Chapitre IV : Gestion de la mémoire centrale

1. Introduction :

La mémoire est une ressource d stockage de données et de programmes. Elle est subdivisée en une suite finie de composants appelés emplacements. Un programme ne peut s'exécuter plusieurs que si ses instructions et ses données (au moins partiellement) sont en mémoire centrale. Si l'on désire exécuter plusieurs programmes simultanément, il faut que chacun soit chargé dans la mémoire.

2. Objectifs d'un gestionnaire de la mémoire

Le gestionnaire de la mémoire est un module du système d'exploitation (programme) dont le rôle est la gestion de la mémoire principale. Il doit garder en permanence des informations concernant l'espace libre ou occupé de la mémoire. Le gestionnaire alloue de la mémoire aux processus qui en ont besoin et récupère l'espace mémoire libéré par un processus. La mémoire principal étant limitée, le gestionnaire ne peut pas charger tous les processus en mémoire centrale. Pour satisfaire les besoins des processus, le gestionnaire doit décharger certains programmes (*les programmes déchargés sont sauvegardés sur disque*) de la mémoire principale et charger d'autres programmes à partir du disque.

En résumé, le gestionnaire de la mémoire doit viser les objectifs suivants :

① **La réallocation** : Le système d'exploitation alloue à chaque programme une zone mémoire. Mais tous les programmes n'ont pas la même taille. De plus, les programmes sont lancés par les utilisateurs, puis se terminent à des moments que le système ne les connaît pas à l'avance. Chaque fois qu'un utilisateur demande le lancement d'un programme, le système doit trouver une place dans la mémoire pour le charger : Il y a réorganisation des programmes en mémoire.

② **La protection** : La coexistence de plusieurs processus en mémoire centrale nécessite la protection de chaque espace mémoire vis-à-vis des autres. Par conséquent, un processus ne peut accéder à l'espace d'un autre processus que s'il est autorisé.

③ **Le partage** : Parfois, il est utile de partager un espace mémoire (programmes ou des données) entre plusieurs processus. Ainsi, le gestionnaire de la mémoire doit autoriser des accès contrôlés sans compromettre la protection

Exemple : un éditeur de texte partagé entre plusieurs utilisateurs d'un système à temps partagé.

3. Fonctions

Un gestionnaire de la mémoire doit assurer les fonctions suivantes :

① Garder trace du statut de chaque portion de la mémoire (Libre/allouée).

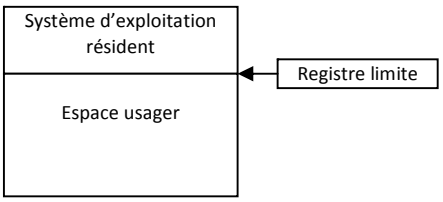
② Déterminer une politique d'allocation de la mémoire qui permet de décider qui doit avoir de la mémoire, combien et pour combien de temps ?

③ Déterminer une technique d'allocation qui permet d'allouer de l'espace à un processus.

④ Déterminer une politique de libération de la mémoire.

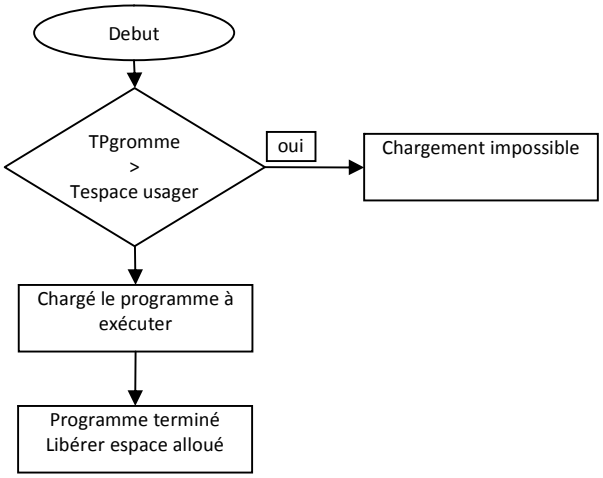
4. Stratégies d'allocation de la mémoire

4.1. Une seule zone contiguë *(Single continuous allocation)* : Dans ce cas, la mémoire est divisée en deux zones contiguës. L'une est allouée en permanence à la partie résidente du système d'exploitation. La deuxième zone permettra de reloger un processus usager ou du système non résident. Cette stratégie a été utilisée par les micro-ordinateurs sous CP/M et PC/DOS.



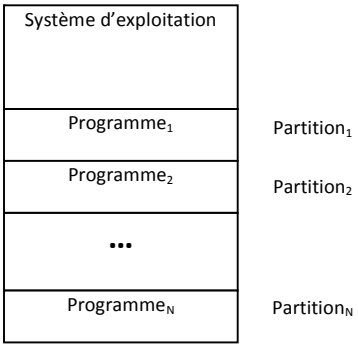
La gestion de la mémoire s'avère simple : le système d'exploitation doit garder trace de deux zones mémoires.

L'algorithme d'allocation peut être décrit par l'organigramme suivant :



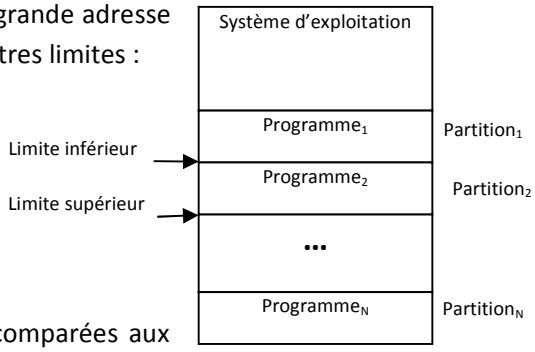
4.2. Partition multiples : Cette stratégie constitue une technique simple pour la mise en œuvre de la multi-programmation. La mémoire est alors partagée en régions séparées ou partitions mémoires. Chaque partition dispose son espace d'adressage. Le partitionnement de la mémoire peut être statique ou dynamique.

4.3.1. Partitions multiples statiques (*fixes*): Dans cette technique, la mémoire centrale est divisée en partitions (zones) éventuellement de tailles fixes. Les tailles des partitions sont fixées une fois pour tout au moment du chargement du système. La fixation des tailles peut être faite manuellement par l'opérateur (par le biais d'une commande), ou automatiquement par le système d'exploitation. Pendant son exécution, un programme ne doit pas accéder en dehors de la partition qui lui est allouée.

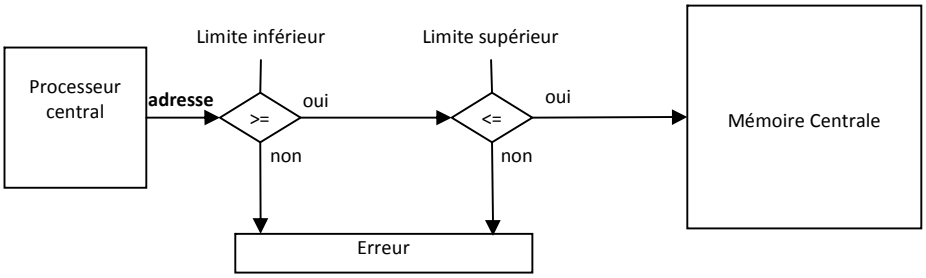


La mise en œuvre de cette technique nécessite un dispositif matériel pour assurer la protection des différents processus présents en mémoire. Deux mécanismes peuvent être utilisés pour assurer cette protection : les registres limites et la clé de protection.

- **Registres limites** : La plus petite et la plus grande adresse de la partition sont chargées dans des registres limites :

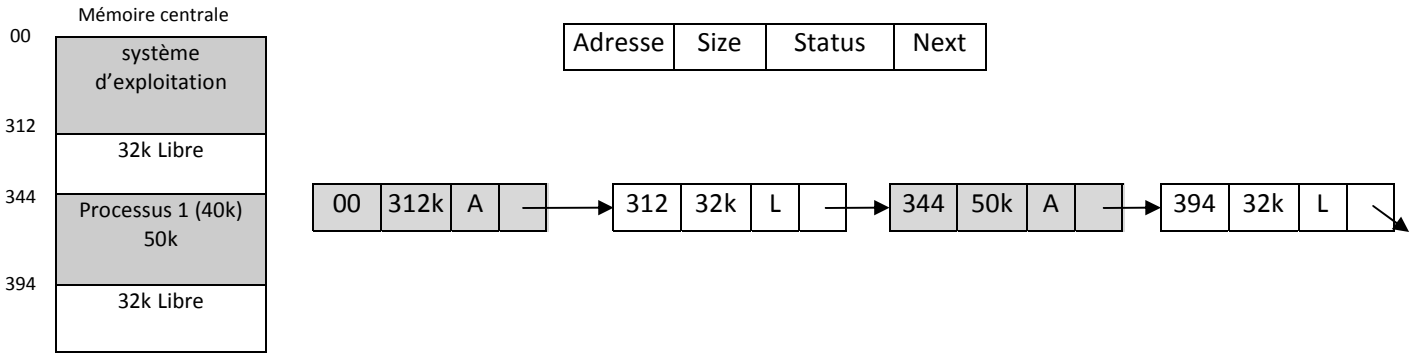


Toutes les adresses des utilisateurs sont comparées aux registres limites.



- **Clé de protection** : Le principe de ce mécanisme est d'associer à chaque partition une clé (4 bits par exemple). Le mot d'état du processus (PSW) est possède un masque de 4 bits contenant une copie de la clé de la partition où se trouve loger le processus. A chaque référence mémoire le hardware compare la clé de la partition accédée à celle du PSW. En cas de violation (clé partition_accedee <> Clé du PSW) un déroutement sera déclenché.
- **Algorithmes et structures des données :** La structure de données utilisée avec ce type de mémoire est une Liste où chaque élément correspond à une partition. La structure d'un élément prend la forme ci-dessous :

Exemple :



⌦ **Chargement des programmes** : Tout travail doit faire une demande explicite de l'espace mémoire nécessaire à l'exécution des programmes qui le composent. Il existe deux types de chargement :

- *Une file d'attente par partition* : chaque travail est inséré dans la file de la partition dont la taille est plus proche (supérieur ou égale) de la taille demandée. L'inconvénient de ce type de chargement est que, un moment donné, on peut trouver des files vides alors que d'autres files contiennent beaucoup de travaux en attente. L'espace inutilisé dans une partition est perdu : c'est la fragmentation interne.
- Une autre approche consiste à mettre tous les travaux dans la même file. Dans ce cas, on alloue au travail choisi la première partition dont la taille est supèrieur ou égale à la taille demandée.

Avantages : Le principal avantage de cette technique est la simplicité des algorithmes (d'allocation et de libération de la mémoire), ce qui conduit à une implémentation facile.

Inconvénients : Gaspillage considérable de l'espace mémoire causé par la différence entre la taille du processus (ou en général la taille demandée) , et celle de la partition correspondante.

4.3.2. Partitions multiples variables *(dynamique)*: Le problème avec la technique des partitions fixes est de déterminer les tailles des partitions qui minimisent la **fragmentation interne** de la mémoire principale. La solution à ce problème consiste à utiliser des partitions dont les tailles varient dynamiquement slon la demande. Une partition est définie par sa taille et son adresse.

① ***Allocation d'une partition*** : L'allocation d'une partition de taille **t** à un processus, consiste à trouver une *zone de taille convenable* parmi les zones (partitions) libres. La taille de la partition choisie doit être supérieure ou égale à la taille **t** demandée par le processus. Si la partition sélectionnée a une taille trop grande, elle est découpée en deux : Une partie de taille **t** est alloué au processus ; l'autre partie est insérée dans « la liste » des partitions libres (c'est ce que l'on appelle un éclatement de partition).

② ***Libération d'une partition*** : Quand un processus termine, il libère sa partition qu'est de nouveau placée dans la liste des zones libres. La libération d'une partition peut crée trois situations différentes selon que la partition est entourée de :

- Deux partitions libres :

- Une partition libre est d'une partition occupée :

- Deux partitions occupées :

Chaque fois que cela est possible, il est utile de regrouper (fusionner) les partitions contiguës dés la libération. Cette fusion consiste à créer, à partir de plusieurs partitions contiguës, une seule partition de grande taille.

③ ***Choix d'une partition*** : Le principal problème de cette technique d'allocation est le choix d'une partition libre de taille convenable. Il existe plusieurs stratégies de sélection d'une partition ; les plus importantes sont :

- *First-Fit* : Allouer la première partition dont la taille est suffisamment grande. La liste est ordonnée selon les adresses croissantes.
- *Best-Fit* : Allouer la partition dont la taille est la plus proche de la taille demandée. La liste est ordonnée (ordre croissant) selon les tailles des partitions.
- *Worst-Fit* : Allouer la plus grande zone libre. La liste est ordonnée (ordre décroissant) selon les tailles des partitions.

Avantages :

- Meilleur degré de multiprogrammation à cause du nombre important de processus présent en mémoire (relativement bien sûr à la technique précédente).
- Les algorithmes utilisés sont simples et faciles à implémenter.

I**nconvénient** : L'inconvénient principal de cette technique étant la présence du phénomène de fragmentation externe : au fur et à mesure que les processus entrent et sortent du système des trous seront crées dans la mémoire (solution n'est pas recommandée : la technique de compactage car, elle nécessite beaucoup de temps processeur).

4.3. Mémoire virtuelle : C'est une technique qui permet l'exécution d'un processus partiellement chargé en mémoire. Ce qui permet d'exécuter un processus dont la taille est nettement supérieur à la taille de la mémoire physique.

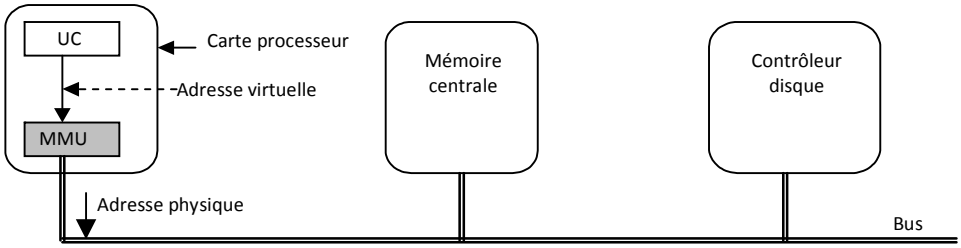
4.3.1. La pagination : L'idée directrice de la pagination consiste à découper l'espace virtuel en pages, l'espace physique est également découpé en blocs. Les blocs et les pages sont de même taille (4 kbytes cas d'Intel). Les pages des processus (espaces virtuels), qui ne sont pas chargées dans des cases de mémoire physiques sont stockées en mémoire secondaire (disque). Notons que certains systèmes chargent toutes les pages du programme en mémoire secondaire (partition disque ***swap***).

4.3.1.1. Espace d'adressage logique et espace d'adressage physique : On appelle **espace d'adressage logique** (*virtuel*) d'un processus l'ensemble de toutes les adresses logiques (virtuelles) générées au cours de l'exécution de ce processus.

L'ensemble de toutes les adresses physiques (réelles) correspondant à ces adresses est appelé **espace d'adressage physique**.

NB : Sur une machine sans mémoire virtuelle, alors il n'ya pas de différence entre adresse virtuelle et physique. Lorsque la mémoire virtuelle est utilisée, les adresses virtuelles ne sont pas directement placées sur le bus de la mémoire. Elles sont envoyées à l'unité de gestion mémoire (MMU) ; ce composant matériel traduit les adresses virtuelles en adresses physiques.

4.3.1.2. Support matériel : Le support matériel nécessaire à la pagination est une unité de gestion de la mémoire MMU (Memory Management Unit).



La vocation de ce circuit (MMU) est de traduire une adresse virtuelle en une adresse physique en utilisant une table dite table des pages. Cette table peut être stockée dans des registres, une mémoire associative ou en mémoire centrale.

4.3.1.3. Traduction des adresses virtuelles en adresses physiques :

Il est à rappeler que la traduction d'une adresse virtuelle en adresse physique est réalisée par un dispositif matériel appelé «unité de gestion mémoire».

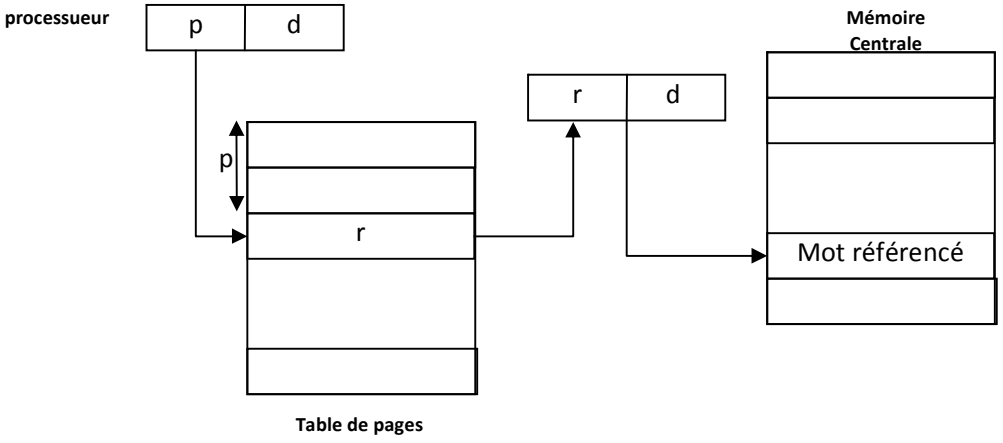
a. ***Adresse virtuelle*** : Dans un système à mémoire paginée, l'adresse est composée de :

- Un numéro de page virtuelle (**p** bits de pois forts : gauche),
- Un déplacement à l'intérieur de la page (**d** bits de poids faibles : droite),
- Taille de la page **= 2^d**.

Adresse virtuelle	Page : **p** bits	Déplacement : **d** bits

b. ***Table de pages :*** Pour établir la correspondance entre page virtuelle et page physique (ou case) on utilise une table que l'on appelle *table de pages*. Cette table a **p** entrées chacune de ces entrées contient le numéro de case (adresse physique) de la page correspondante (et d'autres informations que l'on détaillera ultérieurement). Le numéro de page **p** sert d'index dans la table de pages.

c. ***Calcul de l'adresse physique*** : L'adresse physique notée **[r, d]** d'un mot (octet) d'adresse virtuelle noté **[p, d]** est obtenu en remplaçant le numéro de page p par le numéro de case r trouvé dans la $p^{ème}$ entrée de la table.



Chaque processus a sa propre table de pages. L'adresse de cette table fait partie du PCB du processus (descripteur du processus). A chaque commutation de contexte, la table de pages de la machine est chargée avec la table du processus élu (processus choisi par l'ordonnanceur du processus). Si la table de pages est en mémoire centrale, l'adresse de la table de pages courante est chargée dans un registre (pointeur).

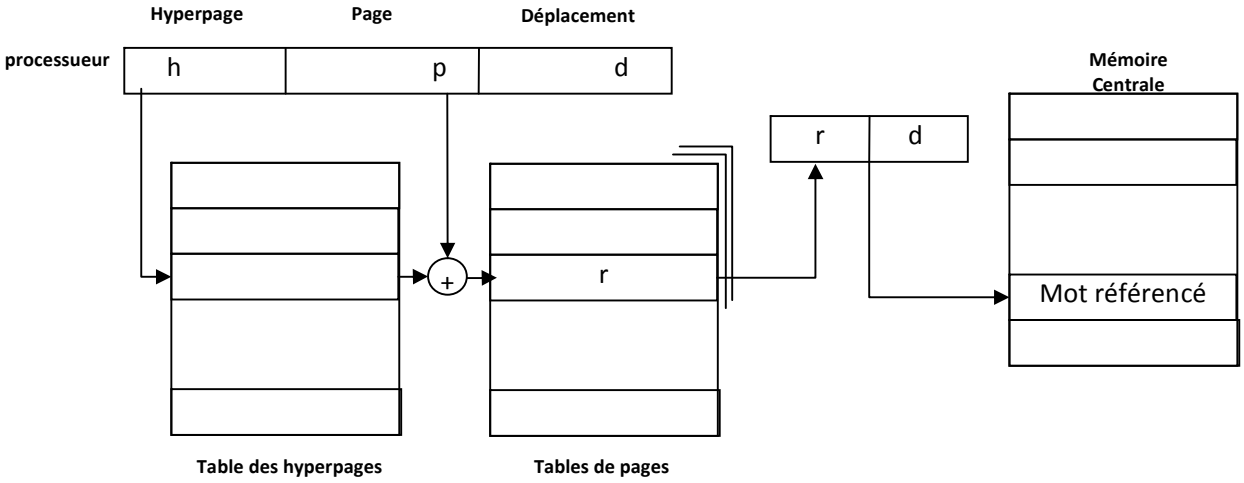
Exemple :

4.3.1.4. Implantation de la table de pages : L'implantation de la table de pages peut être réalisée par un ensemble de registres dédiés. Ce procédé permet d'accélérer le temps de translation d'adresses, il reste réaliste quand le nombre de pages adressables est petit.

***Exemple* (Le DEC-PDP/11)**: l'adresse est constituée de 16 bits et la taille de la page est de 8192 octets (8ko). La table de pages est constituée de 8 registres (8 pages de 8ko) Quand le nombre de pages est assez élevé (IBM 370 : 4096 pages), l'utilisation des registres est non réalisable. Dans ce cas, la table de pages est maintenue dans la mémoire centrale. Un registre base de la table des pages (Page Table Base Register : PTBR) pointe vers la table de pages. Cette technique nécessite deux accès mémoires pour atteindre un mot en mémoire centrale (Un accès à la table de pages et un accès au mot référencé).

4.3.1.5. La pagination à deux niveaux : La plupart des SE modernes supportent un espace adresse logique très grand (2^{32} à 2^{64}). Dans un tel environnement, la table de pages elle-même devient excessivement grande. Par exemple, si l'espace adresse est de 2^{32}, et si la taille d'une page est de 4Ko (2^{12}), alors la table de pages peut posséder jusqu'à 1 million d'entrée ($2^{32}/2^{12}$). Il convient alors de subdiviser la table de pages en parties plus petites. Par exemple, on peut utiliser un schéma de pagination à 2 niveaux, dans lequel la table de pages elle-même est paginée.

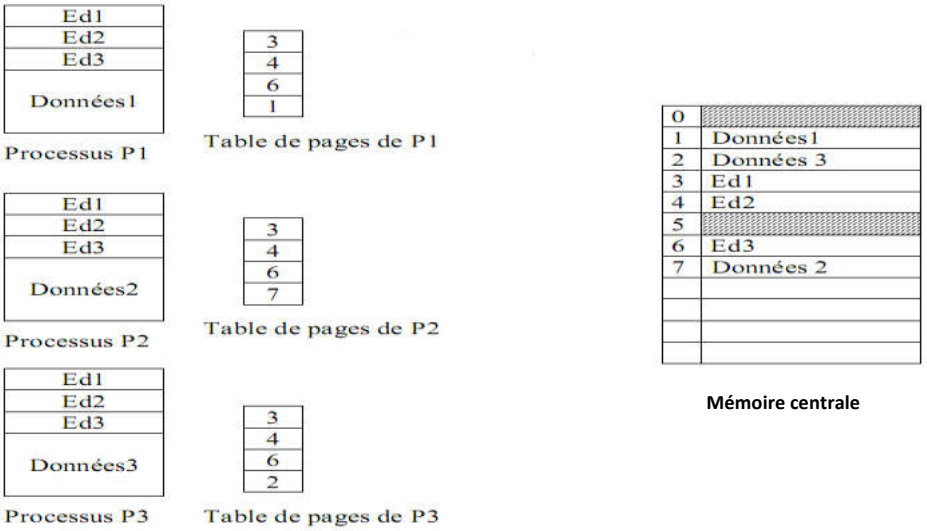
- **Traduction des adresses virtuelles en dresses physiques** :



Trois accès sont nécessaires pour atteindre un mot en mémoire centrale (un accès à la table des hyperpages, un accès à la table de pages et un accès au mot référencé).

NB : *La table des hyperpages et les tables de pages sont stockées en mémoire centrale.*

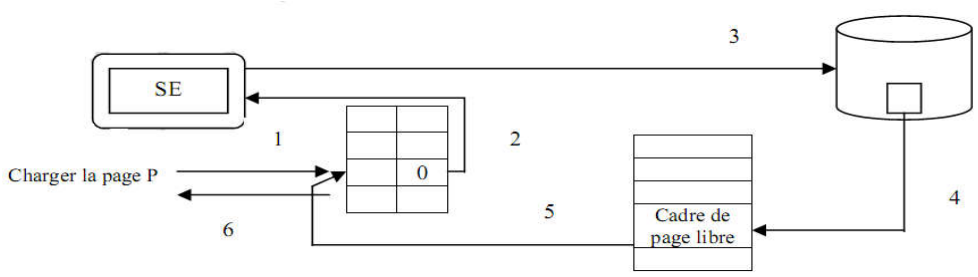
4.3.1.6. Partage du code et des données (partage de pages) : Un autre avantage de la pagination est la possibilité de partager du code commun. Cette caractéristique est particulièrement importante dans un environnement partagé. Imaginons un système qui supporte 40 utilisateurs, chacun d'eux exécutant un éditeur de textes. Si l'éditeur est constitué de 150 Ko de code et 50 Ko de données, on aura besoin de 8000 Ko pour satisfaire les 40 utilisateurs. Cependant si le code est réentrant, il peut être partagé comme le montre le schéma suivant :



4.3.1.7. Pagination à la demande : 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 doit disposer de moyens pour distinguer les pages qui sont en mémoire, et celles qui sont sur disque. Par exemple, on peut utiliser dans la table de 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 (case vide) et charger la page.



1: on fait référence à la page P qui n'existe pas en mémoire
2: Il y a un déroutement vers le SE
3: Le SE vérifie si la page existe bien en mémoire auxiliaire
4: Le SE vérifie s'il y a un cadre de page libre, auquel cas il y charge la page absente
5: Le SE met à jour la table de page
6: Redémarrer l'instruction du processus qui a provoqué le défaut de page.

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.

- **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.

① **Algorithme FIFO** : L'algorithme de remplacement FIFO 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 cases (pages physiques), et soit la chaîne de références suivante : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3. Quel est le nombre de défauts de pages si l'algorithme de remplacement choisi est l'algorithme FIFO ?.

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

Nombre de défauts de pages : 9.

② **Algorithme LRU (*Least Recently Used*)**: L'algorithme LRU (Moins Récemment Utilisée) consiste en cas de défaut de page à remplacer la page qui est restée inutilisée pendant le plus de temps.

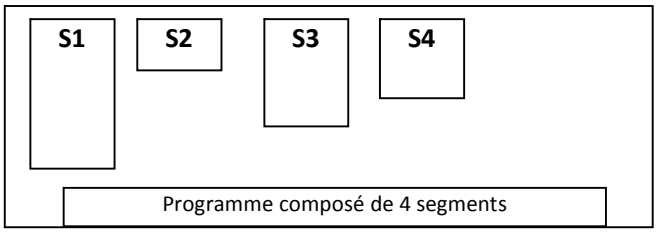
***Exemple* :** Reprenons la même chaîne de références que l'algorithme FIFO précédent.

Chaîne de référence →	7	0	1	2	0	3	0	4	2	3
Cases (pages)	7	7	7	2	2	2	2	4	4	4
		0	0	0	0	0	0	0	0	3
			1	1	1	3	3	3	2	2
Défauts de page →	X	X	X	X		X		X		X

Nombre de défauts de pages : 7.

4.3.2. Segmentation : Jusqu'ici on a considéré l'espace d'adressage virtuel comme un espace linèaire, ayant des adresses comprises entre 0 et n-1. L'utilisateur voit un programme comme un ensemble de modules de tailles variables. Un module peut être une *procédure* ou un ensemble de données. Chaque module est défini par un **nom**.

La segmentation est une technique de gestion mémoire permettant de supporter cette vue de l'utilisateur. L'espace virtuel est constitué d'un ensemble de **segments**.



Un segment correspond à un module du programme (ex : *fonction* ou *procédure*). Chaque segment est défini par un **nom** et une **longueur** (taille).

Une adresse virtuelle est composée du **nom du segment** et d'un **déplacement** à l'intérieur du segment.

Adresse virtuelle	Nom de segment	Déplacement

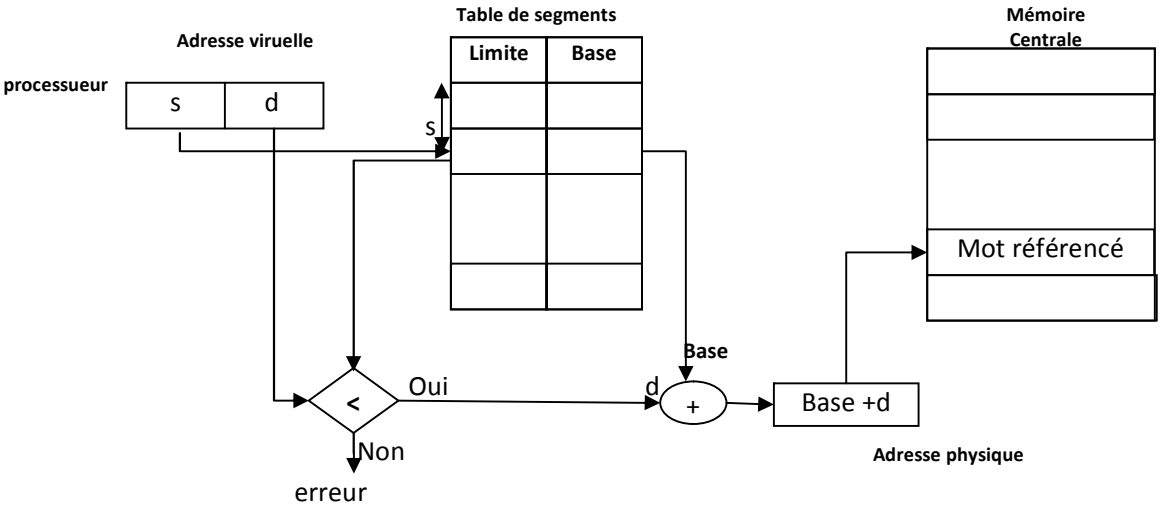
Remarque : Pour simplifier l'implémentation, les segments sont numérotés et référencés par un numéro plutôt que par son nom.

4.3.2.1. Traduction des adresses virtuelles en dresses physiques : L'espace virtuel d'un processus est décrit à l'aide d'une table de segments ; chaque entrée contient :

- La base ou l'adresse début du segment en mémoire centrale
- La limite ou la longueur du segment.

La table contient autant d'entrées que de segments réellement utilisés.

Le bloc de contrôle de processus (PCB) contient un pointeur vers la table de segment.



4.3.2.2. Implémentation de la table de segment : La table de segments peut être chargée dans un ensemble de registres rapides (**base**, **limite**) ou en mémoire centrale. Dans ce dernier cas, deux accès sont nécessaires pour atteindre le mot référencé.