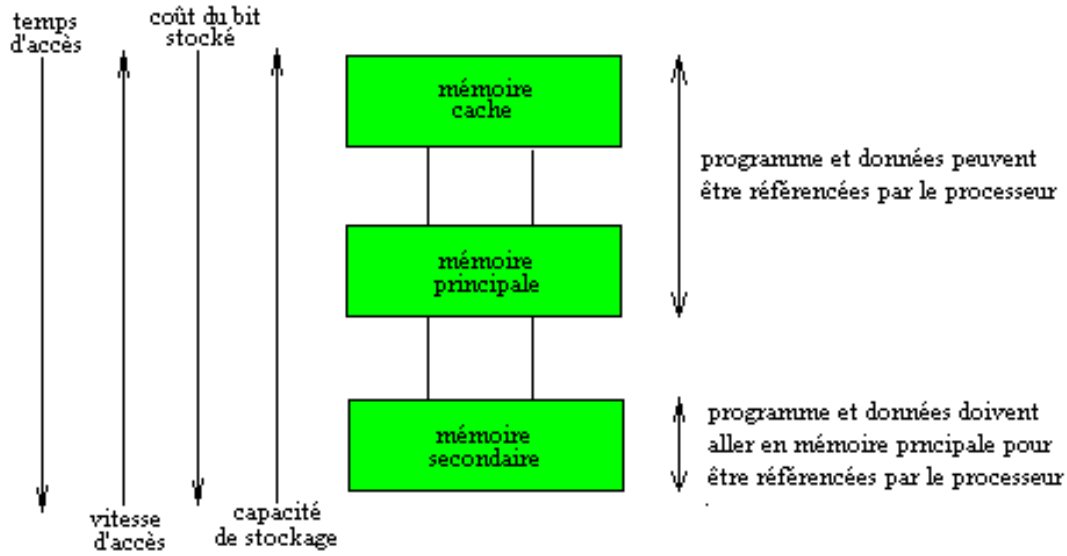
Gestion de la mémoire

Mémoire physique

Généralités

Autrefois, la mémoire principale était une ressource coûteuse. Elle devait donc être utilisée de manière optimale et diverses stratégies étaient employées. Par ailleurs, l'apparition de diverses variétés de mémoire ont conduit à une hiérarchie basée sur le temps d'accès (ou inversement la capacité de stockage):



Les principales stratégies de gestion de la mémoire se classent en trois catégories :

- stratégies de recherche (fetch stratégies) : recherche à la demande, anticipation (prefetch)
- stratégies de placement : first fit, best fit, worst fit
- stratégies de remplacement : random, fifo, lru, lfu, nur

Une notion importante a également, avec le progrès technologique, fait son apparition, celle de non-contiguïté ; jadis, en effet, l'allocation mémoire était contiguë, chaque programme occupait un bloc d'adresses séquentielles ; la non-contiguïté, au contraire, correspond à la répartition d'un programme sur plusieurs blocs non nécessairement adjacents.

système mono-utilisateur

Commençons par examiner cette situation simple qui correspond à des temps anciens des systèmes d'exploitation. Un utilisateur seul est présent et possède la machine pour lui tout seul.

Gestion de la mémoire

Solution des exercices

Solution de l'exercice 1

- FIFO : la page la plus anciennement chargée est celle qui sera remplacée : il s'agit de la page 2
- LRU : la page la moins récemment utilisée est celle qui sera remplacée : il s'agit de la page 1
- NRU : on se base sur les bits rb et mb. Rappelons que ces deux bits permettent un classement en 4 catégories : catégorie 0 (rb=0 et mb=1), catégorie 1 (rb=0 et mb=1), catégorie 2 (rb=1 et mb=0), catégorie 4 (rb=1 et mb=1). rb est émis à zéro périodiquement. Les catégories de bas niveau sont les premières concernées par le remplacement de pages : il s'agit ici de la page 0.

Solution de l'exercice 2

1) First Fit : utilisation de la première zone libre

état initial	10K	4K	20K	18K	7K	9K	12K	15K
placement de 12K :	10K	4K	8K	18K	7K	9K	12K	15K
placement de 10K :		4K	8K	18K	7K	9K	12K	15K
placement de 9K :		4K	8K	9K	7K	9K	12K	15K

2) Best Fit : meilleur ajustement

état initial	10K	4K	20K	18K	7K	9K	12K	15K
placement de 12K :	10K	4K	20K	18K	7K	9K		15K
placement de 10K :		4K	20K	18K	7K	9K		15K
placement de 9K :		4K	20K	18K	7K			15K

3) Worse Fit : on prend le plus grand emplacement libre

état initial	10K	4K	20K	18K	7K	9K	12K	15K
placement de 12K :	10K	4K	8K	18K	7K	9K	12K	15K
placement de 10K :	10K	4K	8K	8K	7K	9K	12K	15K
placement de 9K :	10K	4K	8K	8K	7K	9K	12K	6K

Solution de l'exercice 3

- L'adresse virtuelle 20 (page 0) correspond à l'adresse physique 8 x 1024 + 20 = 8 212
- L'adresse virtuelle 4100 (page 1) correspond à l'adresse physique 4100
- L'adresse physique 24684 (page 6) correspond à la page virtuelle 2 et à l'adresse 8 x 1024 + (24684 - 24 x 1024) = 8300
- L'adresse virtuelle 26000 (page 6) ne correspond pas à une page physique (il y aura une interruption pour charger cette page désirée en mémoire physique).

Gestion de la mémoire

Exercices

Exercice 1

Un ordinateur possède une mémoire de 4 pages. Pour chacue des pages, le gestionnaire de mémoire tient à jour les indicateurs suivants : date de chargement, date de dernière référence, rb (bit indiquant si la page a été référencée), mb (bit indiquant si la page a été modifiée). A un instant donné, la situation est la suivante :

page	t.chargement	t.dern.ref.	rb	mb
0	126	279	0	0
1	230	260	1	0
2	120	272	1	1
3	160	280	1	1

Indiquer page qu'il faudra remplacer prochainement dans le contexte de chacune des stratégies suivantes :

- FIFO,
- LRU,
- NRU

Exercice 2

Dans un système de gestion mémoire à partitions variables, on constate que la liste des "trous" est la suivante (dans l'ordre des adresses mémoire croissantes) :

10K 4K 20K 18K 7K 9K 12K 15K

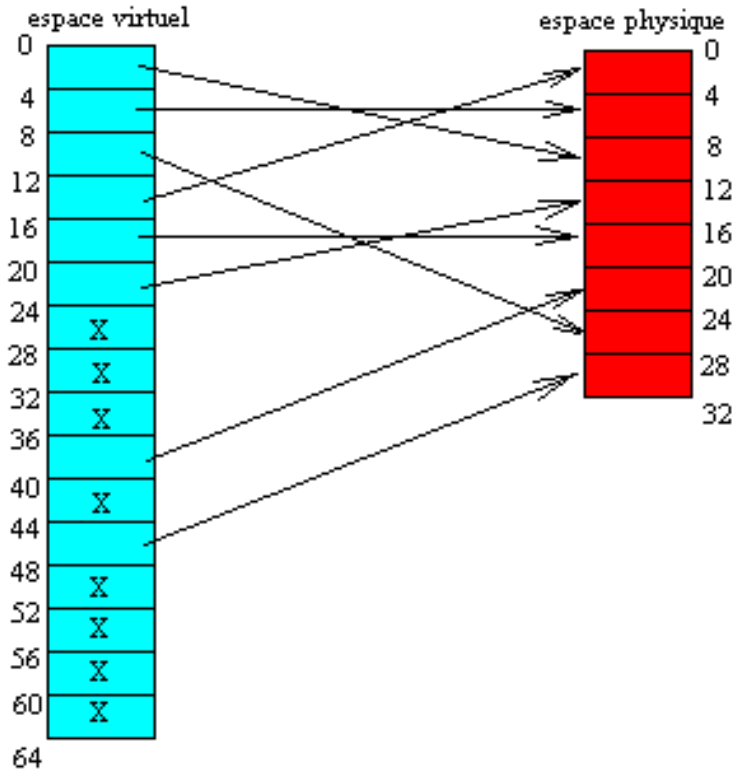
On veut placer successivement des données de volumes respectifs

12K 10K 9K

dans la mémoire. Indiquer, dans l'ordre des adresses croissantes, la nouvelle liste des trous après l'opération précédente dans le cadre de chacune des stratégies de placement : First Fit, Best Fit, Worst Fit.

Exercice 3

Le schéma suivant représente une topographie (mapping) faisant correspondre les adresses virtuelles et les adresses physiques d'un système à mémoire virtuelle (pages de 4K). La mémoire physique correspond à 8 pages :



Compléter le tableau de correspondance suivant :

adresse virtuelle	adresse physique
20	
41000	
	24684

26000	

Exercice 4

Un système possède une mémoire principale de 4200 mots de 8 bits. A un moment, la mémoire est occupée par des blocs A, B, C de longueurs respectives 1000, 500, 800 octets et dont les adresses d début sont respectivement 1000, 2900, 3400. Quand un nouveau bloc est chargé en mémoire, la stratégie suivante est utilisée :

a) on utilise d'abord l'algorithme du best fit pour localiser un trou de taille appropriée

b) si l'algorithme précédent échoue, on réorganise la mémoire en concaténant les blocs présents à partir de l'adresse 0, puis on reprend l'algorithme du best fit.

Indiquer par un schéma la configuration de la mémoire après le chargement successif des blocs suivants :

D : longueur 500 octets
E : longueur 1200 octets
F : longueur 200 octets

Exercice 5

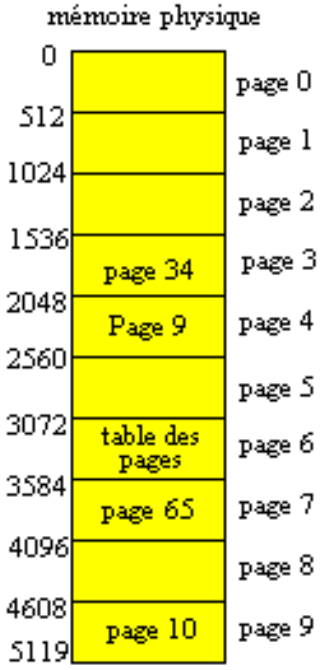
Dans un système paginé, la taille d'une page est de 512 mots, la mémoire virtuelle possède 512 pages numérotées de 0 à 511. La mémoire physique possède 10 pages numérotées de 0 à 9. Le contenu courant de la mémoire physique est donné ci-dessous.

1) La table des pages possède une structure simplifiée à deux colonnes, la première indiquant le numéro de page virtuelle, la seconde le numéro de page physique. Quel est l'état courant de la table des pages ?

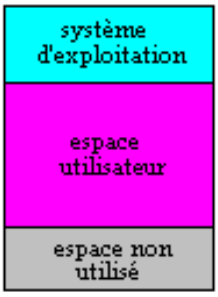
2) La page virtuelle 49 est chargée à l'adresse physique 0 et la page virtuelle 34 est remplacée par la page virtuelle 12. Donner la nouvelle table des pages.

3) Quelles sont les adresses physiques correspondant aux adresses virtuelles 4608, 5119, 5120, 33300 ?

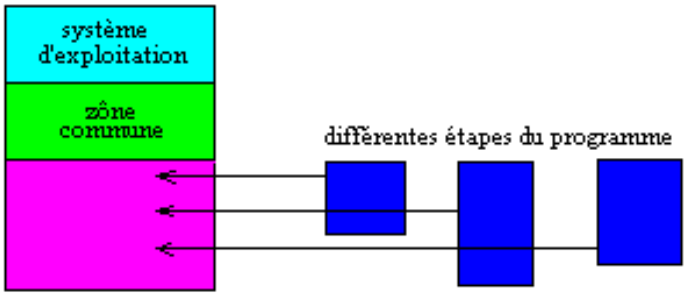
4) Que se passe-t-il quand l'adresse virtuelle 33300 est référencée ?



Si le programme de cet utilisateur n'est pas trop volumineux, il peut tenir entièrement en mémoire.



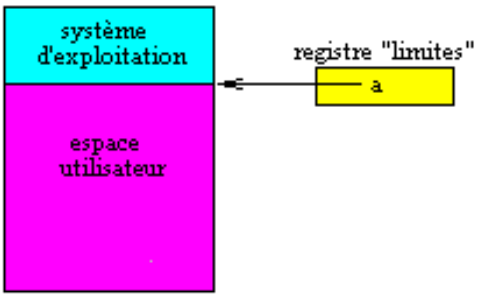
Si, par contre, le programme est trop volumineux, on pratique par overlays.



Un premier problème à régler est celui de la protection : il ne faut pas que deux zones d'information se chevauchent, en particulier, le programme utilisateur ne doit pas empiéter sur le système d'exploitation ; on utilise, pour cela, un registre "limites" :

On effectue un test comparatif : adresse > a ?

Lorsqu'on travaille dans la zone dédiée à l'utilisateur, on est en mode "utilisateur" (user). Pour les appels au système, on utilise des instructions spéciales (privilégiées) : mode "superviseur" (kernel ou system)..

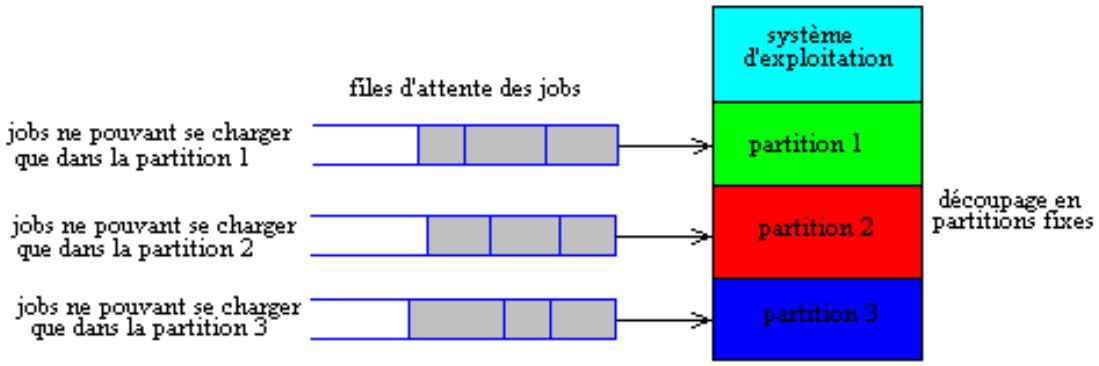


multiprogrammation avec partitions fixes

La multiprogrammation permet une meilleure utilisation du processeur ; plusieurs programmes doivent alors être en mémoire et par suite il convient d'augmenter la taille de la mémoire. Celle-ci est alors découpée en partitions de taille fixe.

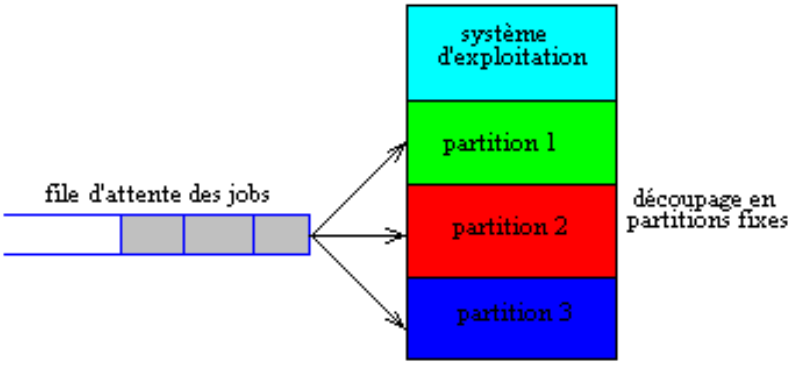
- chargement et traduction en "absolu"

La compilation produit des programmes avec des adresses absolues (ce qui suppose que le programmeur connaisse la machine d'exploitation) ; ils ne peuvent donc se charger que dans une partition donnée

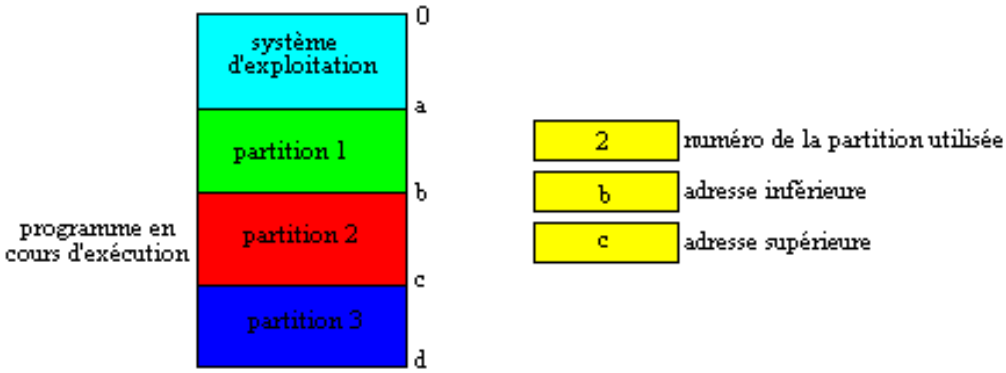


- chargement et traduction en "relogeable"

Une file d'attente suffit, mais les compilateurs et les chargeurs (ici avec un mécanisme de translation d'adresse) sont évidemment plus complexes

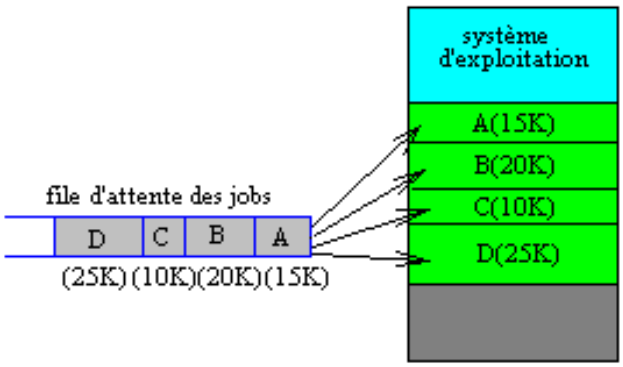


Le système de protection est également basé sur l'utilisation de registres.



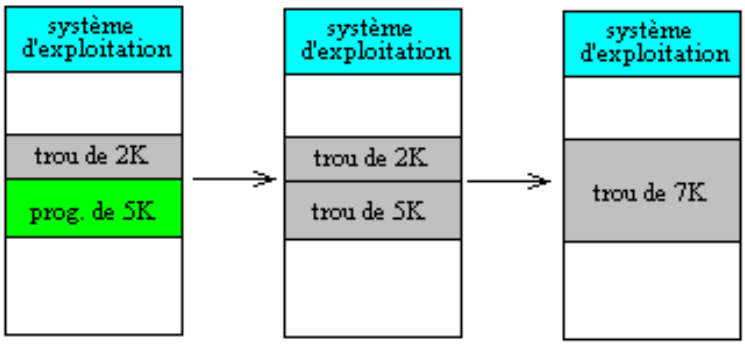
multiprogrammation avec partitions variables

L'utilisation de partitions de taille variable a pur objectif le meilleur ajustement des partitions aux tailles des jobs (n'oublions pas qu'autrefois la mémoire était coûteuse, il fallait donc optimiser son usage).

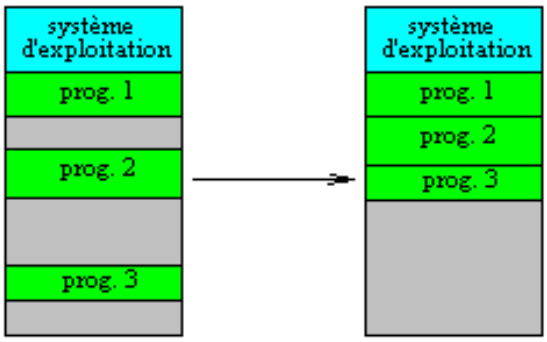


Un nouveau problème survient : un job qui se termine laisse un "trou". Ce trou est utilisé pour accueillir un nouveau job qui ne le remplira pas complètement en général. On va donc, au fur et à mesure que le temps passe, vers une multitude de "petits trous". La somme de ces petits trous peut alors n'être pas négligeable et deux techniques régulatrices peuvent être mises en oeuvre :

- la coalescence : on désigne sous ce nom une réunion de trous contigus en un seul trou.



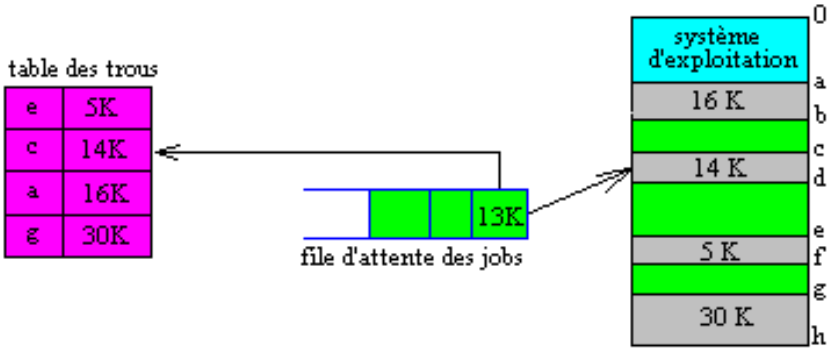
- le compactage : réorganisation complète de la mémoire ; son inconvénient est évidemment l'arrêt de l'exécution des travaux pendant le compactage.



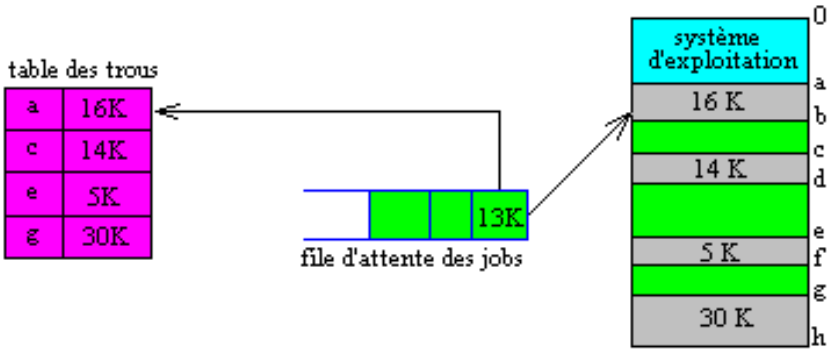
stratégies de placement

Dans le cadre de la multiprogrammation avec partitions variables, examinons les stratégies de placement :

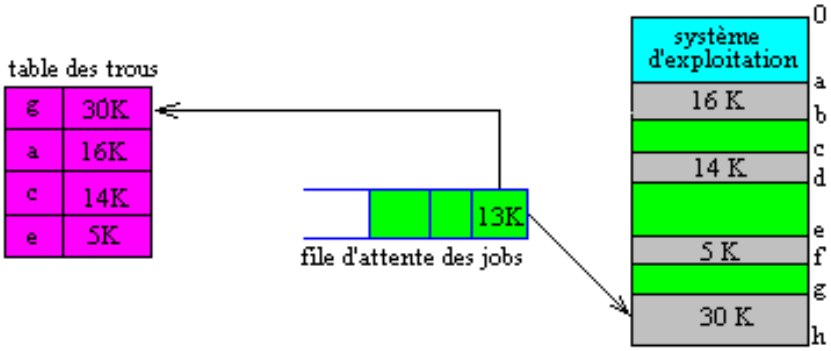
- Best fit : un nouveau job est placé dans le trou qui s'ajuste le mieux. La gestion de l'espace libre (donc des trous) est effectué au moyen d'une table triée par ordre croissant des tailles des trous.



- First fit : un nouveau trou est placé dans le premier trou qui peut l'accueillir. La table des trous peut être non triée ou triée par ordre croissant des adresses de début des trous.

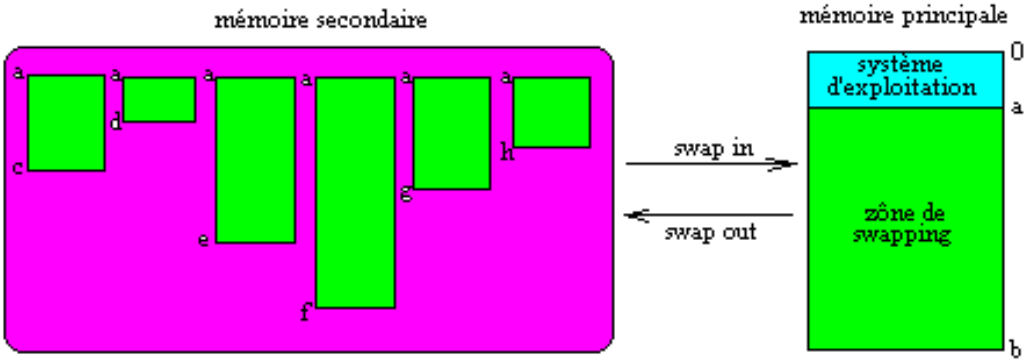


- Worst fit : un nouveau job est automatiquement placé dans le trou de plus grande taille. La table des trous est triée par ordre décroissant de taille des trous.



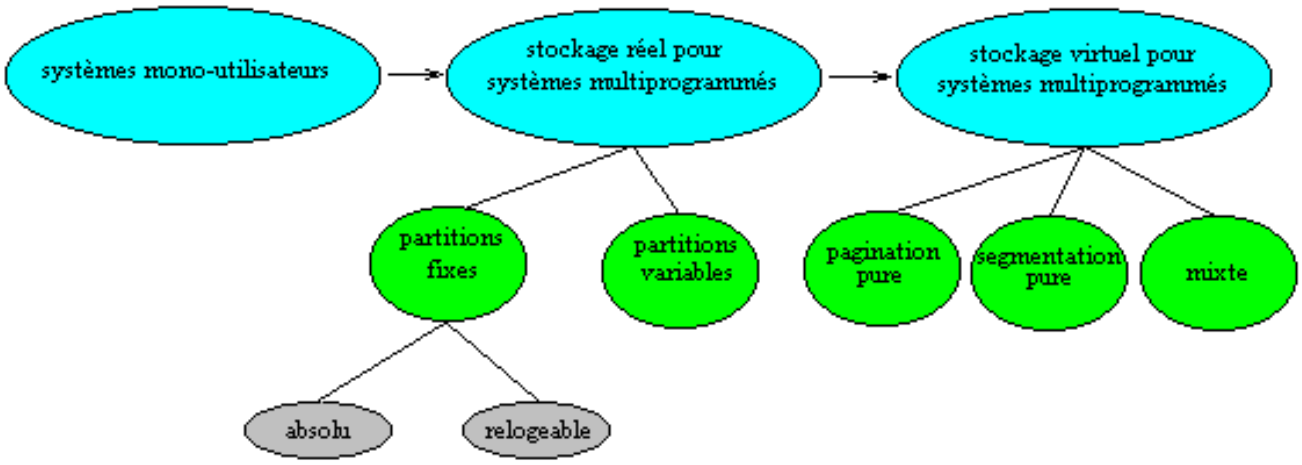
multiprogrammation avec swapping

Les systèmes en temps partagé, à leurs débuts, utilisaient la technique du swapping. On verra plus loin que la mémoire virtuelle reprend cette technique.



Organisation en mémoire virtuelle

Le concept de mémoire virtuelle est apparu en 1960 (Atlas, Université de Manchester). L'évolution des techniques est décrite dans le schéma ci-dessous :

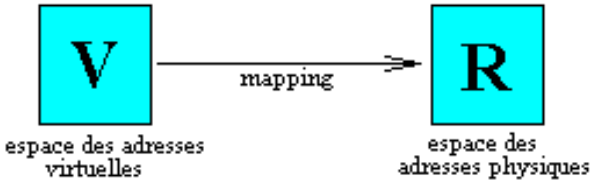


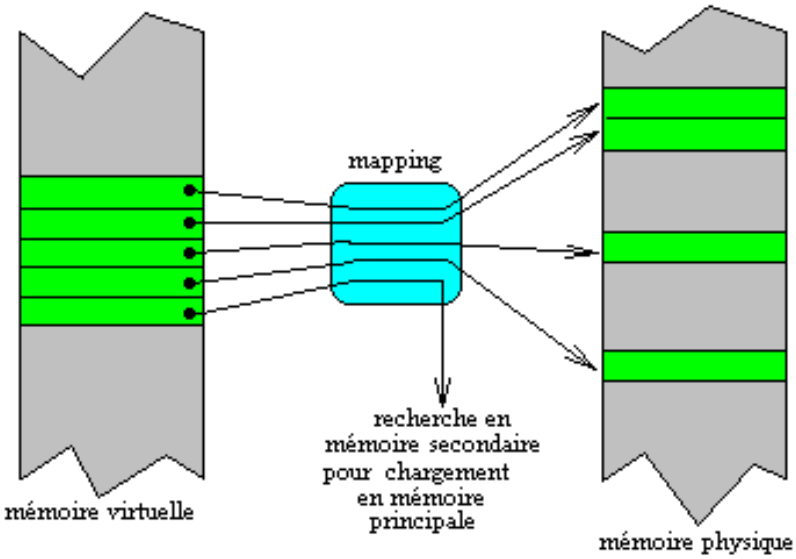
Concepts de base

Le principe fondamental est la dissociation entre

- l'adresse référencée dans un processus en exécution : adresse virtuelle
- l'adresse utilisée en mémoire principale : adresse physique ou réelle

La correspondance entre l'adresse virtuelle et l'adresse physique est effectuée par un "mapping" (Dynamic Address Translation)



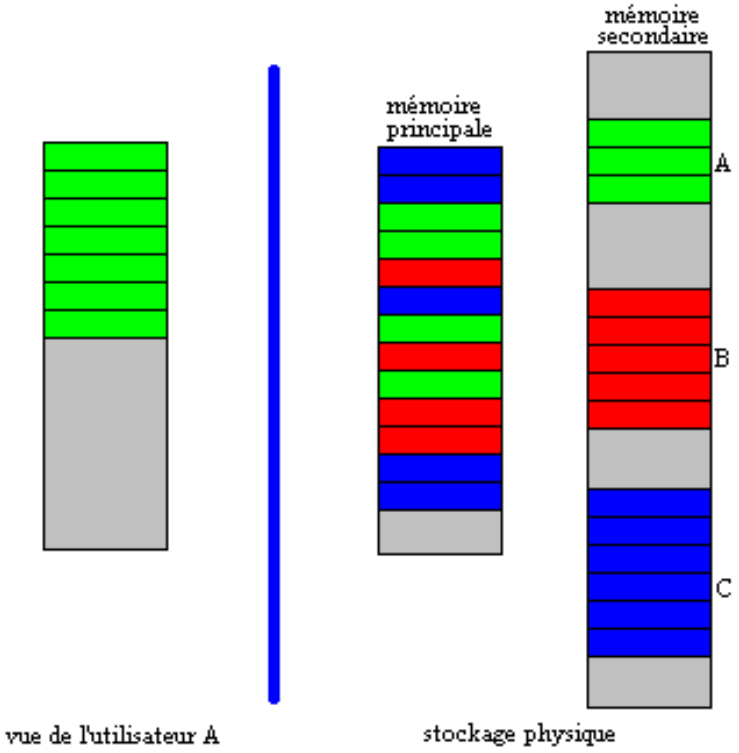


Chaque fois qu'une adresse virtuelle est référencée, le mapping (ou topographie) convertit cette adresse en une adresse physique qui est l'emplacement du début de bloc en mémoire principale. Bien entendu, ceci suppose que le bloc a préalablement été chargé en mémoire principale. Si ce n'est pas le cas, un processus d'interruption permet de rechercher le bloc en mémoire secondaire, de le charger en mémoire principale (il possède alors une adresse physique) et d'effectuer la correspondance adresse virtuelle vers adresse physique.

Les intérêts du concept de mémoire virtuelle sont évidents :

- L'espace physique de stockage (et notamment la hiérarchie des mémoires) est utilisé de manière optimale.
- L'utilisateur est libéré des contraintes de stockage
- L'utilisateur a une vue logique de l'espace de stockage.

A un instant donné, la situation est la suivante, pour plusieurs programmes en exécution :

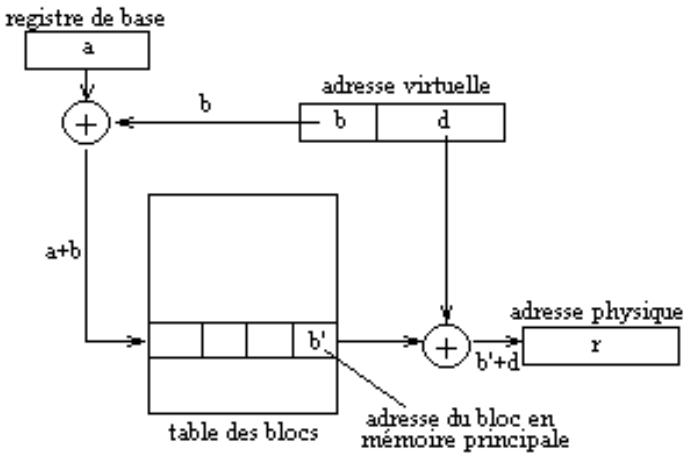


Les transferts entre la mémoire principale et la mémoire secondaire nécessite une organisation de l'information en blocs ; le système d'exploitation possède à tout moment la liste des blocs qui sont en mémoire principale. Si les blocs ont une taille fixe, on les appelle des **pages** ; s'ils sont de taille variable, on les appelle des **segments**.

La structure d'une adresse virtuelle est la suivante :

et le schéma ci-dessous explicite le mécanisme général du mapping :

le numéro de bloc b, combiné à une adresse de base située dans un registre, permet d'atteindre un élément de la table des blocs. Cet élément donne, entre autres informations, l'adresse d'un bloc en mémoire principale ; la partie d est la même pour l'adresse virtuelle et l'adresse physique : elle permet de trouver un octet particulier dans le bloc (adressage relatif).

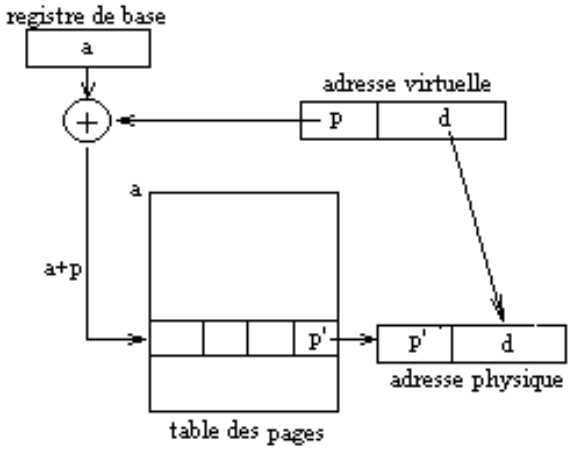


Pagination

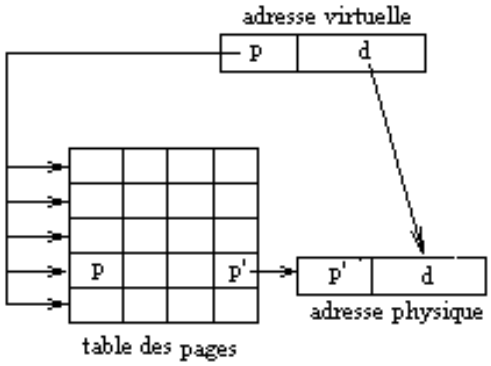
Les pages sont des blocs de taille fixe et l'adresse virtuelle est de la forme (p,d) où p est le numéro de page en mémoire virtuelle. Un élément de la table des pages sera de la forme (r, s, D, p') où r est le bit de résidence (0 si la page est en mémoire secondaire, 1 si la page est chargée en mémoire principale), s est l'adresse en mémoire secondaire (si r=0), p' est le numéro de page en mémoire principale (si r=1) ; D correspond aux droits d'accès de la page (lecture, écriture, exécution).

On peut distinguer trois types de mapping :

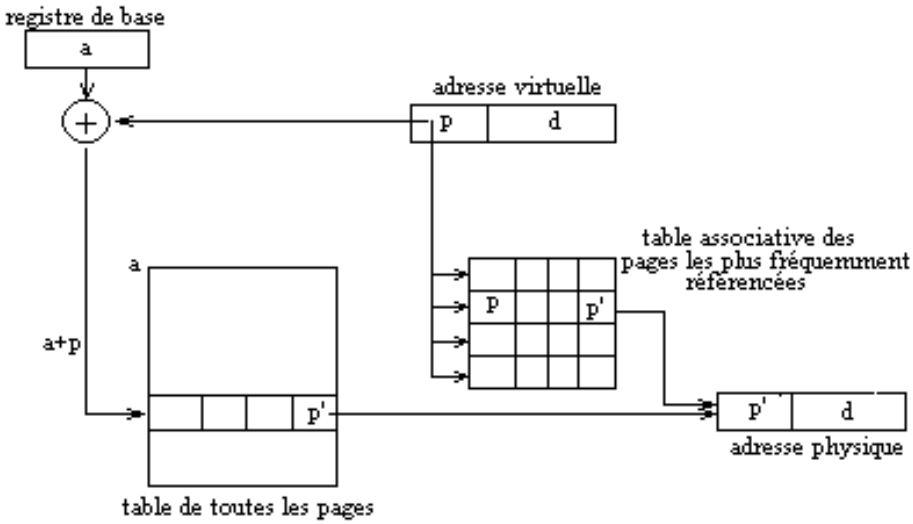
mapping "direct" : p est converti en p' d'après la table des pages.



mapping "associatif" : une mémoire associative est utilisée pour la table des pages (accès par contenu et non par adresse) ce qui permet d'augmenter la vitesse de traitement.

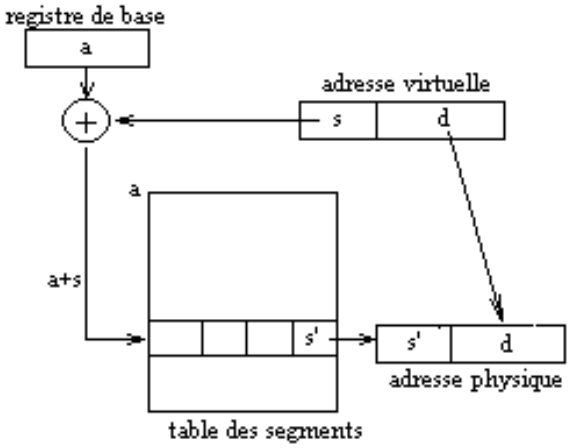


mapping "direct-associatif" : on essaie d'abord de trouver p' avec la table associative, sinon on utilise la table générale.



Segmentation

Un segment est un bloc de taille variable ; une adresse virtuelle est de la forme (s,d) où s est le numéro de segment. Le mécanisme de mapping est assez similaire à celui de la pagination. Un élément de la table des segments sera de la forme (r, s, l, D, s') où r , s, D, s' ont la même signification que précédemment (en remplaçant page par segment), l désigne la longueur du segment.



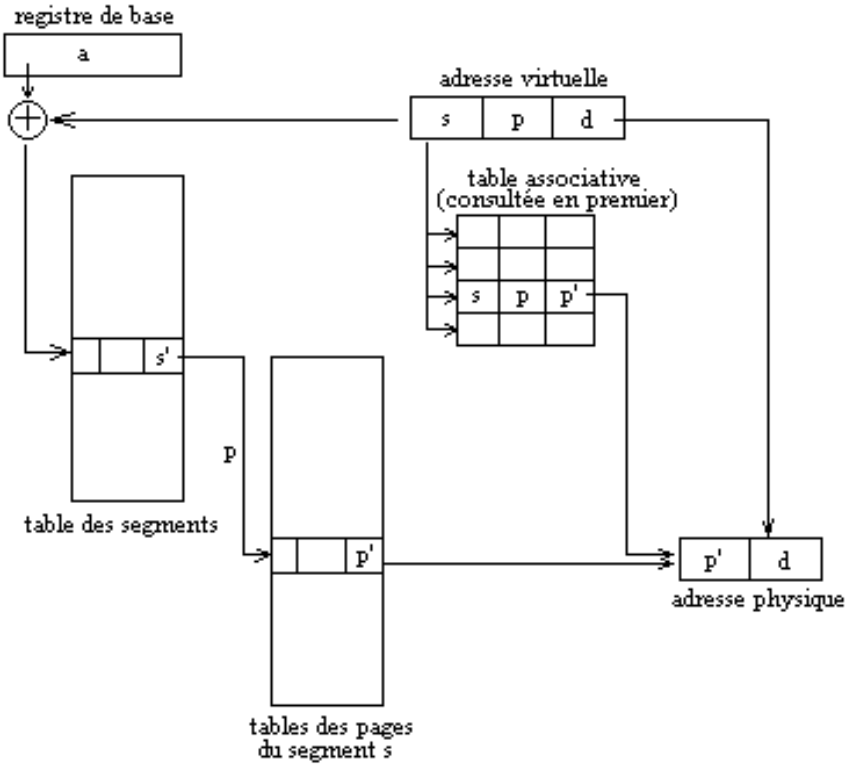
Systèmes mixtes

Il existe des systèmes mêlant segmentation et pagination.

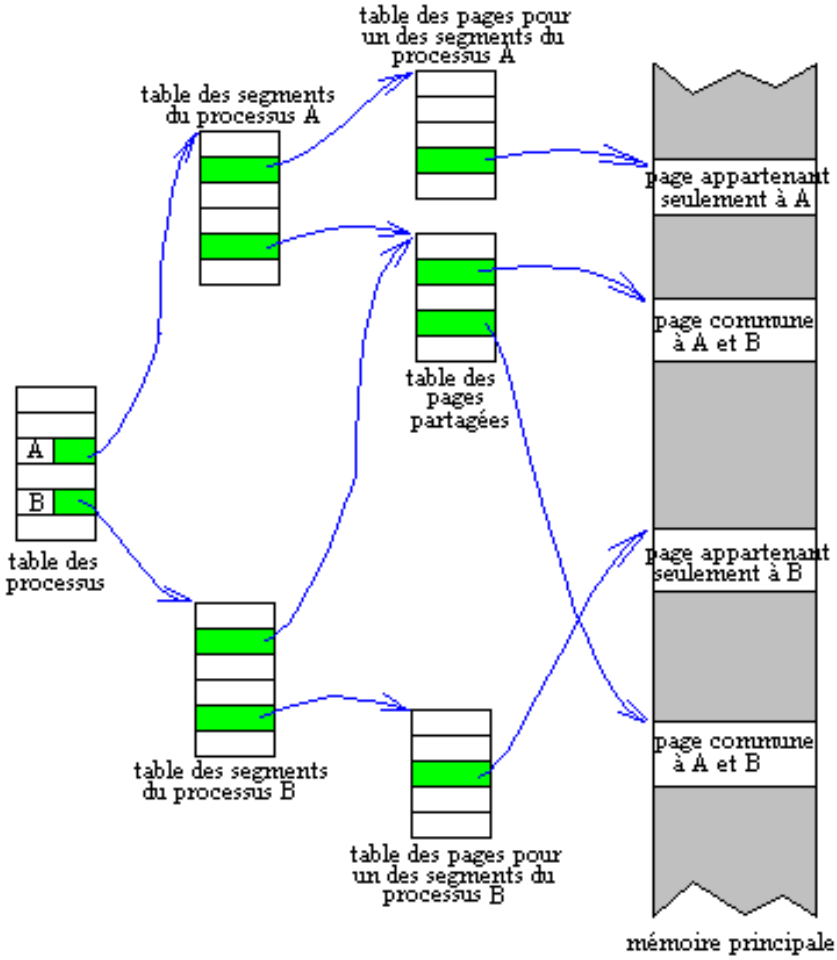
exemple : segment 1 = 4 pages ; segment 2 = 2 pages, etc...

Une adresse virtuelle est alors de la forme (s, p, d) où s est le numéro de segment, p le numéro de page dans le segment et d le numéro d'octet dans la page (déplacement).

On essaie d'abord la recherche avec la table associative, sinon on utilise les tables segments-pages.



Le schéma ci-dessous montre comment est effectué le partage de zones mémoire dans un système segmenté-paginé :



Gestion de la mémoire virtuelle

Stratégies de remplacement en pagination

Lorsque le système est en fonctionnement, la mémoire principale contient des pages chargées ; au bout d'un moment, elle peut être saturée ; pour charger une nouvelle page, il faut en "écraser" une présente. En général c'est le principe d'optimalité qui guide ce remplacement : on remplace les pages qui ne sont plus utilisées, mais, bien entendu, ceci n'est qu'un voeu car il est difficile de prédire qu'une page ne sera plus utilisée par la suite. C'est pourquoi, un certain nombre de stratégies ont été mises au point pour s'approcher au plus près du principe d'optimalité.

- **random**

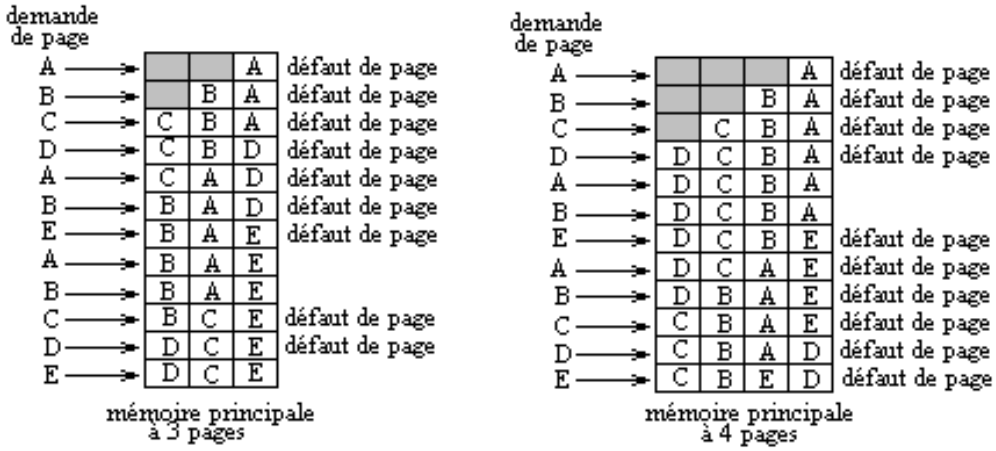
La stratégie est simple : on tire au sort les pages qui doivent être remplacées. Quoique très démocratique, cette stratégie est rarement utilisée.

- **FIFO**

Chaque page qui vient d'être chargée est affectée d'une date (de chargement). En se basant sur cette date, il est possible d'éliminer les pages les plus anciennes. Telle est la stratégie FIFO (First In First Out). Cette méthode rencontre deux problèmes :

a) il est difficile de garder des pages (très utilisées) de manière permanente.

b) anomalie FIFO ; 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.



- **LRU**

La stratégie LRU (Least Recently Use) consiste à remplacer la page la moins récemment utilisée. C'est donc ici l'utilisation et non le chargement qui guide la politique de remplacement. Bien entendu, il faut estampiller chaque page avec une date chaque fois qu'elle est utilisée. Le problème que l'on rencontre dans cette stratégie est celui des traitements faisant intervenir des boucles de programme.

- **LFU**

La stratégie LFU (Least Frequently Use) est plus affinée que la précédente ; ici c'est a fréquence d'utilisation des pages qui est pris en compte, ce qui permet (théoriquement) de résoudre le problème des boucles. On remplace donc la page la moins fréquemment utilisée ; une page sera donc munie d'un compteur du nombre d'utilisations de cette page. Bien évidemment, une page qui vient d'être chargée possède une probabilité importante d'être remplacée.

- **NUR**

La stratégie NUR (Not Used Recently) est assez populaire et a pour base l'idée suivante : une page qui n'a pas été utilisée récemment a peu de chances de l'être ultérieurement. Pour mettre en oeuvre la stratégie NUR, 2 bits sont affectés à chaque page : r_b (bit de référence) et m_b (bit de modification ou dirty bit) :

r_b = 0 si la page n'a pas été référencée et r_b = 1 si la page a été référencée
m_b = 0 si la page n'a pas été modifiée et m_b = 1 si la page a été modifiée

Au départ, $r_b = m_b = 0$; l'algorithme de remplacement de page est le suivant :

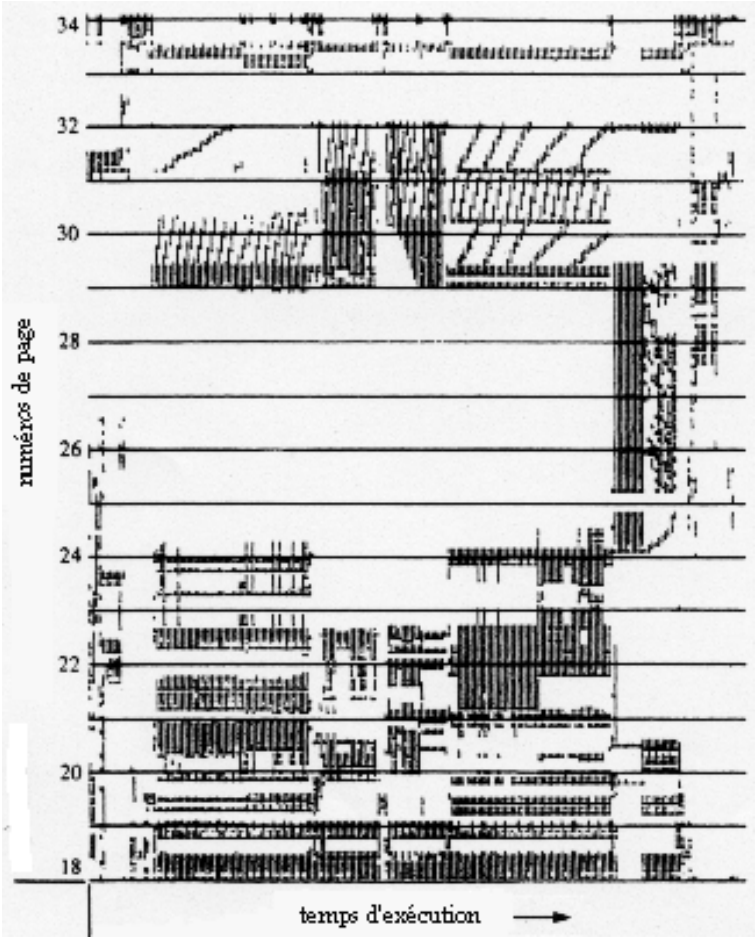
```
Chercher une page avec r_b = 0 (page non modifiée
Si on trouve alors
   Si m_b = 0 (page non modifiée) alors
      Remplacer la page
   sinon
      Continuer la recherche
   FinSi
sinon
   Remplacer la page référencée
FinSi
```

Périodiquement, on remet r_b à 0, ce qui signifie que, à un instant donné, il y a 4 catégories de pages :

$r_b = 0$ et $m_b = 0$ $r_b = 0$ et $m_b = 1$	pages ayant une plus grande probabilité d'être remplacées
$r_b = 1$ et $m_b = 0$ $r_b = 1$ et $m_b = 1$	pages ayant une probabilité plus faible d'être remplacées

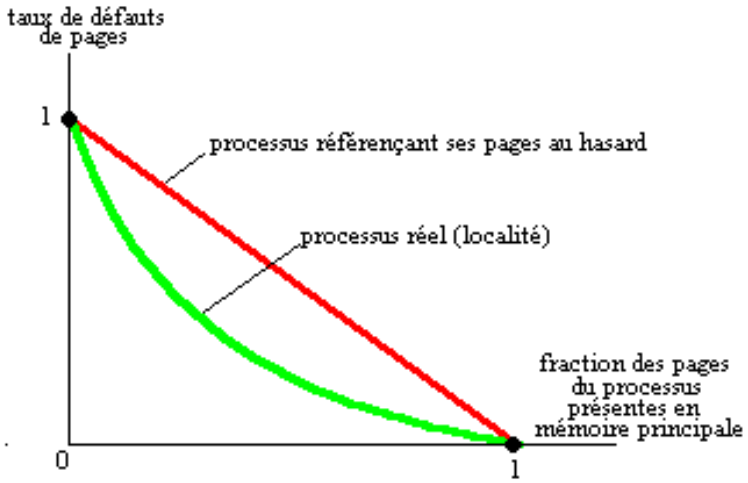
Localité

L'observation montre que les références aux informations situées en mémoire principale n'ont pas une distribution uniforme mais présentent une localité spatiale et temporelle.



On constate, sur le relevé ci-dessus que l'occupation spatio-temporelle de la mémoire est loin d'être homogène.

Si l'on étudie la corrélation entre le taux de défaut de pages et la fraction de pages d'un processus présentes en mémoire principale, on obtient des courbes du genre indiqué ci-dessous :

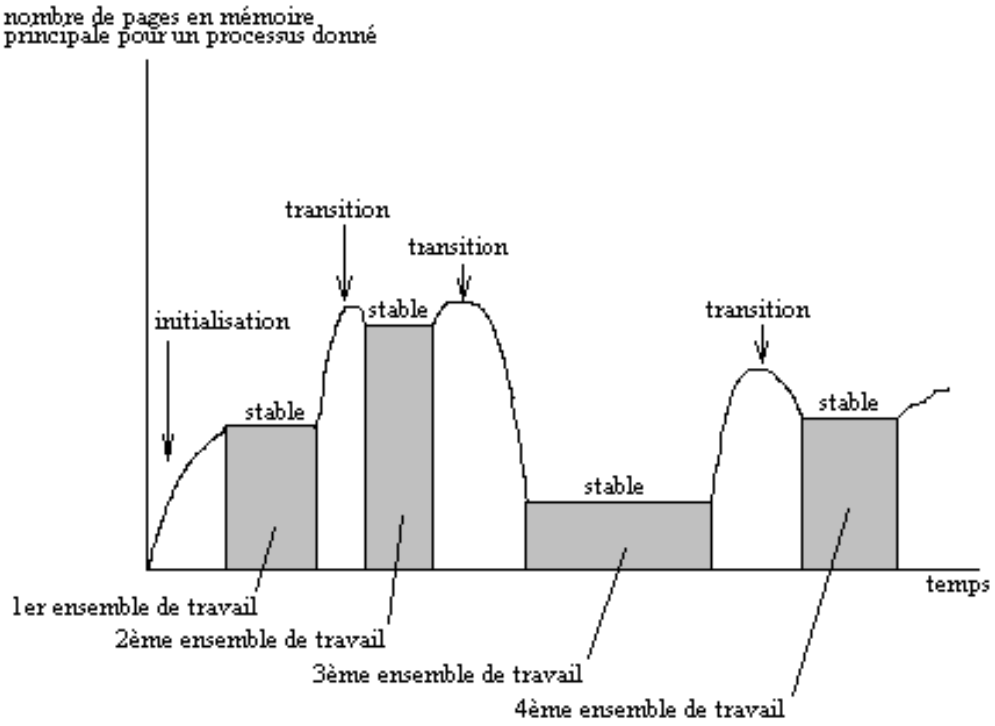


L'idéal serait d'avoir, pour chaque processus, toutes les pages correspondantes en mémoire ; ceci n'est évidemment pas possible car la mémoire principale a une taille finie alors que le nombre de processus et donc le nombre de pages est indéfini. Cependant, il est possible d'après les travaux de Dennery (1968), de placer en mémoire, pour chaque processus, un ensemble de pages permettant l'avancement des exécutions. Ces ensembles de pages sont des ensembles de travail (Worksets). n peut en donner plusieurs définitions :

- définition formelle : un ensemble de travail est un ensemble de pages activement référencées par un processus ; pour une efficacité optimale, l'ensemble de travail doit se trouver en mémoire principale.
- définition idéale : un ensemble de travail est un ensemble de pages qui doivent être mises en mémoire principale pour que le processus s'exécute efficacement.
- définition physique : un ensemble de travail W(t, dt) d'un processus au temps t est l'ensemble des pages

référencées entre t-dt et t où dt est la taille de la fenêtre de l'ensemble de travail.

Au cours de son exécution, un processus peut travailler avec plusieurs ensembles de travail successifs qu'il faut, à chaque étape, constituer (par chargement de pages). On a donc une succession d'exécutions et de transitions.



Stratégies de recherche

Il y a deux stratégies principales :

à la demande

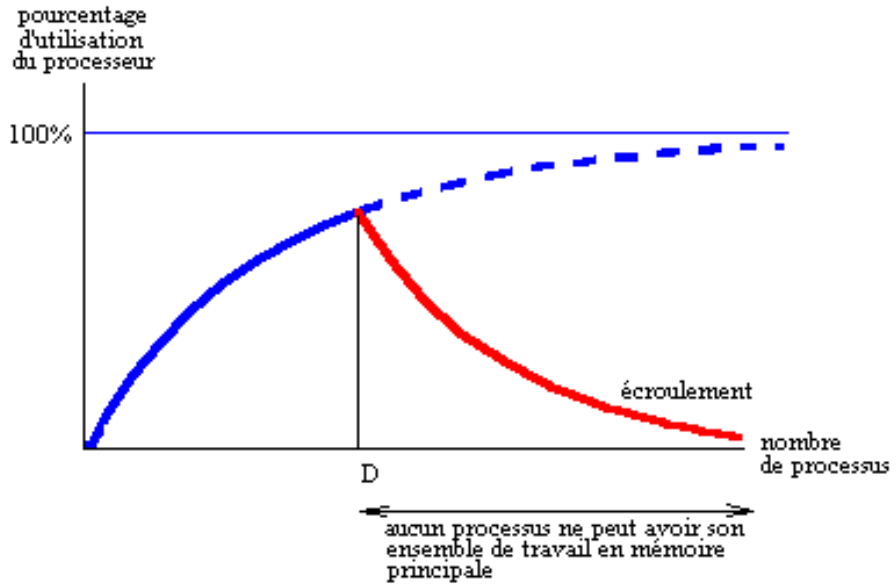
On charge une page quand on en a besoin. Il est en effet difficile de prévoir les pages dont on aura besoin ultérieurement. Ne sont chargées en mémoire principale que les pages nécessaires. L'inconvénient de cette stratégie simple est évidemment le taux quelquefois élevé de défaut de pages qui obligent à une interruption pour chargement.

anticipation (prefetch)

Cette stratégie est plus complexe car le système d'exploitation doit "deviner" quelles sont les pages qui seront ultérieurement utilisées par un processus. Si le processus est séquentiel, cela ne pose pas trop de problèmes. Si le processus contient de nombreux déroutements et sauts de programmes, cela est beaucoup plus délicat.

Ecroulement d'un système

Plusieurs processus utilisent le même processeur. Pour un fonctionnement efficace, chaque processus doit avoir en mémoire principale son ensemble de travail. Quand la nombre de processus est élevé, ceci n'est plus possible et on constate un écroulement du système (plus aucun programme ne s'exécute).



Pour éviter l'écroulement il faut respecter la règle suivante :

$$\sum_{i=1}^{N} \text{taille } (W_i) \leq \text{taille mémoire}$$

où W_i est l'ensemble de travail du processus i.

exemple d'écroulement : Soit une mémoire principale de 7 pages ; deux processus sont présents dans le système : A qui possède un ensemble de travail de 4 pages et B qui possède un ensemble de travail de 4 pages également. On constate que la règle précédente est violée.

Au début, les pages des ensembles de travail sont chargées, puis il arrive une phase d'écroulement où chaque chargement de page d'un processus écrase une page (indispensable) de l'autre.