Université Mohamed Kheider Biskra
Faculté des sciences exactes et sciences de la nature et de la vie
Département d'informatique

Support de cours

Module : Systèmes d'exploitation I

Chapitre 2 : Gestion de la mémoire

(Partie 02)

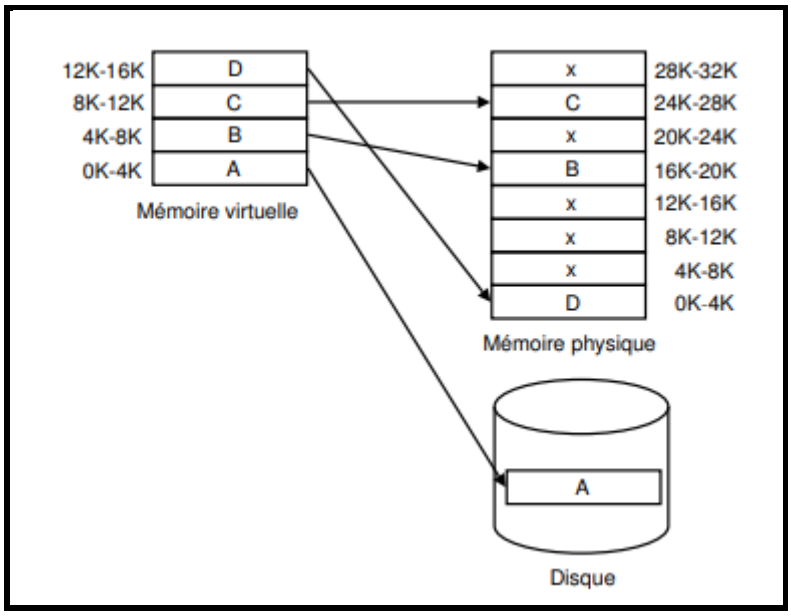
Niveau : 2LMD
Chargé de module : Mme. D. Boukhlouf

Année universitaire 2018/2019

II. Allocation non contiguë

1. Mémoire virtuelle

L a mémoire virtuelle est une technique qui permet d'exécuter des programmes dont la taille excède la taille de la mémoire réelle. L'espace d'adressage d'un processus, généré par les compilateurs et les éditeurs de liens, constitue la mémoire virtuelle du processus ou espace d'adressage logique, comme le montre la figure ci-dessous. Avec des techniques adéquates, comme on verra dans le présent chapitre, sa taille peut être très supérieure à celle de la mémoire physique ou réelle.



Exemple (voir la figure ci-dessus) Espace des adresses virtuelles et espace physique : Le processus dans son espace virtuel continu comprend quatre pages : A, B, C et D. Trois blocs sont situés dans la mémoire physique et un est sur le disque.

Il serait trop coûteux d'attribuer à tout processus un espace d'adressage complet, surtout parce que beaucoup n'utilisent qu'une petite partie de son espace d'adressage. En général, la mémoire virtuelle et la mémoire physique sont structurées en unités d'allocations (pages pour la mémoire virtuelle et cadres pour la mémoire physique). La taille d'une page est égale à celle d'un cadre. Lorsqu'un processus est en cours d'exécution, seule une partie de son espace d'adressage est en mémoire.

Les adresses virtuelles référencées par l'instruction en cours doivent être traduites en adresses physiques. Cette conversion d'adresse est effectuée par des circuits matériels de gestion.

Si cette adresse correspond à une adresse en mémoire physique, on transmet sur le bus l'adresse réelle,

Sinon il se produit un **défaut de page**.

Par exemple, si la mémoire physique est de 32 Ko et l'adressage est codé sur 16 bits, l'espace d'adressage logique peut atteindre la taille 2^{16} soit 64 Ko.

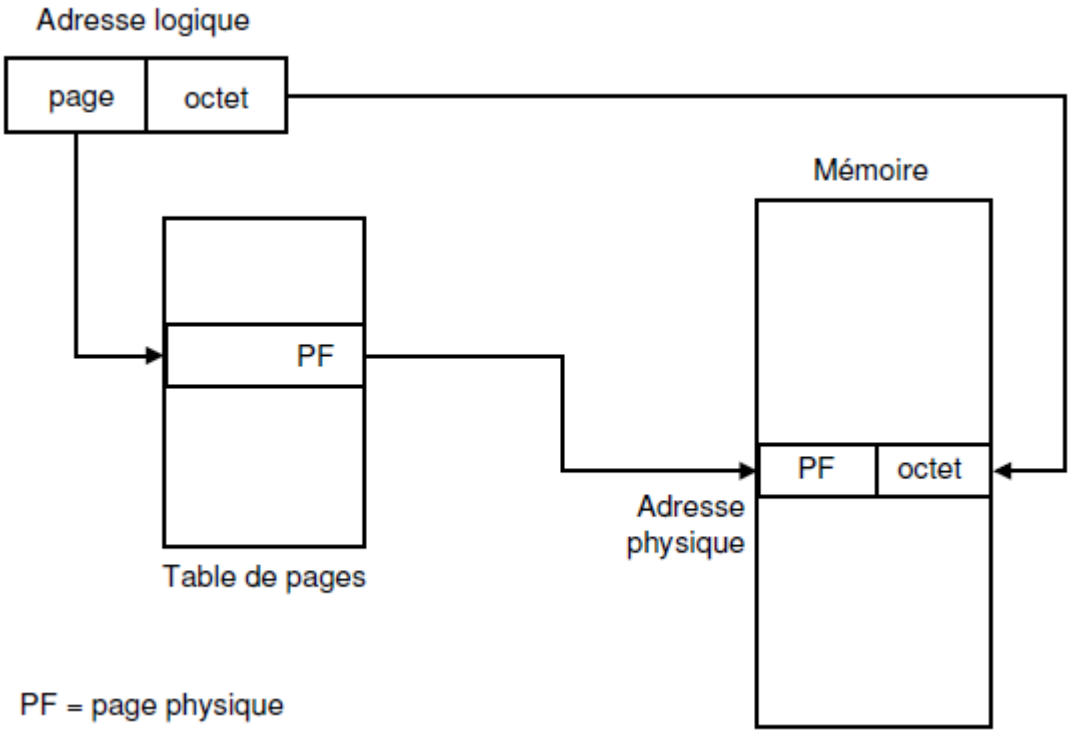
L'espace d'adressage est structuré en un ensemble d'unités appelées **pages** ou **segments**, qui peuvent être chargées séparément en mémoire.

L'ensemble de son espace d'adressage (mémoire virtuelle) est stocké sur disque. Pour exécuter un processus, le système d'exploitation charge en mémoire uniquement une ou quelques unités (pages ou segments) y compris celle qui contient le début du programme. Lorsqu'un processus est en cours d'exécution, seule une partie de son espace d'adressage est en mémoire principale. Cette partie est dite résidante. Les parties de cet

espace sont chargées en mémoire principale à la demande. Elles peuvent être dispersées en mémoire centrale.

2. Pagination pure

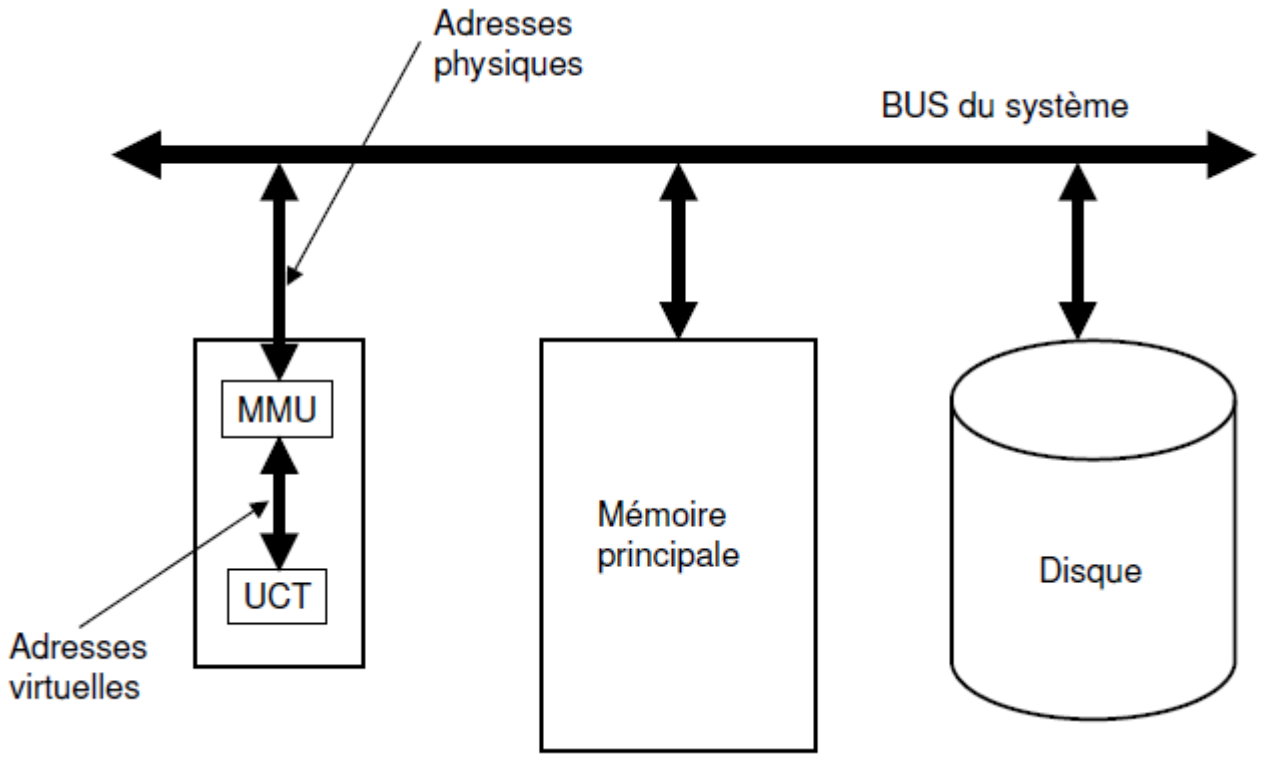
La mémoire virtuelle et la mémoire physique sont structurées en unités d'allocations appelés **pages** pour la mémoire virtuelle et cases ou **cadres** pour la mémoire physique. La taille d'une page est fixe et égale à celle d'un cadre. Elle varie entre 2 Ko et 16 Ko. 4 Ko est une valeur typique assez répandue. Un schéma de la traduction d'adresses lors de la **pagination pure** est montré sur la figure ci-dessous. Dans la pagination il n'y a pas de fragmentation externe car toutes les pages sont de même taille. Par contre, il peut y avoir une fragmentation interne si la dernière page de l'espace d'adressage logique n'est pas pleine.



3. Unité de gestion de la mémoire (MMU)

Les adresses virtuelles, générées par les compilateurs et les éditeurs de liens, sont des couples composés d'un numéro de page et d'un déplacement relatif au début de la page. Les adresses virtuelles référencées par l'instruction en cours d'exécution doivent être converties en adresses physiques. Cette conversion d'adresse est effectuée par le MMU, qui est un ensemble de circuits matériels de gestion.

Nous allons considérer la page l'unité de transfert. Le MMU vérifie si l'adresse virtuelle reçue correspond à une adresse en mémoire physique, comme montré à la figure ci dessous. Si c'est le cas, le MMU transmet sur le bus de la mémoire l'adresse réelle, sinon il se produit un défaut de page. Un défaut de page provoque un déroutement (ou TRAP) dont le rôle est de ramener à partir du disque la page manquante référencée. La correspondance entre les pages et les cases est mémorisée dans une table appelée **Table de pages** (**TP**). Le nombre d'entrées dans cette table est égal au nombre de pages virtuelles. La **Table de pages** d'un processus doit être en totalité ou en partie en mémoire centrale lors de l'exécution du processus. Elle est nécessaire pour la conversion des adresses virtuelles en adresses physiques.



4. Structure de la Table de pages

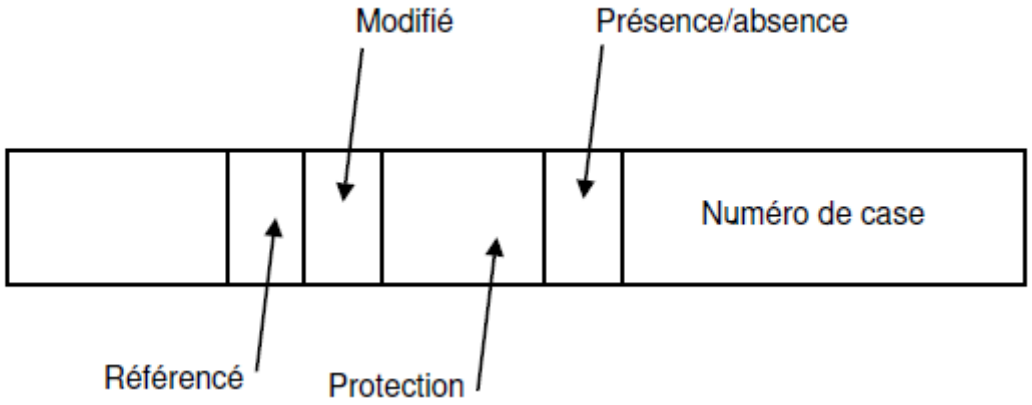
Chaque entrée de la **Table de pages** est composée de plusieurs champs, notamment :

1. Le bit de présence.
2. Le bit de référence (R).
3. Les bits de protection.
4. Le bit de modification (M).
5. Le numéro de case correspondant à la page.

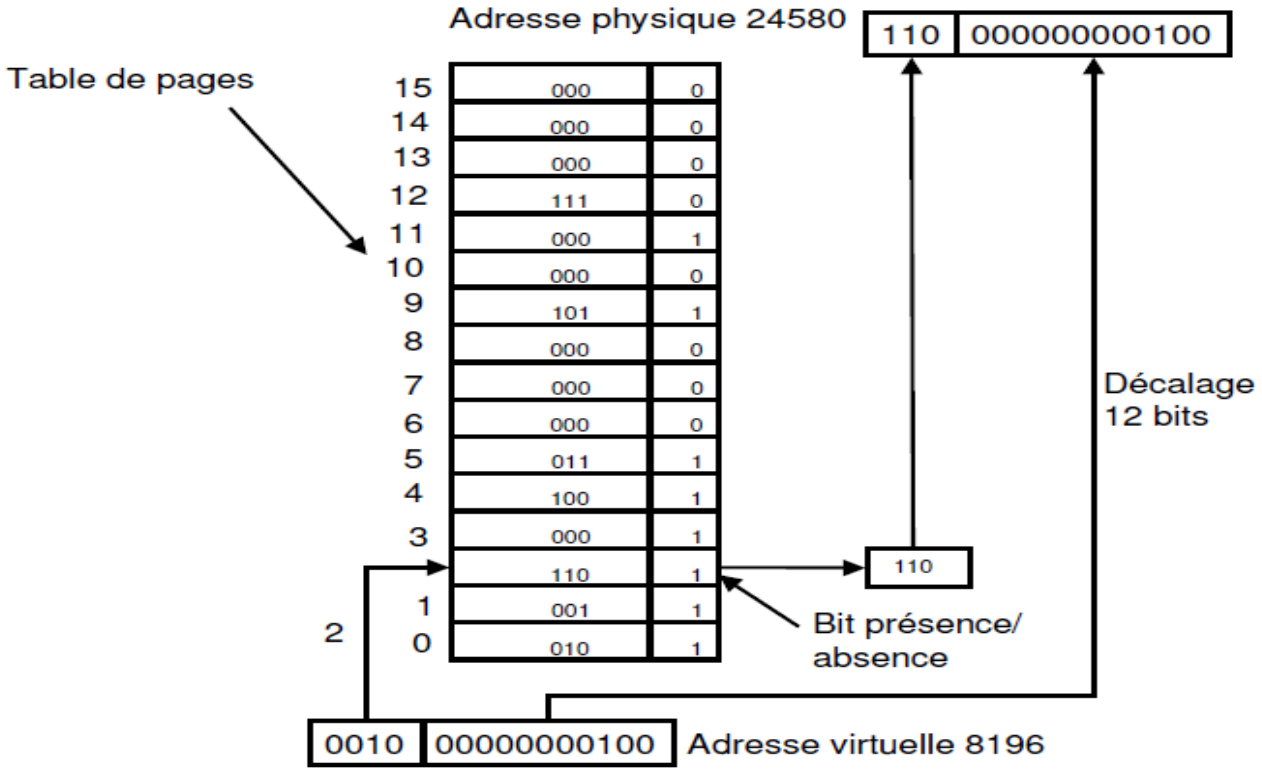
La structure d'une table de pages typique, bien que dépendant de la machine, est montrée sur la figure ci dessous. La taille varie de machine à machine, mais 32 bits est une taille répandue.

5. Fonctionnement d'un MMU

Le MMU reçoit, en entrée une adresse virtuelle et envoie en sortie l'adresse physique ou provoque un déroutement. Dans l'exemple de figure ci dessous, l'adresse virtuelle est codée sur 16 bits. Les 4 bits de poids fort indiquent le numéro de page, comprise entre 0 et 15. Les autres bits donnent le déplacement dans la page, entre 0 et 4095. Le MMU examine l'entrée dans la **Table de pages** correspondant au numéro de page, dans ce cas 2. Si le bit de présence est à 0, la page n'est pas en mémoire alors le MMU provoque un déroutement.

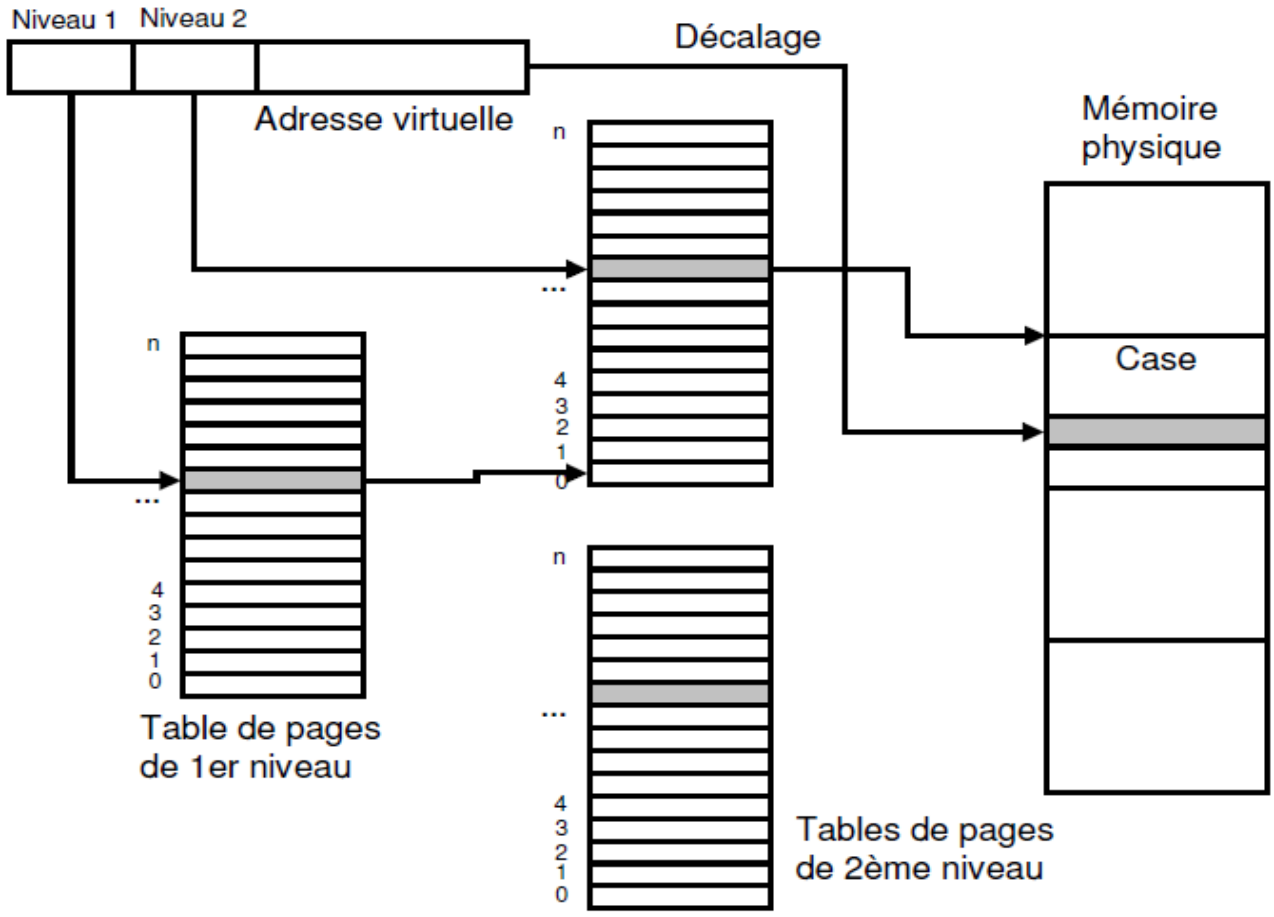


Sinon, il détermine l'adresse physique en recopiant dans les 3 bits de poids le plus fort le numéro de case (110) correspondant au numéro de page (0010), et dans les 12 bits de poids le plus faible de l'adresse virtuelle. L'adresse virtuelle 8196 (0010 0000 0000 0100) est convertie alors en adresse physique 24580 (1100 0000 000 0100) comme le montre la figure suivante. Le système d'exploitation maintient une copie de la table de pages pour processus, qui permet d'effectuer la translation des adresses.



6. Table de pages à plusieurs niveaux

La taille de la table de pages peut être très grande : par exemple, on aurait plus de 1 million d'entrées (2^{20}) pour un adressage virtuel sur 32 bits et des pages de 4 Ko. Pour éviter d'avoir des tables trop grandes en mémoire, de nombreux ordinateurs utilisent des tables des pages à plusieurs niveaux. Un schéma de table de pages à deux niveaux est montrée sur la figure suivante.



Par exemple, une table de pages à deux niveaux, pour un adressage sur 32 bits et des pages de 4 Ko, est composée de 1024 tables de 1 Ko. Il est ainsi possible de charger uniquement les tables de 1 Ko nécessaires. Dans ce cas, une adresse virtuelle de 32 bits est composée de trois champs : un pointeur sur la table du 1er niveau, un pointeur sur une table du 2ème niveau et un déplacement dans la page, de 12 bits.

7. Accès à la table de pages

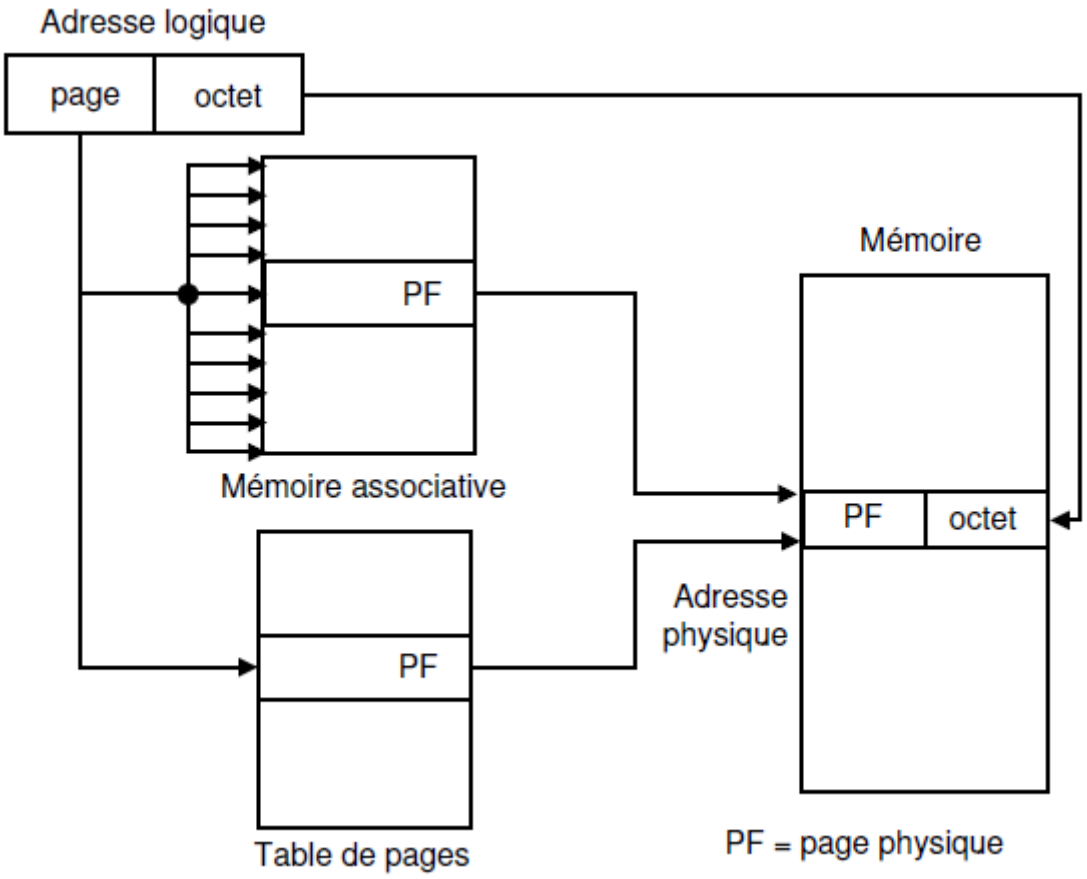
L'accès à la table de pages (qui se trouve en mémoire) peut se faire via un registre du MMU qui pointe sur la table de pages. Ceci est une solution lente, car elle nécessite des accès mémoire. Lorsqu'un processus passe à l'état élu, l'adresse de la table de pages est chargée dans le registre. On a constaté que la pagination peut avoir un grand impact sur les performances du système.

8. MMU avec une mémoire associative

Si la table de pages est grande, la solution précédente n'est pas réalisable. Afin d'accélérer la traduction d'adresses, on peut doter le MMU d'une table de registres machines rapides indexée au moyen du numéro de page virtuelle. Lorsqu'un processus passe à l'état élu, le système d'exploitation charge la **Table de pages** du processus dans la **Table de registres** à partir d'une copie située en mémoire centrale. La solution consiste à doter le MMU d'un dispositif appelé **mémoire associative**, qui est composée d'un petit nombre d'entrées, normalement entre 8 et 64. On a observé que la plupart des programmes ont tendance à faire un grand nombre de références à un petit nombre de pages. La mémoire associative contient des circuits spéciaux pour accéder aux adresses qui sont hautement référencées. La mémoire associative est appelée aussi **TLB** Translation Lookside Buffer . Ce composant contient des informations sur les dernières pages référencées. Chaque entrée du **TLB** est composée de :

1. Un bit de validité.
2. Un numéro de page virtuelle.
3. Un bit de modification (M).
4. Deux bits de protection.
5. Un numéro de case.

La traduction d'adresses en utilisant une mémoire associative est montrée à la figure suivante. Lorsqu'une adresse virtuelle est présentée au MMU,



il contrôle d'abord si le numéro de la page virtuelle est présent dans la mémoire associative, en le comparant simultanément (en parallèle) à toutes les entrées. S'il le trouve et le mode d'accès est conforme aux bits de protection, la case est prise directement de la mémoire associative (sans passer par la Table de pages). Si le numéro de page est présent dans la mémoire associative mais le mode d'accès est non conforme, il se produit un défaut de protection. Si le numéro de page n'est pas dans la mémoire associative, le MMU accède à la Table de pages à l'entrée correspondant au numéro de pages. Si le bit de présence de l'entrée trouvée est à 1, le MMU remplace une des entrées de la mémoire associative par l'entrée trouvée. Sinon, il provoque un défaut de page.

9. Nombre de cases allouées à un processus

On peut allouer un même nombre de cases mémoire à chaque processus. Par exemple, si la mémoire totale fait 100 pages et qu'il y a cinq processus, chaque processus recevra 20 pages. On peut aussi allouer les cases proportionnellement aux tailles des programmes. Si un processus est deux fois plus grand qu'un autre, il recevra le double de cases. L'allocation des cases peut se faire lors du chargement ou à la demande au cours de l'exécution.

10. Algorithmes de remplacement de page

A la suite d'un défaut de page, le système d'exploitation doit ramener en mémoire la page manquante à partir du disque. S'il n'y a pas de cadres libres en mémoire, il doit retirer une page de la mémoire pour la remplacer par celle demandée. Si la page à retirer a été modifiée depuis son chargement en mémoire, il faut la réécrire sur le disque. Quelle est la page à retirer de manière à minimiser le nombre de défauts de page ?

Le choix de la page à remplacer peut se limiter :

- aux pages du processus qui ont provoqué le défaut de page (**allocation locale**)
- ou à l'ensemble des pages en mémoire (**allocation globale**).

En général, l'allocation globale produit de meilleurs résultats que l'allocation locale. Des algorithmes de remplacement de page ont été proposés. Ces algorithmes mémorisent les références passées aux pages. Le choix de la page à retirer dépend des références passées. Il y a plusieurs algorithmes de remplacement de page : l'algorithme de remplacement aléatoire qui choisit au hasard la page victime (qui n'est utilisé que pour des comparaisons), l'algorithme optimal, l'algorithme FIFO, l'algorithme de l'horloge, et l'algorithme de la page la moins récemment utilisée, et bien d'autres.

10.1 Remplacement de page FIFO

Il mémorise dans une file de discipline FIFO (premier entré, premier sorti) les pages présentes en mémoire. Lorsqu'un défaut de page se produit, il retire la plus ancienne, c'est à dire celle qui se trouve en tête de file.

Exemple .

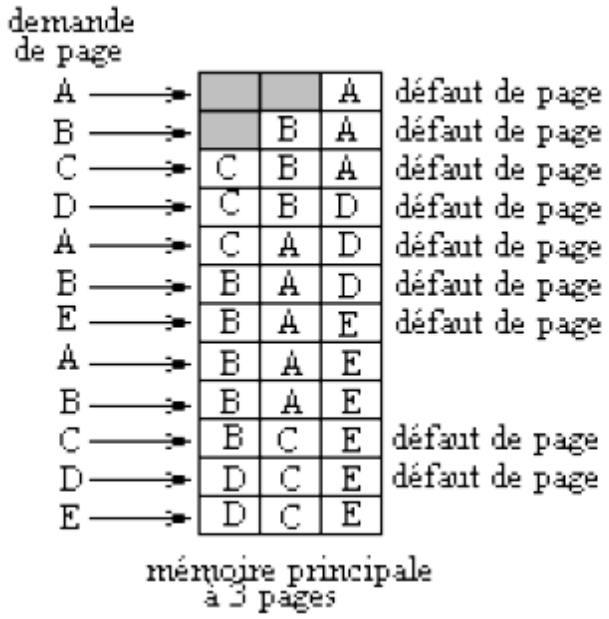
La suite de références w ={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1}

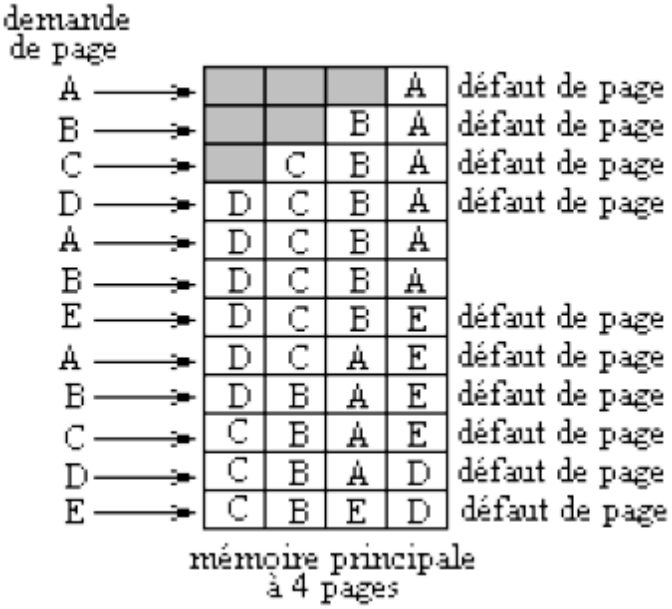
Avec m=3 cases, fait 15 défauts de page avec l'algorithme FIFO, comme le montre le tableau suivant :

m \ ω	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0	0	0	0	0	**7**	7	7
1		**0**	0	0	0	**3**	3	3	**2**	2	2	2	2	**1**	1	1	1	1	**0**	0
2			**1**	1	1	1	**0**	0	0	**3**	3	3	3	3	**2**	2	2	2	2	**1**

Cet algorithme ne tient pas compte de l'utilisation de chaque page. Par exemple, à la dixième référence la page 0 est retirée pour être remplacée par la page 3 puis tout de suite après la page retirée est rechargée. L'algorithme est rarement utilisé car il y a beaucoup de défauts de page.

anomalie de Belady : cette anomalie, étudiée par Belady, Nelson, Shedler, est spécifique à la stratégie FIFO. L'intuition conduit à penser que plus il y a de place en mémoire principale, moins il y a de défauts de pages. En réalité (c'est l'anomalie), cela est quelquefois faux. L'exemple ci-dessous explicite ce paradoxe : dans une mémoire à 3 pages, on rencontre 9 défauts de page ; dans une mémoire à 4 pages (donc plus grande), on rencontre pour le même traitement 10 défauts de page.





10.2 L'algorithme de remplacement de page optimal :

L'algorithme optimal de Belady consiste à retirer la page qui sera référencée le plus tard possible dans le futur. Cette stratégie est impossible à mettre en œuvre car il est difficile de prévoir les références futures d'un programme. Le remplacement de page optimal a été cependant utilisé comme base de référence pour les autres stratégies, car il minimise le nombre de défauts de page.

Exemple Soit un système avec N cases de mémoire et une suite de références :

w ={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1}. Voir la figure suivante :

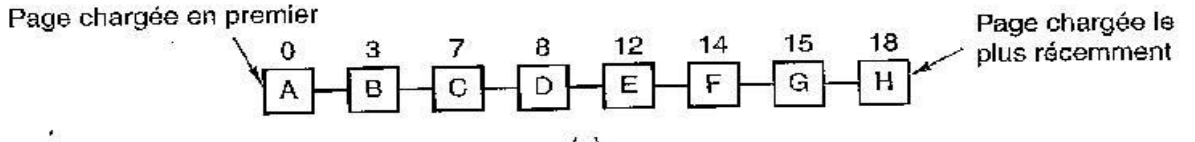
m \ ω	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	**7**	7	7	**2**	2	2	2	2	2	2	2	2	2	2	2	2	2	**7**	7	7
1		**0**	0	0	0	0	0	**4**	4	4	**0**	0	0	0	0	0	0	0	0	0
2			**1**	1	1	**3**	3	3	3	3	3	3	3	**1**	1	1	1	1	1	1

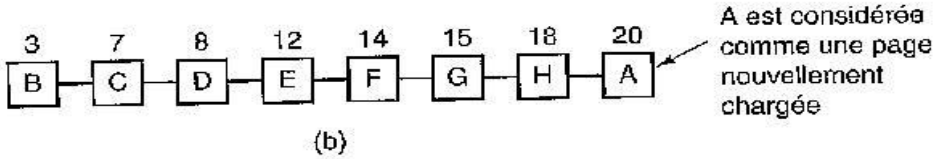
L'algorithme optimal fait donc 9 défauts de page.

10.3 L'algorithme de remplacement de page de la seconde chance

On peut apporter une modification simple à l'algorithme FIFO afin d'éviter la suppression d'une page à laquelle on a couramment recours : il s'agit d'inspecter le bit R de la page la plus ancienne. S'il est à 0, la page est à la fois ancienne et inutilisée, elle est donc remplacée immédiatement. Si le bit R est à 1, le bit est effacé, la page est placée à la fin de la liste des pages, et son temps de chargement est mis à jour comme si elle venait d'arriver en mémoire. La recherche continue alors. L'opération de cet algorithme, appelée seconde chance, est illustrée ci-dessous nous voyons que les pages de A à H se trouvent dans une liste chaînée, triées par leur temps d'arrivée en mémoire.

Supposons qu'un défaut de page se produise à l'instant 20. La page la plus ancienne est A, qui est arrivée à l'instant 0, quand le processus a démarré. Si le bit R de la page A est à 0, elle est évincée de la mémoire: elle est soit écrite sur le disque (si elle est modifiée), soit simplement abandonnée (si elle est restée identique à la page sur disque). D'un autre côté, si le bit R est à 1, A est placée à la fin de la liste et son temps de chargement est mis à jour avec le temps courant





(20). Le bit R est aussi mis à 0. La recherche d'une page convenable continue alors avec B.
Opération de la seconde chance.
(a) Pages triées dans un ordre FIFO.
(b) Liste de pages lorsqu'un défaut de page se produit à l'instant 20 et que le bit R de A est à 1.
Les nombres au-dessus des pages correspondent à l'instant de leur chargement. Le travail de l'algorithme de la seconde chance consiste à chercher une ancienne page qui n'a pas été référencée dans le précédent intervalle d'horloge. Si toutes les pages ont été référencées, l'algorithme de la seconde chance dégénère en pur algorithme FIFO. Pour être concrets, imaginons que toutes les pages de la figure précédente a ont leur bit R à 1. Le système d'exploitation déplace Finalement, il revient à la page A, dont le bit R est maintenant à 0. À ce moment-là, A est évincée. Ainsi, cet algorithme se termine
toujours.une par une les pages à la fin de la liste, effaçant le bit R chaque fois qu'il ajoute une page à la fin de la liste.

Exemple : La suite de références w ={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1}
Avec m=3 cases, fait 11 défauts de page.

7	1	2	1	2	1	2	1	2	1	2	0	2	0	2	1	2	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0	7	1	7	1
0	1	0	0	0	1	0	0	0	1	0	0	4	1	4	1	4	0	4	0	4	0	4	0	4	0	2	1	2	1	2	0	2	0	0	1
1	1	1	0	1	0	3	1	3	1	3	0	3	0	3	1	3	0	3	0	3	0	3	1	3	0	3	0	1	1	1	0	1	0	1	0

10.4 Remplacement de la page la moins récemment utilisée (LRU Least Recently Used)

L'algorithme LRU mémorise dans une liste chaînée toutes les pages en mémoire. La page la plus utilisée est en tête de liste et la moins utilisée est en queue. Lorsqu'un défaut de page se produit, la page la moins utilisée est retirée. Pour minimiser la recherche et la modification de la liste chaînée, ces opérations peuvent être réalisées par le matériel. Cet algorithme est coûteux.

Exemple

La suite de références w={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1}
Avec m=3 cases, fait 12 défauts de page avec l'algorithme de remplacement de la page la moins récemment utilisée, comme le montre le tableau suivant :

m \ ω	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0	1	1	1	1	**1**	1	1
1		**0**	0	0	0	0	0	**0**	0	**3**	3	3	3	3	3	**0**	0	0	0	0
2			**1**	1	1	**3**	3	3	**2**	2	2	2	2	2	2	2	2	**7**	7	7

Le problème avec cet algorithme est la difficulté d'implémentation, qui requiert un support du hardware. Il faut une manière de mémoriser le temps à chaque fois qu'une page est référencée. On peut aussi utiliser une technique de vieillissement, où un registre de n bits est associé à chaque page. Le bit le plus significatif est mis à 1 chaque fois que la page est référencée. Régulièrement, on décale vers la droite les bits de ce registre. Lorsque qu'on doit expulser une page, on choisit celle dont la valeur est la plus petite. On pourrait aussi mettre une page au dessus d'une pile chaque fois qu'elle est référencée. On expulsera la page qui se trouve au fond de la pile.

a.Algorithme LRU avec la méthode des masques :

C'est une méthode d'implémentation de l'algorithme de remplacement LRU selon la méthode des masques.
1. A chaque page est associée un octet.
2. La valeur initiale du masque est un octet (00000000). Le bit de poids fort est mis à 1 à chaque utilisation de cette page.

3. A chaque période (N ms), on fait un décalage à droite de l'octet associé à chaque page.
4. la page victime c'est la page qui a la plus petite valeur de masque au moment du lancement de l'algorithme.

```
Algorithme LRU ;
Var trouv :booléen ;
Min : entier ;
Début
Min← octet-min(nombre-page-charge-mc) ;
Trouv← test-unique(min) ;
Si (trouv) Alors {la page est unique}
Num-page-victime ← Selection(min,list-page-charg) ;
Sinon {la page n'est pas unique}
Num-page-victime ← Selection(min,list-page-charg, FIFO) ;
Finsi ;
```

b. Comparaison entre trois méthodes d'implantation de LRU

Méthode de Pile	Méthode des compteurs	Méthode des Masques
Elle est difficile à gérer	Elle est facile à mettre en œuvre.	Elle est facile mais nécessite un décalage (mise à jour)
La taille de la pile augmente	Le nombre de compteurs augmente avec la taille de la MC	En augmentant la taille de la MC on va augmenter le nombre d'octets nécessaire.

10.5 Remplacement de la page la moins fréquemment utilisée : LFU(Least frequently used)

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.

Exemple :

Chaine de références : **7-0-1-2-0-3-0-4-2-3-0-3-2-1-2-0-1-7-0-1.**

7	1	2	1	2	1	2	1	2	1	4	1	4	1	3	1	3	1	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2
0	1	0	1	0	2	0	2	0	3	0	3	0	3	0	3	0	4	0	4	0	4	0	4	0	5	0	5	0	5	0	6	0	6
1	1	1	1	1	1	3	1	3	1	3	1	2	1	2	1	2	1	2	2	1	1	2	1	1	1	1	1	7	1	7	1	1	1

Il y'a 13 défauts de page.

11. Écroulement du système

Si le système passe plus de temps à traiter les défauts de page qu'à exécuter des processus on peut avoir des problèmes de l'écroulement du système. Si le nombre de processus est trop grand, l'espace propre à chacun sera insuffisant, ils passeront alors leur temps à gérer des défauts de pages. C'est ce qu'on appelle l'**écroulement du système**. On peut limiter le risque d'écroulement en surveillant le nombre de défauts de page provoqués par un processus. Si un processus provoque trop de défauts de pages (au-dessus d'une limite

supérieure) on lui allouera plus de pages ; au-dessous d'une limite inférieure, on lui en retirera. S'il n'y a plus de pages disponibles et trop de défauts de pages, on devra suspendre un des processus.

12. Segmentation

Dans un système paginé, l'espace d'adressage virtuel d'un processus est à une dimension. Or en général, un processus est composé d'un ensemble d'unités logiques :

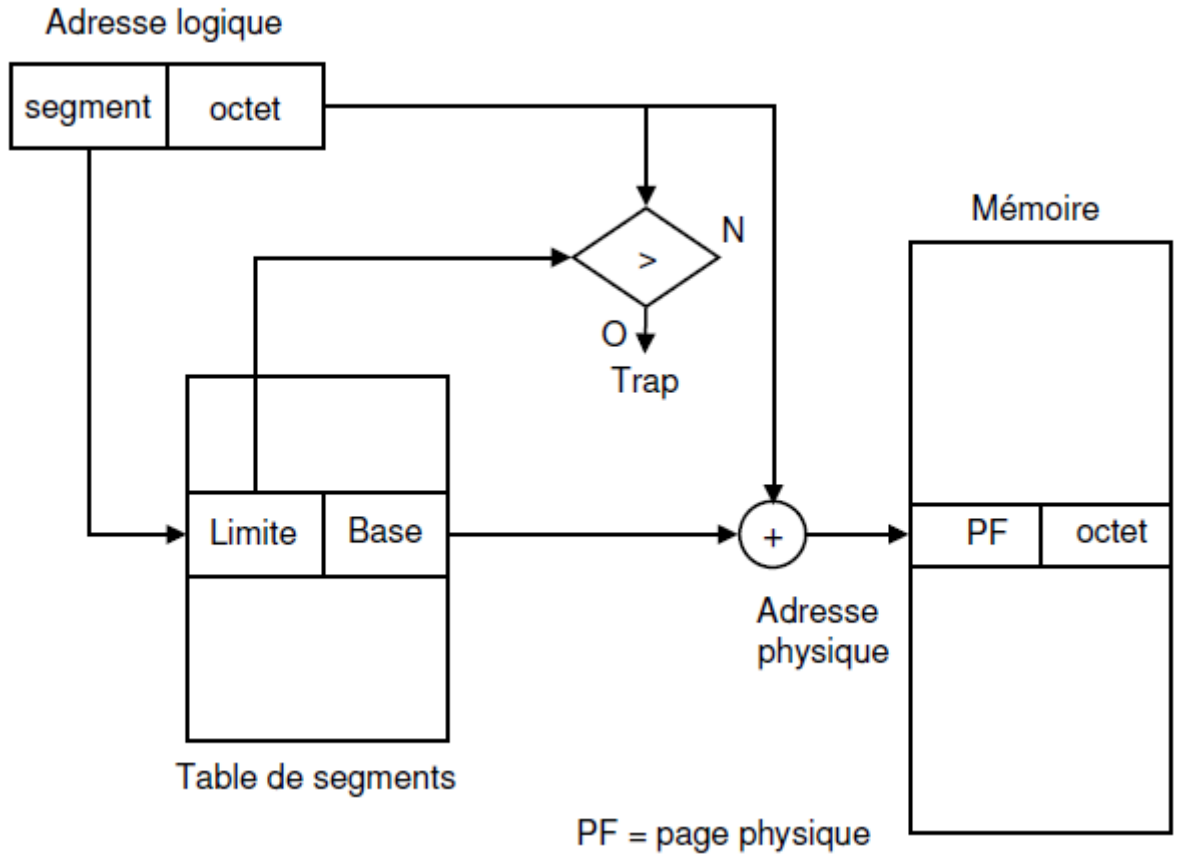
. **Les différents codes :** le programme principal, les procédures, les fonctions bibliothèques.

. **Les données initialisées**.

. **Les données non initialisées**.

. **Les piles d'exécution**.

L'idée de la technique de **segmentation** est d'avoir un espace d'adressage à deux dimensions. On peut associer à chaque unité logique un espace d'adressage appelé **segment**. L'espace d'adressage d'un processus est composé d'un ensemble de segments. Ces segments sont de tailles différentes (fragmentation externe). Un schéma de traduction d'adresses par segmentation est montré sur la figure ci dessous. La segmentation facilite l'édition de liens, ainsi que le partage entre processus de segments de données ou de codes.

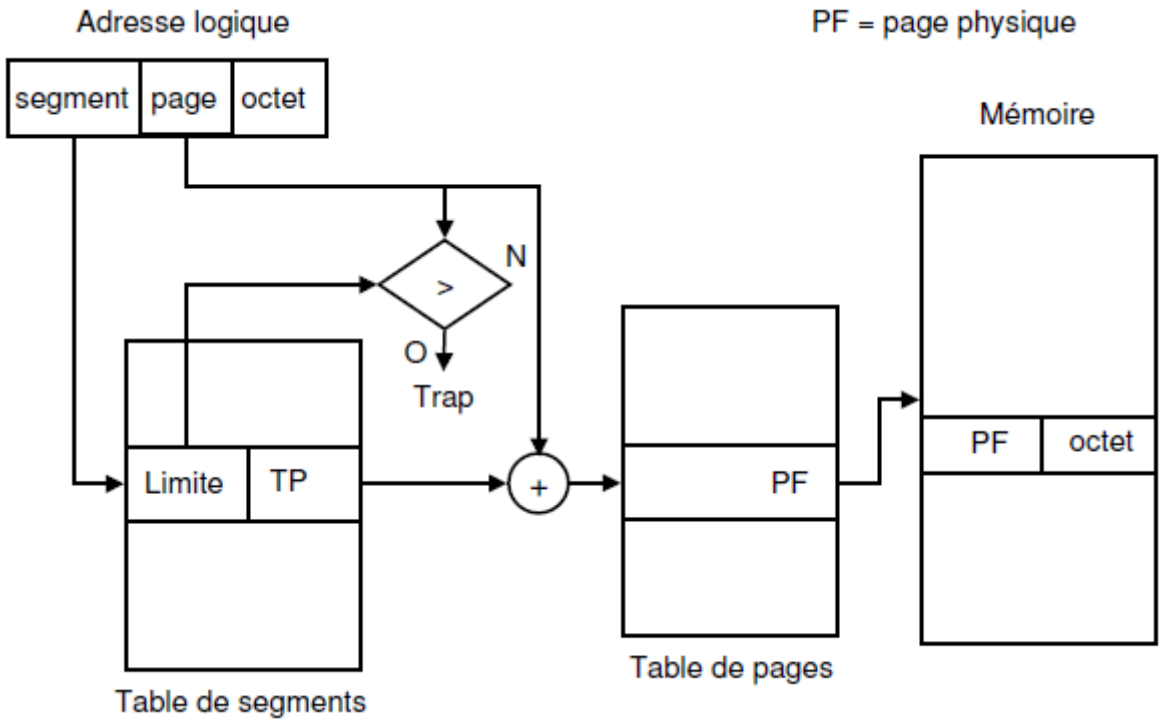


13. Segmentation paginée

La segmentation peut être combinée avec la pagination. Chaque segment est composé d'un ensemble de pages. Les adresses générées par les compilateurs et les éditeurs de liens, dans ce cas, sont alors des triplets :

<numéro du segment, numéro de page, déplacement dans la page>

Le schéma de traduction d'adresses par segmentation paginée est montré sur la figure ci-dessous..



14. Mémoire cache

La mémoire cache est une mémoire à temps d'accès très court (10 ns). Elle coûte plus chère. Le temps d'accès à la mémoire principale est 100 ns. La mémoire cache est placée entre le processeur et la mémoire centrale. Dans un système à pagination, lorsqu'une adresse virtuelle est référencée, le système examine si la page est présente dans le cache. Si c'est le cas, l'adresse virtuelle est convertie en adresse physique. Sinon, le système localise la page puis la recopie dans le cache. Le but du cache est de minimiser le temps d'accès moyen t.

t = temps d'accès + taux d'échec *temps de traitement de l'échec.