Chapitre 4 : Gestion de la mémoire

4.1 Introduction

La mémoire physique sur un système se divise en deux catégories :

- la mémoire vive : composée de circuit intégrés, donc très rapide.
- la mémoire de masse (secondaire) : composée de supports magnétiques (disque dur, bandes magnétiques...), qui est beaucoup plus lente que la mémoire vive.

La mémoire est une ressource rare. Il n'en existe qu'un seul exemplaire et tous les processus actifs doivent la partager. Si les mécanismes de gestion de ce partage sont peu efficaces l'ordinateur sera peu performant, quelque soit la puissance de son processeur. Dans ce chapitre on va mettre l'accent sur la gestion de la mémoire pour le stockage des processus concurrents.

4.2 Gestion sans recouvrement ni pagination

4.2.1 La monoprogrammation : Le modèle de base

La représentation de base, pour un système monoprocesseur et mono tâche, est montrée dans figure suivante : il s'agit d'une partition contiguë.

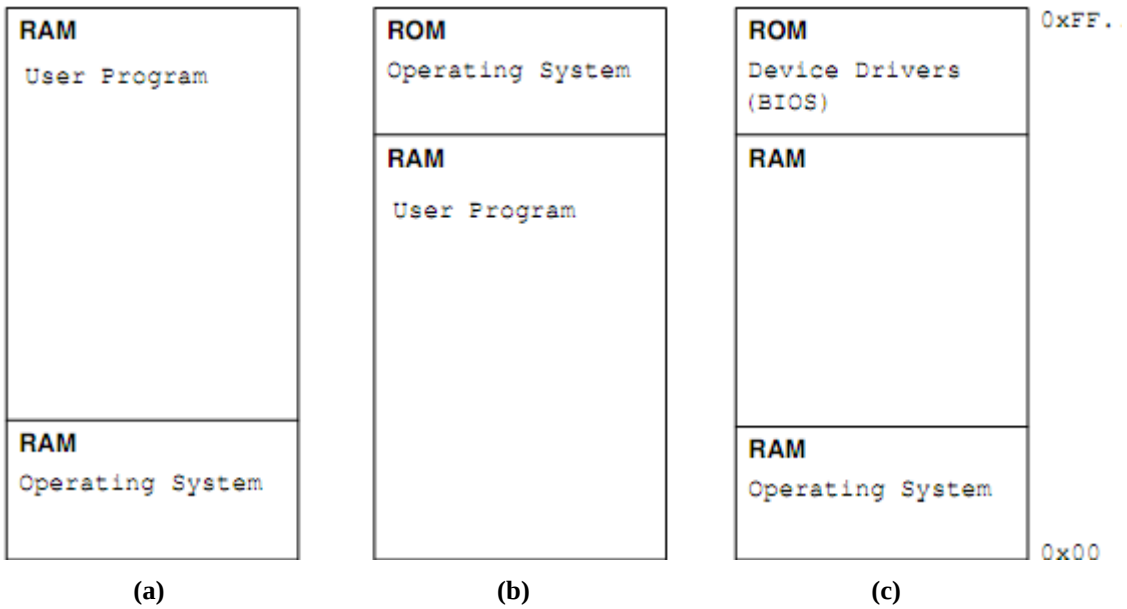


Figure 5-1 : Les trois organisations de base pour un système mono tâche.

En général, le système d'exploitation se trouve au niveau des premières adresses de la zone mémoire de la RAM. Pour des systèmes avec un SE embarqué (consoles de jeu, téléphones mobiles, etc) le système se trouve souvent dans une partie non modifiable (ROM).

Afin de garantir de façon transparente mais flexible, l'amorçage d'un système quelconque, on retrouve souvent une combinaison des deux approches (RAM et ROM). La ROM contient alors un système minimal permettant de piloter les périphériques de base (clavier -- disque -- écran) et de charger le code d'amorçage à un endroit bien précis. Ce code est exécuté lors de la mise sous tension de l'ordinateur, et le « vrai » système d'exploitation, se trouvant dans la zone d'amorçage sur le disque, est ensuite chargé, prenant le relais du système minimal.

4.2.2 La multiprogrammation

a. Multiprogrammation avec partitions fixes

i. Partition fixes de tailles égales

Cette partition peut être faite une fois pour toute au démarrage du système par l'opérateur de la machine,

qui subdivise la mémoire en partitions fixes de tailles égales. Chaque nouveau processus est placé dans une partition vide. Ceci engendrera le problème de fragmentation interne due au faite que si la taille de partition est supérieure à l'espace recueilli par un certain processus le reste de cette partition restera vide ce qui cause une perte d'espace mémoire.

Le sous système chargé de la gestion de mémoire met en place une structure des données appelée table des partitions ayant pour rôle l'indication de l'état (libre ou occupée) de chaque partition en identifiant chacune soit par son adresse soit par son numéro.

ii. Partition fixes de tailles inégales

Dans ce cas la mémoire est subdivisée en partitions de tailles inégales. Chaque nouveau processus est placé dans la file d'attente de la plus petite partition qui peut le contenir. Cette façon de faire peut conduire à faire attendre un processus dans une file, alors qu'une autre partition pouvant le contenir est libre.

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.
    L'alternative à cette approche consiste à n'utiliser qu'une seule file d'attente : dès qu'une partition se libère, le système y place le premier processus de la file qui peut y tenir. Cette solution réduit la fragmentation interne de la mémoire.

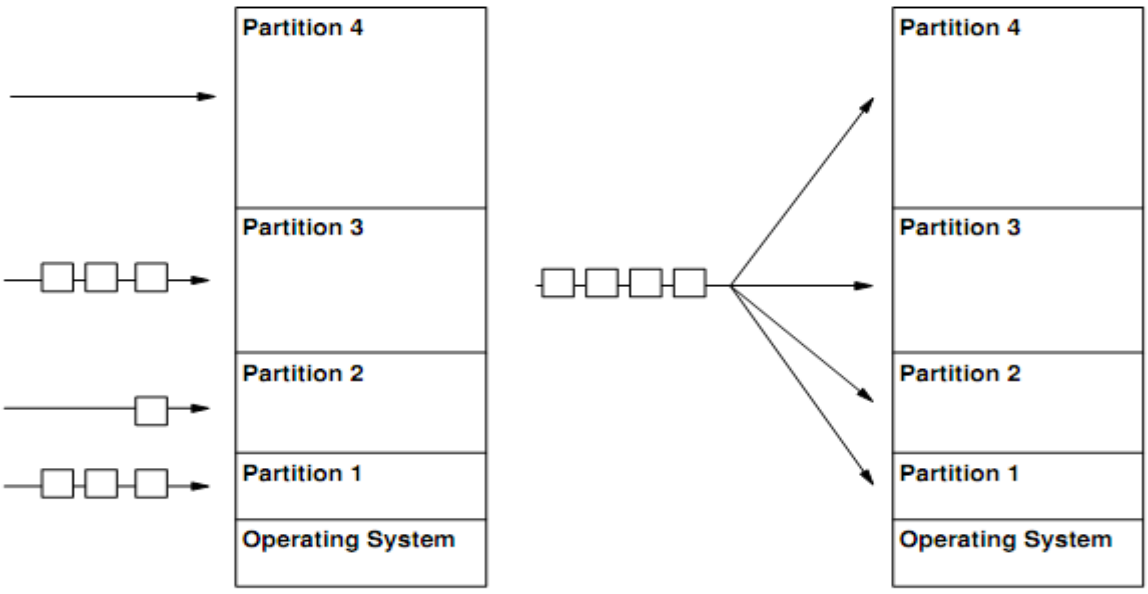


Figure 5-2 : Multiprogrammation && partition fixe

b. Multiprogrammation avec partitions variables

Au lancement du système, on crée une seule zone libre de taille maximale. Lorsqu'on charge un programme, on le place dans une zone libre suffisante, et on lui alloue exactement la mémoire nécessaire. Le reste devient une nouvelle zone libre. Lorsqu'un programme s'achève, sa partition redevient libre, et peut éventuellement grossir une zone libre voisine. Il n'y a donc ce qu'on appelle fragmentation externe.

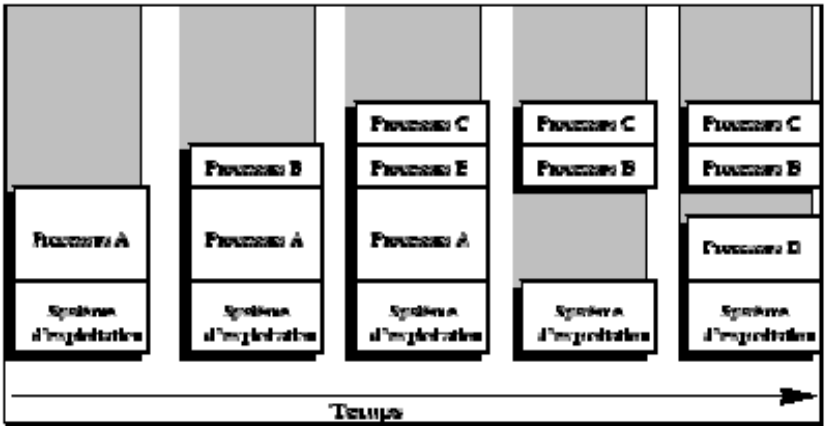


Figure 5-3 : Principe d'allocation de partitions de taille variable.

La solution la plus flexible est d'adopter un système avec des partitions de taille variable et un système de va-et-vient qui permet d'utiliser le disque comme mémoire secondaire et d'y stocker un nombre de processus inactifs ou en attente.

i- Stratégies de placement et d'allocation

Dans une organisation à partitions variables, il existe à chaque instant un ensemble de partitions libres de différentes tailles. Quand un processus demande de la mémoire, le gestionnaire de la mémoire centrale (module de SE) cherche parmi cet ensemble de partitions, une partition suffisamment grande pour ce processus.

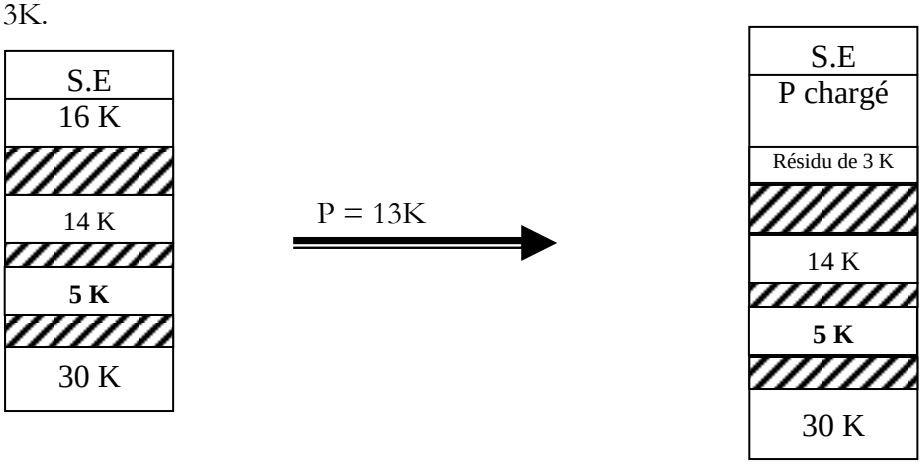
Le placement des programmes dans les partitions se fait alors suivant différentes stratégies :

1/ Stratégie du 1er qui convient (First Fit) :

Un programme est placé dans la partition trouvée dont la taille est au moins égale à celle du programme.

Exemple :

Soit le schéma de MC suivant avec quatre partitions libres. On désire allouer de l'espace à un programme P de taille 13K.



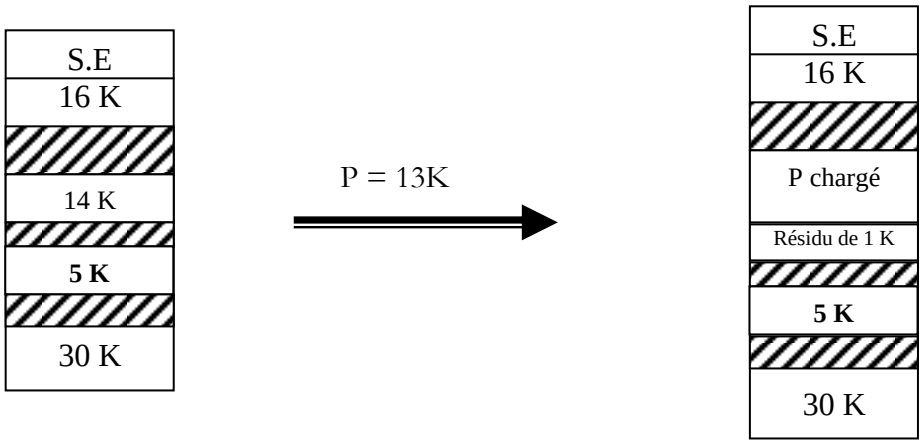
Avantage : stratégie rapide en temps d'exécution.

Inconvénients : on peut créer des partitions qui ne sont pas intéressantes (exemple 3K), qui ne seront jamais utilisées (fragmentation interne).

2/ Stratégie du meilleur qui convient (Best Fit)

Un programme est placé dans la partition dont le résidu mémoire sera minimal. C'est la plus petite partition dont la taille est supérieure ou égale à la taille du programme.

Exemple précédent :



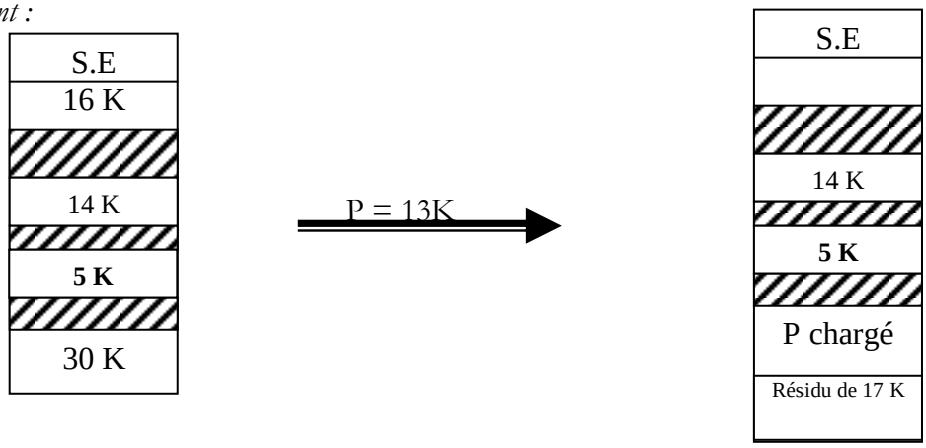
Avantage : stratégie qui tente de réduire la fragmentation de la mémoire (résidus de petites tailles).

Inconvénients : il faut connaître les tailles de chaque partition libre, et maintenir une liste des blocs ou partitions triées selon la taille. Le premier élément de la liste, pointe sur la plus petite partition.

3/ Stratégie du pire qui convient (Worst Fit) :

Un programme est placé dans la partition la plus large. La liste des blocs libres dans ce cas doit être triée selon la taille en ordre décroissant.

Exemple précédent :



Avantage : la stratégie tente de réduire la fragmentation de la mémoire centrale en plus petits résidents. En effet, en choisissant la plus large partition, le résidu sera assez grand pour pouvoir contenir de nouveaux programmes.

- Les simulations ont montré que la stratégie Worst Fit est meilleure que le Best Fit, et les deux sont meilleures que le First Fit.

ii- Compactage mémoire

L'organisation à partitions variables n'élimine pas complètement, la fragmentation externe (une partition M

est disponible mais le programme en attente nécessite un espace de taille N tel que N>M) quelle que soit la stratégie de placement utilisée. Il est alors possible d'avoir plusieurs partitions libres qui ne conviennent à aucun programme en attente. Afin de pallier ce problème, une réorganisation de la mémoire est effectuée. Elle est dite compactage.

Une opération de compactage consiste à rassembler les partitions résidus en une partition plus grande. Le compactage se fait par un déplacement de programmes.

Exemple :

On désire exécuter les programmes A, B, C, mais aucun ne peut l'être puisque aucune partition mémoire de taille supérieure ou égale n'est disponible dans le schéma mémoire suivant :

A	B	C
35	52	60

On exécute alors une opération de compactage de la mémoire centrale.

Remarque : le compactage est très coûteux, en temps et se fait alors périodiquement.

Structures utilisées pour l'état des partitions mémoire

Afin d'allouer une partition mémoire à un programme, le système doit garder trace de l'état libre ou occupé des partitions constituant la MC. Il existe deux manières pour gérer les partitions :

- Utilisation des bitmaps.
- Utilisation des listes chaînées.

a)Utilisation des bitmaps (table de bits)

La mémoire est divisée en unités (quelques mots mémoire à plusieurs kilo-octets), à chaque unité on fait correspondre dans une table de bits :

- la valeur 1 si l'unité mémoire est occupée ;
- la valeur 0 si l'unité mémoire est libre.

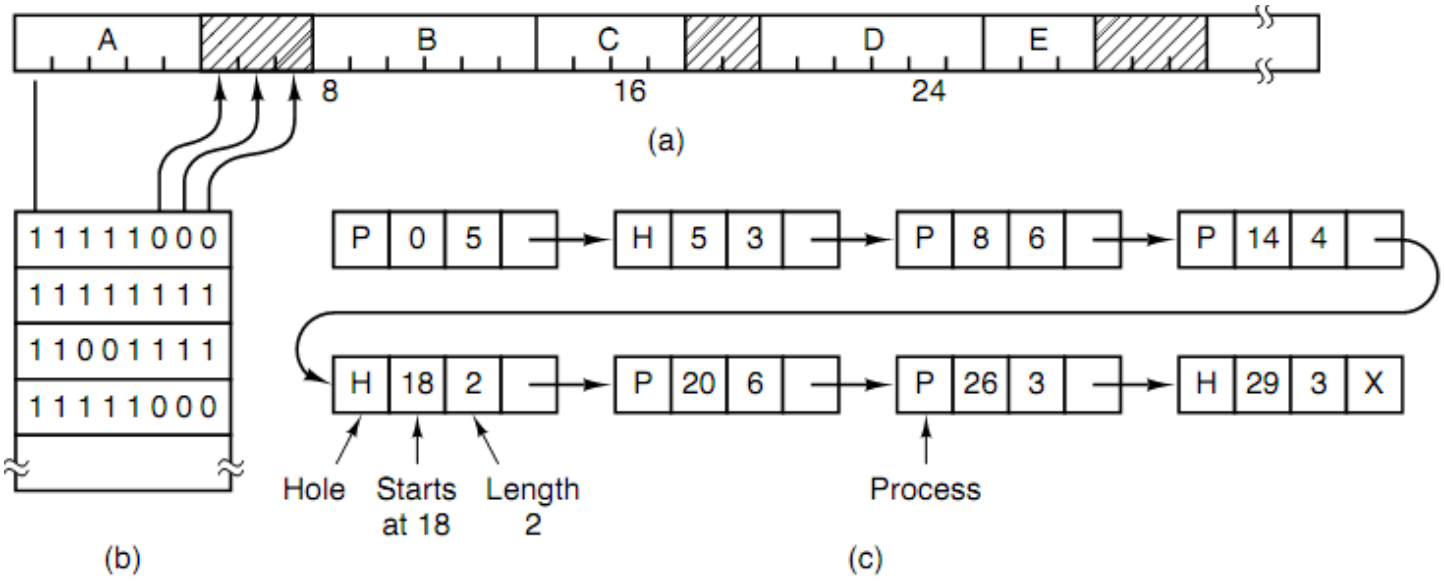


FIG. 4.4 – (a) Une partie de la mémoire occupée par cinq processus (A, B, C, D et E) et trois zones libres. Les régions hachurées sont libres. (b) La table de bits (0 = libre, 1 = occupée). (c) Liste chaînée des zones libres.

La figure 4.4 (a) et (b) représente un exemple de table de bits correspondant à l'occupation de la mémoire par cinq processus.

Pour allouer k unités contiguës, le gestionnaire de mémoire doit parcourir la table de bits à la recherche de k zéro consécutifs. Cette méthode est rarement utilisée car la méthode de recherche est lente (k zéros consécutifs).

b) Utilisation des listes chaînée

Une autre solution consiste à chaîner les segments libres et occupés. La figure 4.4 (c) montre l'exemple d'un tel chaînage, les segments occupés par un processus sont marqués (P) les libres sont marqués (H). La liste est triée sur les adresses, ce qui facilite la mise à jour.

Lorsqu'on libère la mémoire occupée par un segment, il faut fusionner le segment libre avec le ou les segments adjacents libres s'ils existent (cf. figure 4.5).

FIG. 4.5 – Quatre combinaisons de voisins possibles d'un processus X qui termine et libère le segment qu'il occupe

4.3 Gestion avec recouvrement sans pagination

Dès que le nombre de processus devient supérieur au nombre de partitions, il faut pouvoir simuler la présence en mémoire centrale (MC) de tous les processus pour pouvoir satisfaire au principe d'équité et minimiser le temps de réponse des processus. La technique du recouvrement permet de stocker temporairement sur disque des images de processus afin de libérer de la MC pour d'autres processus.

On pourrait utiliser des partitions fixes, mais on utilise en pratique des partitions de taille variable, car le nombre, la taille et la position des processus peuvent varier dynamiquement au cours du temps. On n'est plus limité par des partitions trop grandes ou trop petites comme avec les partitions fixes. Cette amélioration de l'usage de la MC nécessite un mécanisme plus complexe d'allocation et de libération.

4.3.1 Le swapping (va et vient)

Dans un système monoprogrammé ou batch, l'allocation mémoire à un programme est définitive. En d'autres termes, le processus qui a reçu une partition la gardera jusqu'à sa terminaison.

A l'encontre, une alternative consiste à libérer temporairement des partitions occupées, afin de charger les programmes en attente dans le disque. On parle alors de méthode de swapping ou va-et-vient. Quel est le principe de cette méthode?

Le swapping consiste à gérer la mémoire centrale par allocation et préemption de l'allocation suivant l'état du processus, de la même manière qu'est géré le processeur central. Le processus qui a reçu une partition au départ

de son exécution peut sortir de la mémoire centrale vers le disque, suite à une décision du système, dans le but d'exécuter des processus en attente de libération de la ressource mémoire. Le processus stocké sur disque peut être remis plus tard dans une partition de la MC, pas nécessairement la même qu'auparavant.

Exemple :

Soit 4 programmes (jobs) à exécuter : A, B, C et D, stockés en MS. Voici un scénario de chargement/ swapping en MC.

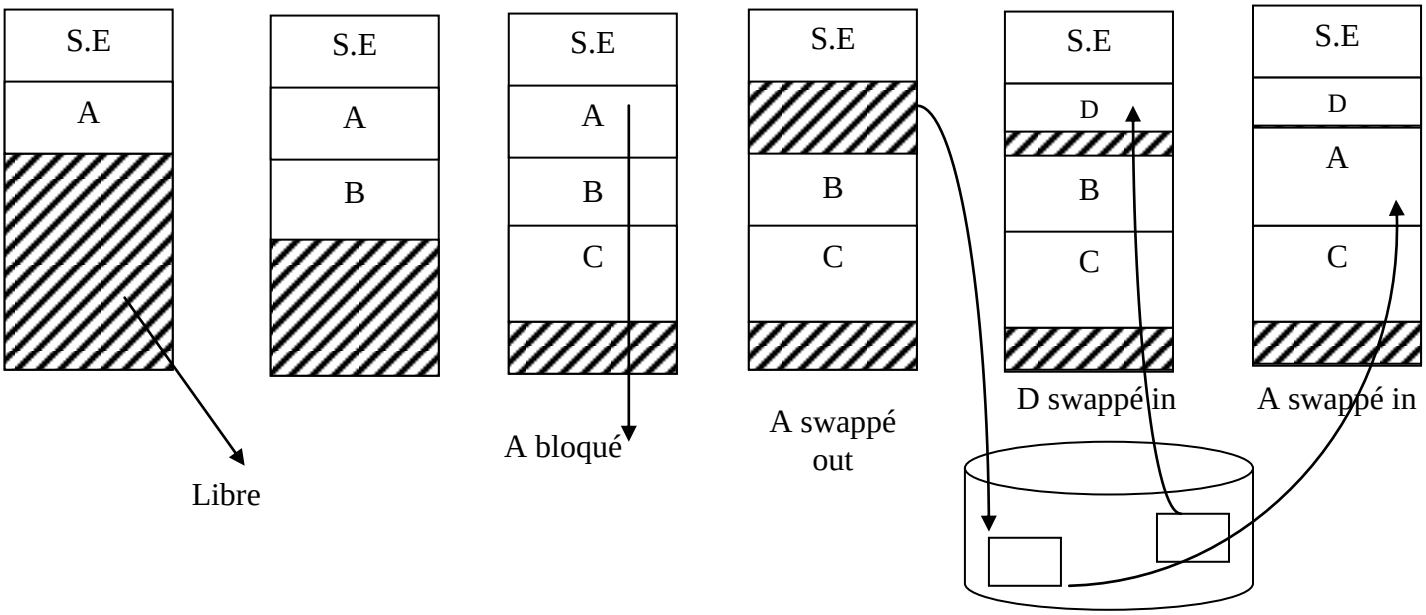


Figure 5- 6 : Gestion de la mémoire avec va et vient

4.4 Gestion avec recouvrement, avec pagination ou segmentation

4.4.1 Notion de mémoire virtuelle

Le principe de la mémoire virtuelle tend à étendre l'espace physique mémoire à un espace mémoire beaucoup plus grand, et ce, en utilisant une partie du disque (mémoire secondaire). Cette extension est réalisée par l'utilisation de :

- Une mémoire secondaire (disque dur) et
- Un système de gestion de la mémoire virtuelle.

Les programmes sont alors dispersés entre la mémoire physique et la mémoire secondaire.

Comme les programmes peuvent être plus grands que la mémoire physique, ceux-ci ne peuvent être stockés complètement en MC. Ils sont alors découpés en blocs, et ramené bloc par bloc de la mémoire secondaire en mémoire physique pour être exécutés. Ainsi, une partie des blocs est stockée en MC, et les autres blocs restent en mémoire secondaire.

D'autre part, avec cette organisation, les blocs en MC ne sont pas nécessairement stockés en zone contiguë. Ils sont dispersés en MC.

Il existe plusieurs techniques de réalisation de la mémoire virtuelle : La pagination, la segmentation et la segmentation paginée.

a) La pagination

La plupart des systèmes de mémoire virtuelle utilisent la technologie de pagination.

a-1) Définition

On parle de pagination lorsque :

- D'une part, la MC est organisée en blocs de tailles égales et fixes appelés pages physiques ou cadre de page. On dit que la mémoire est paginée.
- D'autre part, les processus sont divisés en blocs appelés pages virtuelles, ayant la même taille que les cadres de pages. On dit que les processus sont paginés.

Les processus sont mis en mémoire secondaire, en une zone appelée zone swap. Les adresses manipulés par ces processus swap. Les adresses manipulées par ces processus sont dites virtuelles, et l'on parlé espace d'adressage virtuel.

La pagination consiste donc à découper les deux espaces d'adressage (espace mémoire physique et espace virtuel) en pages de la même taille, et à mettre en oeuvre un mécanisme de transfert de pages entre la mémoire virtuelle et la mémoire réelle.

Comment se fait alors la correspondance entre une page virtuelle d'un processus et la page réelle physique en MC ?

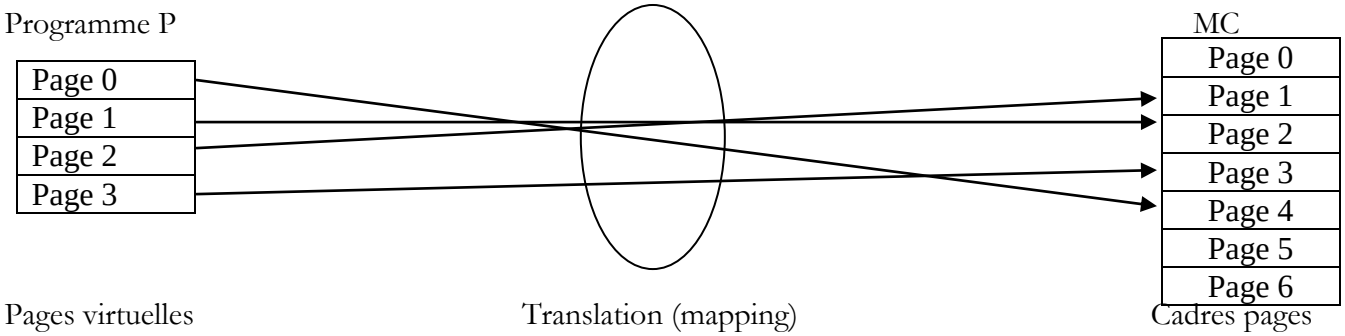
- **Correspondance entre l'adresse virtuelle référencée dans le programme et l'adresse réelle en MC**

Sur des machines sans mémoire virtuelle, l'adresse qui référencie un objet d'un programme est mise directement sur le bus d'adresse mémoire, et le mot mémoire physique est lu ou écrit.

Lorsque la mémoire virtuelle est utilisée, l'adresse d'un objet référencié dans un programme est logique virtuelle. Cette adresse n'est pas mise directement sur le bus d'adresse mémoire, mais elle est envoyée à une unité physique appelée unité de gestion mémoire (Memory Management Unit MMU), qui fait la correspondance entre l'adresse virtuelle et l'adresse physique correspondante.

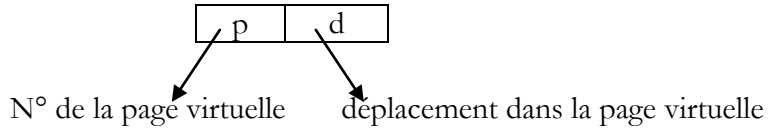
Les processeurs disposent actuellement d'un dispositif, le MMU "Memory Manager Unit" qui permet de placer des processus en mémoire sans nécessairement placer les pages de processus dans des cadres de pages contigus.

Comment se fait alors cette correspondance (mapping d'adresses) ?



- **Translation d'adresses (mapping)**

L'adresse virtuelle d'un mot dans une page virtuelle est structurée en un couple (p, d) comme suit :



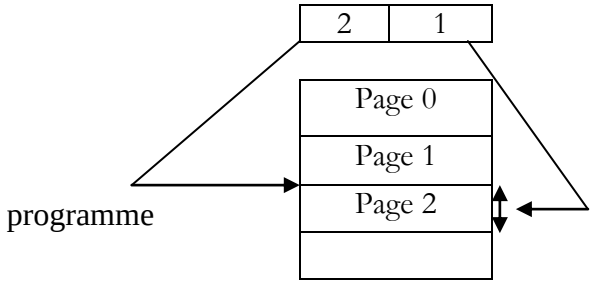
Les deux informations, numéro de page et déplacement dans le page, peuvent être calculées à partir de

l'adresse logique A du mot, en se basant sur la taille T de la page :

L'adresse A est égale à (p, d) telle que :

P= A/T où / est la division entière.
D= A modulo T.

Exemple :
Soit un mot d'adresse virtuelle 2001 =(2,1) dans un système paginé de taille 1000 octets.



Comment l'unité de gestion mémoire (MMU) met en correspondance les adresses physiques et virtuelles ?

- **Table des pages**

La table des pages sauvegarde des informations propres à chaque page virtuelle d'un processus. Elle fait correspondre à chaque page virtuelle un ensemble d'informations constituant ainsi un descripteur de la page virtuelle.
On peut citer essentiellement :

r	S	pr	m

- Un bit indicateur qui signale la présence ou l'absence de la page en mémoire centrale.

  r= { 0 → la page n'est pas en MC.
     { 1 → la page est en MC.
- Emplacement de la page en mémoire secondaire (notée s).
- Adresse mémoire de la page si la page a été chargée en MC (notée pr).
- Un bit de modification m qui indique si la page a été modifiée pendant l'exécution, et doit donc être recopiée en mémoire secondaire.

Il est possible d'avoir d'autres champs concernant une page (un ou plusieurs bits de protection qui spécifient les types d'accès autorisés à la page, …..etc).

L'adresse de la table des pages d'un processus est stockée dans un registre appelé base de la table des pages d'un processus (PTBR : Page Table Base Register). Ce registre est stocké dans son bloc de contrôle PCB.

La translation d'adresse se fait par *matériel* selon le mécanisme suivant :

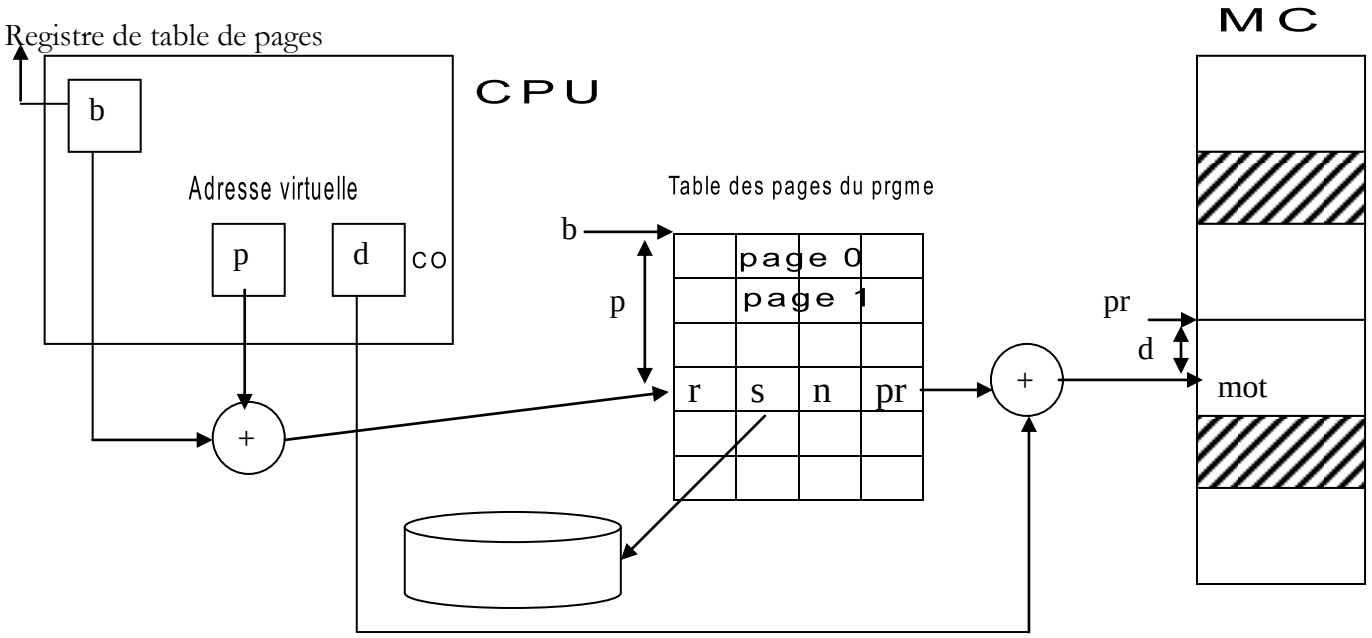
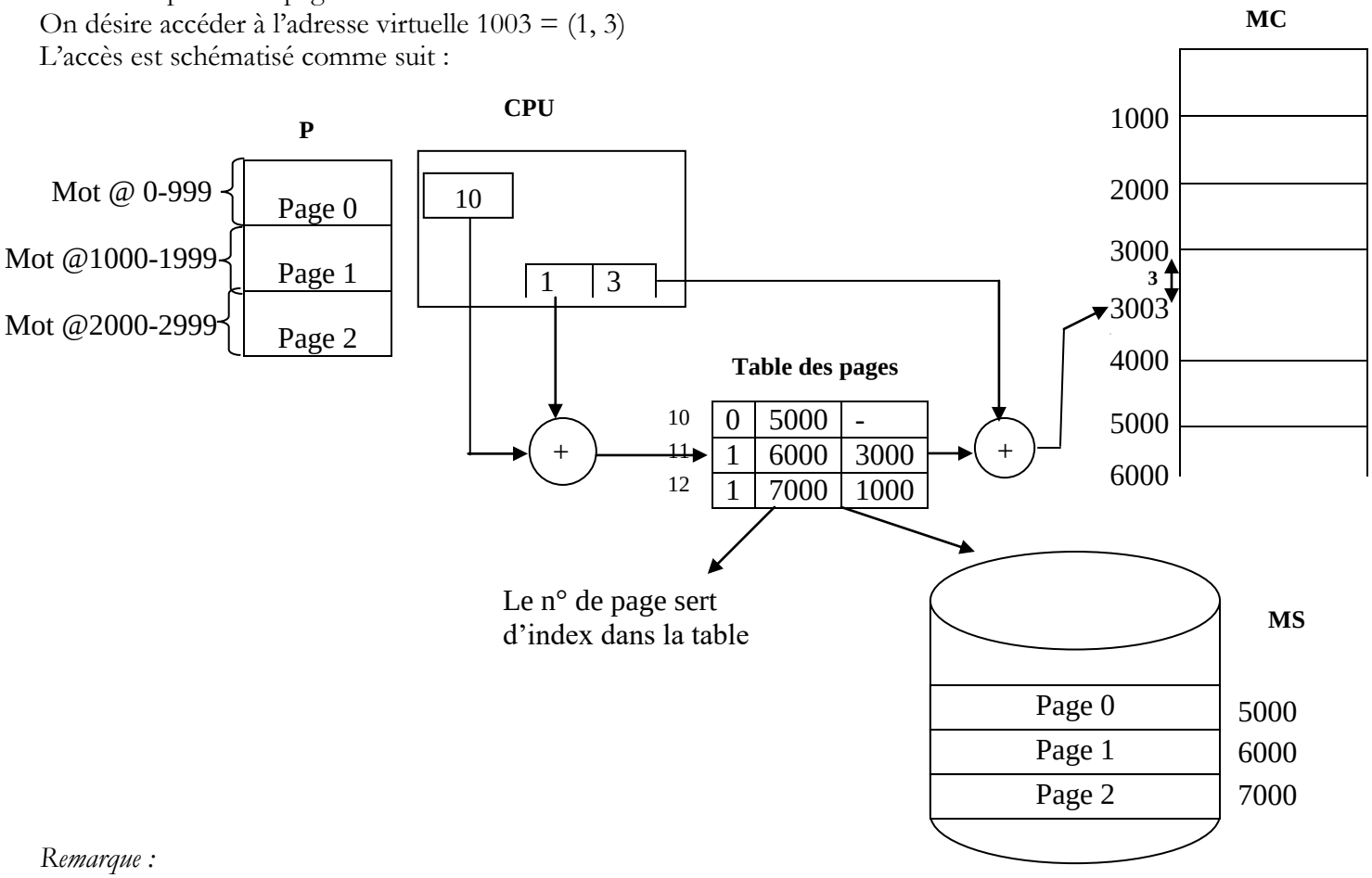


Figure 5- 7 : Mécanisme de translation d'adresse en pagination

Exemple :
Soit une MC de taille de 6000 octets, une page est de taille 1000 octets. Considérons un job P de taille 3000 octets. Il sera composé de 3 pages.
On désire accéder à l'adresse virtuelle 1003 = (1, 3)
L'accès est schématisé comme suit :



Remarque :
L'accès à une adresse d'un mot mémoire nécessite 2 accès mémoire : un accès pour la table des pages, et un autre pour l'emplacement lui-même.

Avantages :
-Allocation des pages simplifiée : les pages étant de tailles égales, aucun compactage n'est nécessaire.
-problème de fragmentation externe résolu

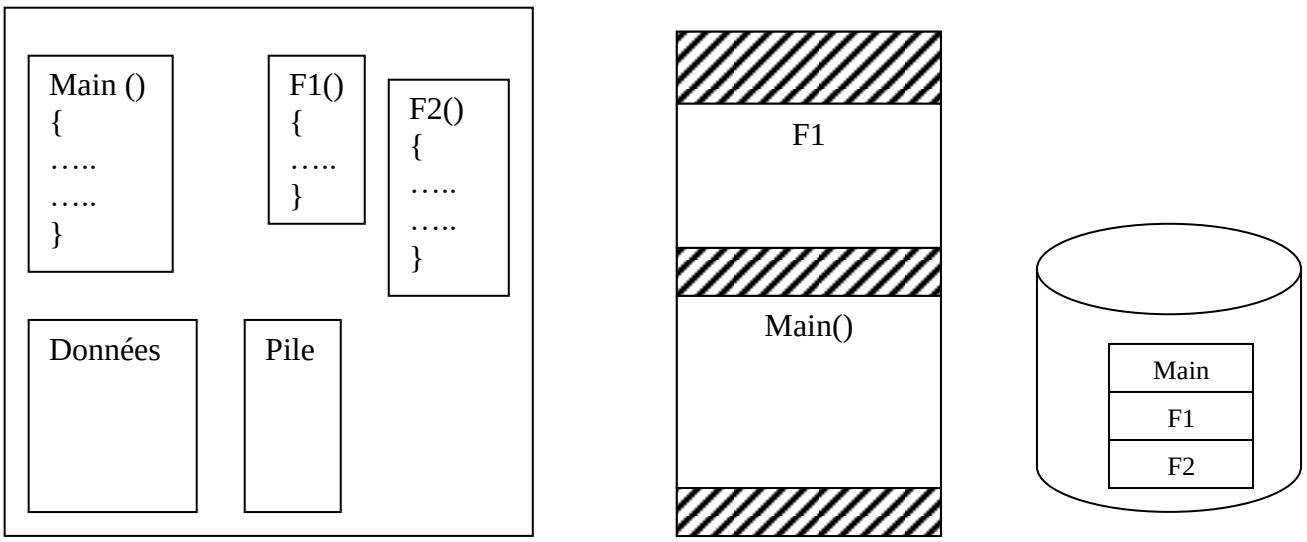
Inconvénients : possibilités de fragmentation interne.

b) La segmentation

Un programme est vue comme un ensemble d'unités logiques : programmes, sous-programmes et structures de données. Ce découpage fonctionnel n'est pas reflété par la pagination. La segmentation est une stratégie qui reproduit ce découpage logique.

Dans une mémoire segmentée, chaque unité logique est stockée dans bloc mémoire à part : un segment. Un segment est de taille variable.
Un programme est composé d'un ensemble de segments dispersés en MC et en MS.
Le découpage en segments est effectué automatiquement par le compilateur (segments : variables globales, pile d'exécution, code ….).
La translation d'adresse virtuelle est alors similaire à celle vue en pagination. Chaque processus détient une table des segments. Sauf, qu'un segment est aussi caractérisé par un nom ou numéro et une taille.
Une vérification de la validité d'adresse est effectuée à chaque référence en :

- vérifiant que le numéro du segment ne dépasse la longueur de la table des segments,
- et que le déplacement est inférieur à taille du segment.



Structure en segments d'un programme MC MS

Dans ce cas, chaque processus est associé une table des segments, qui contient pour chaque segment :

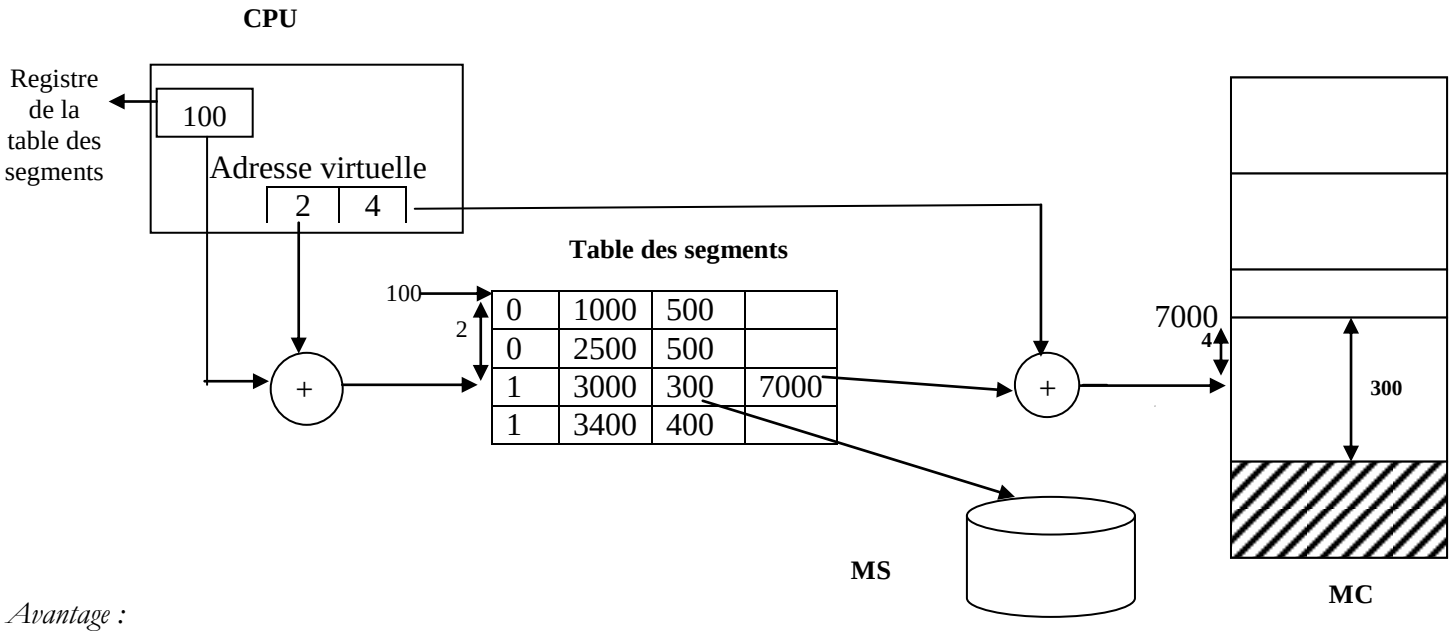
- un identificateur r de présence en MC.
- L'adresse SS du segment
- La taille T du segment
- L'adresse SR du segment en MC (si r=1)

Un registre appelé Registre de base de la table des segments (STBR) (Segment Table Base Register) contient adresse table de segment du processus avec un registre longueur de la table (SPLR) (Segment Table Length Register) contient nombre de segments du processus, sont utilisés pour l'accès à la table des segments (les 2 registres dans le MMU).

Exemple :
Soit un programme usager est constitué de 4 segments

S0	Main()	1500
S1	F1()	500
S2	F2()	300
S3	Data	400

On désire accéder à l'adresse virtuelle (2,3)



Avantage :

- Elimine la fragmentation interne

*Inconvénien*ts : fragmentation externe

c) La segmentation paginée

L'idée était de tirer profil des avantages de chacune des organisations précédentes. Un segment est alors découpé en pages de tailles fixes.
Un processus dispose sa propre table des segments et à chaque segment est associé une table des pages du segment.
Une adresse virtuelle est alors :

S : numéro segment
P : numéro page du segment
D : déplacement dans la page

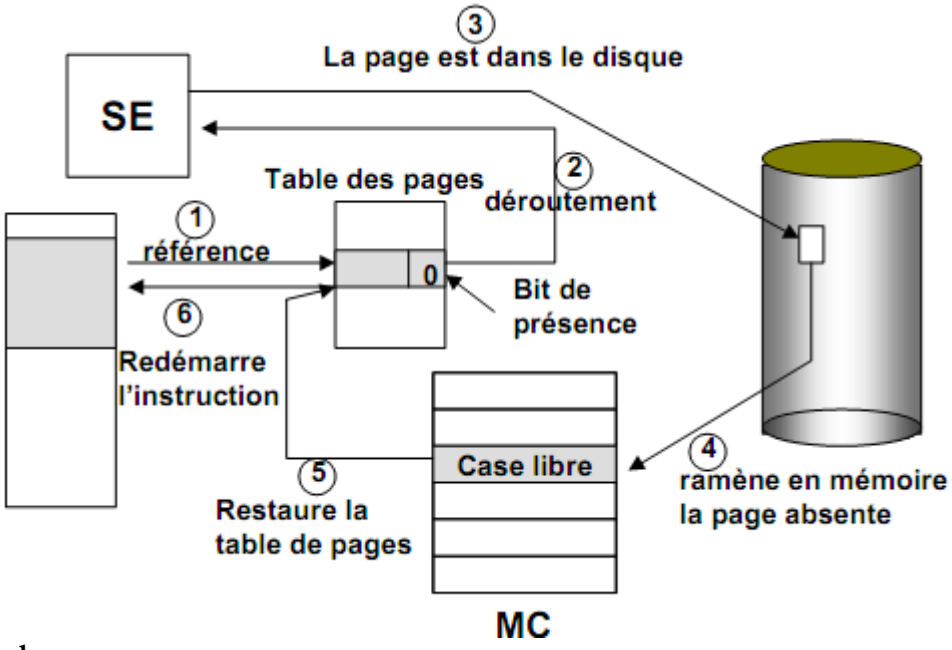
La table des segments décrit chaque segment par la taille du segment en nombres de pages et l'adresse de sa table des pages. La table des pages comporte les mêmes informations que dans le cas de la pagination.

4.5 Pagination à la demande :

La mémoire virtuelle est communément implémentée avec la pagination à la demande. Quand un processus doit être transféré en mémoire, la routine de pagination devine quelles pages seront utilisées avant que le processus soit mis en mémoire auxiliaire de nouveau. Au lieu de transférer en mémoire un processus complet, la routine de pagination ramène seulement les pages qui lui sont nécessaires. Ainsi, elle évite que l'on charge en mémoire des pages qui ne seront jamais employées, en réduisant le temps de swaping et la quantité de mémoire dont on a besoin.

Avec cette technique, le SE dispose de moyens pour distinguer les pages qui sont en mémoire, et celles qui sont sur disque. Il utilise dans la table des pages un bit valide/invalide pour décrire si la page est chargée en mémoire ou non.

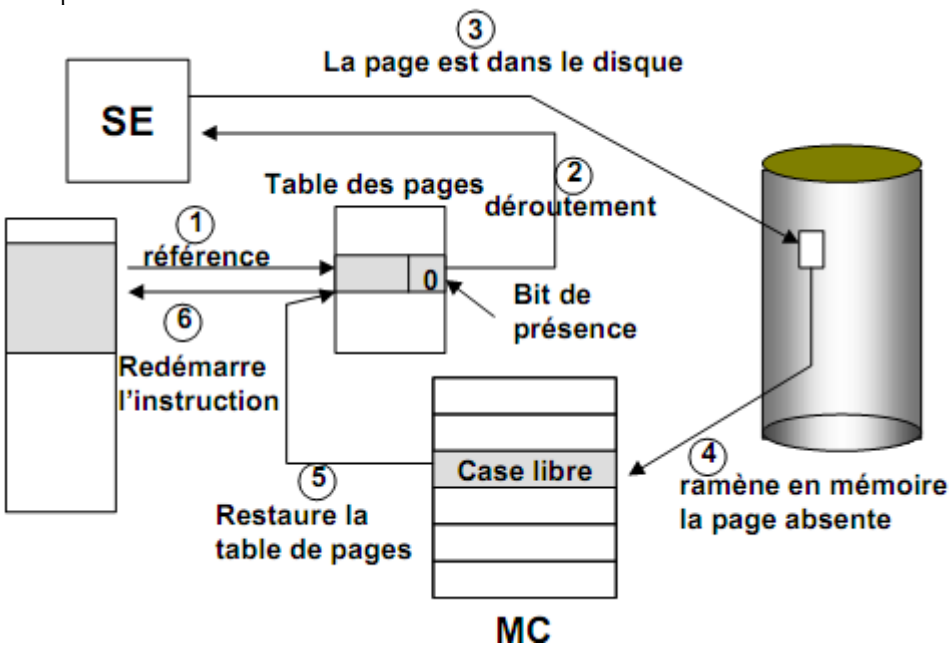
Que se passe-t-il si un processus essaie d'utiliser une page qui n'est pas en mémoire ?
L'accès à une page marquée invalidé provoque un défaut de page. En essayant d'accéder à cette page, il y a un déroutement vers le SE. La procédure permettant de traiter ce défaut de page est la suivante :
• S'assurer que la référence de la page est correcte.
• S'assurer que la page désirée est bien en mémoire auxiliaire.
• Trouver un cadre de page libre et charger la page.



4.1 Remplacement de pages

Lorsque le SE se rend compte au moment de charger une page qu'il n'existe aucun cadre de page disponible, il peut faire recours à un remplacement de page. Ainsi le code complet d'une procédure de traitement d'un défaut de pages est le suivant :

1. Trouver l'emplacement de la page désirée sur disque.
2. Trouver un cadre de page libre. S'il existe un cadre de pages libre, l'utiliser, sinon utiliser un algorithme de remplacement de pages pour sélectionner un cadre de page victime.
3. Enregistrer la page victime dans le disque et modifier la table de page.
4. Charger la page désirée dans le cadre de page récemment libéré et modifier la table de pages.
5. Redémarrer le processus utilisateur.

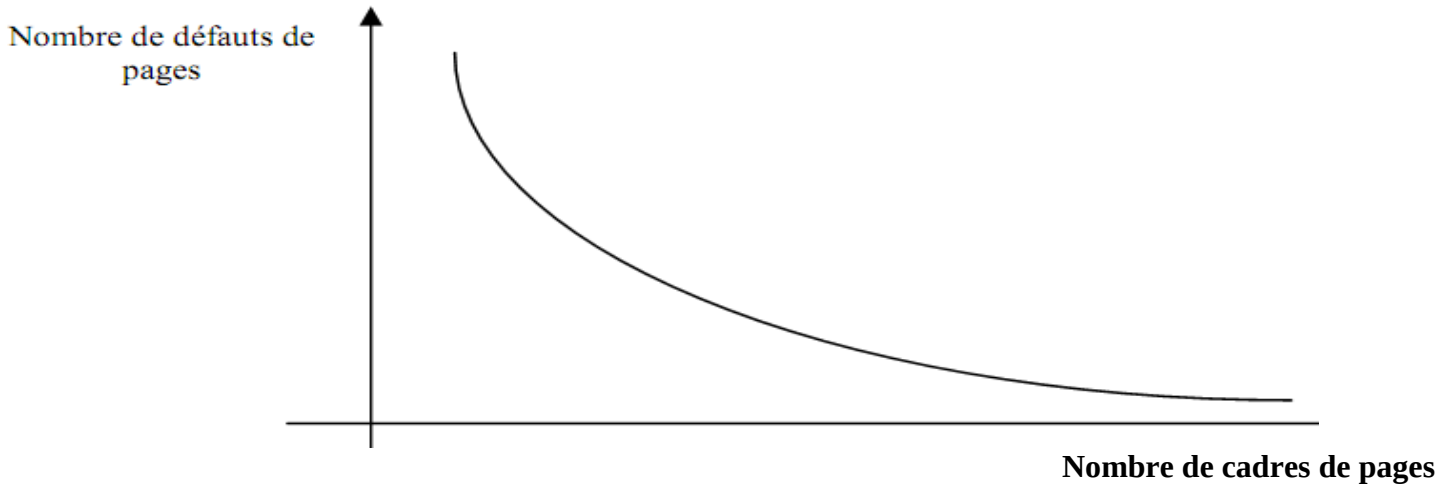


4.1.1 Algorithmes de remplacement de pages

Il existe plusieurs algorithmes différents de remplacement de pages. En général, on souhaite celui qui provoquera le taux de défauts de pages le plus bas.
On évalue un algorithme en l'exécutant sur une séquence particulière de références mémoires et en calculant le nombre de défauts de page. Cette chaîne est appelée : chaîne de références.

Afin de déterminer le nombre de défauts de pages pour une chaîne de références et un algorithme de remplacement particulier, on doit également connaître le nombre de cadres de pages disponibles.

Evidemment, au fur et à mesure que le nombre de cadres de pages augmente, le nombre de défauts de pages doit diminuer, comme le montre la figure suivante :



a) Algorithme FIFO

L'algorithme de remplacement FIFO (First In First Out) est le plus simple à réaliser. Avec cet algorithme, quand on doit remplacer une page, c'est la plus ancienne qu'on doit sélectionner.

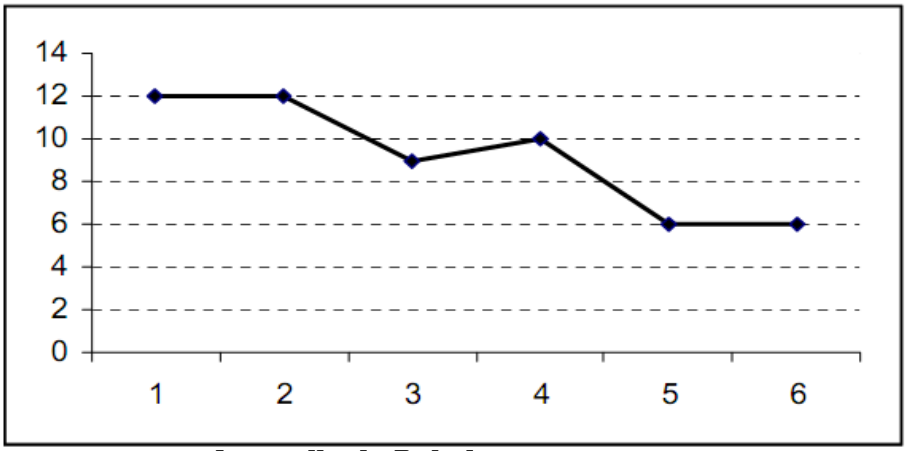
Exemple : Considérons un système à mémoire paginée ayant 3 cadres de pages (pages physiques), et soit la chaîne de références suivante : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Quel est le nombre de défauts de pages si l'algorithme de remplacement choisi est l'algorithme FIFO ?.
Pour répondre à cette question, faisons un déroulement qui fait ressortir les états successifs du système et en marquant les différents défauts de pages.

Chaîne de référence →	7	0	1	2	0	3	0	4	2	3
Cadres de page	7	7	7	2	2	2	2	4	4	4
		0	0	0	0	3	3	3	2	2
			1	1	1	1	0	0	0	3
Défauts de page →	X	X	X	X		X	X	X	X	X

	0	3	2	1	2	0	1	7	0	1
	0	0	0	0	0	0	0	7	7	7
	2	2	2	1	1	1	1	1	0	0
	3	3	3	3	2	2	2	2	2	1
	X			X	X			X	X	X

Nombre de défauts de pages : 15.

Afin d'illustrer les problèmes rencontrés avec un algorithme de remplacement FIFO, on peut envisager la chaîne de références suivantes : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. La figure suivante montre le nombre de défauts de pages en fonction des cadres de pages disponibles.



Anomalie de Belady

Cadres de pages	Défauts de pages
1	12
2	12
3	9
4	10
5	6
6	6

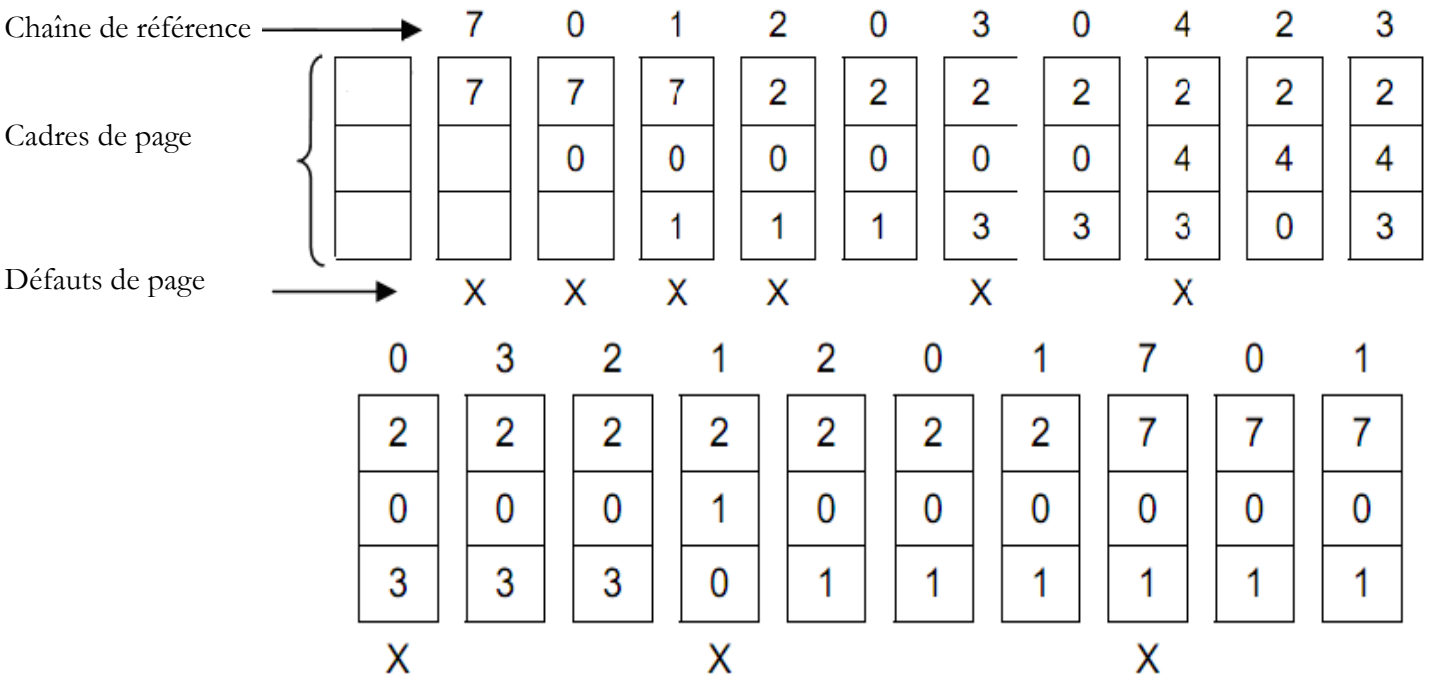
On remarque que le nombre de défauts de pages pour 4 cadres de pages (ie : 10) est supérieur au nombre de défauts de pages pour 3 cadres de pages (ie : 9). Ce résultat fort inattendu est connu sous le nom de l'anomalie de Belady. Elle décrit une situation où le taux de défauts de pages augmente au fur et à mesure que le nombre de cadres de pages augmente, contrairement à la règle générale.

L'algorithme de remplacement FIFO est simple à implémenter mais ses performances ne sont pas toujours bonnes.

b) Algorithme optimal

L'algorithme de remplacement optimal fournit le taux de défauts de pages le plus bas de tous les autres algorithmes, de plus il ne souffre pas de l'anomalie de Belady. Son principe est de remplacer la page qui mettra le plus de temps à être de nouveau utilisée.

Exemple : Reprenons la même chaîne de références que l'algorithme FIFO précédent, et calculons le nombre de défauts de pages avec cet algorithme.



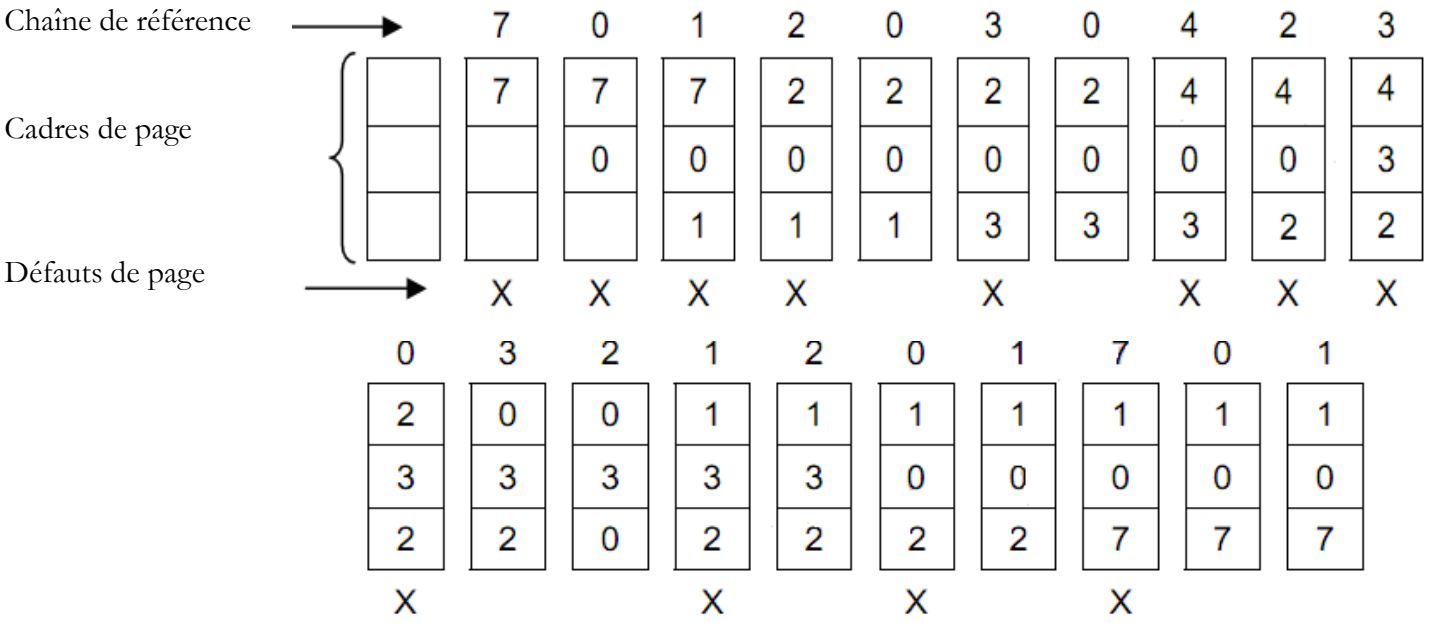
Nombre de défauts de pages : 09.

Malheureusement, l'algorithme optimal est difficile à mettre en œuvre car il requiert une connaissance future de la chaîne de références. Il est utilisé essentiellement pour faire des études comparatives.

c) Algorithme LRU

L'algorithme LRU (Least Recently Used) sélectionne pour le remplacement d'une page victime, la page la moins récemment utilisée. C'est à dire qu'on doit remplacer une page, l'algorithme sélectionne la page qui n'a pas été utilisée pendant la plus grande période de temps.

Exemple : Reprenons la même chaîne de références que l'algorithme FIFO précédent, et calculons le nombre de défauts de pages avec cet algorithme.



Nombre de défauts de pages : 12

d) L'algorithme de remplacement LFU (Least Frequently Used) «la moins souvent utilisée)

On garde un compteur qui est incrémenté à chaque fois que le cadre est référencé, et la victime sera le cadre dont le compteur est le plus bas (la page la moins fréquemment utilisée).

Exemple : En reprenant la même chaîne de références que l'algorithme FIFO précédent, le nombre de défauts de pages trouvé avec cet algorithme est de 11.

Cet algorithme présente un problème quand une page est très utilisée pendant la phase initiale d'un programme, mais qu'elle n'est plus employée à nouveau par la suite. Comme elle a été amplement utilisée, elle possède un grand compte et reste donc en mémoire même si elle n'est pas utilisée.