit à effectuer les E/S dans des tampons du noyau et à copier les données plus tard dans les pages des utilisateurs.

Pages partagées : lorsque plusieurs processus exécutent le même ensemble de code, on a intérêt à utiliser des pages de code partagées, plutôt que des pages dupliquées. Mais on aura à prévoir une gestion spécifique des pages partagées pour éviter des défauts de pages artificiels.