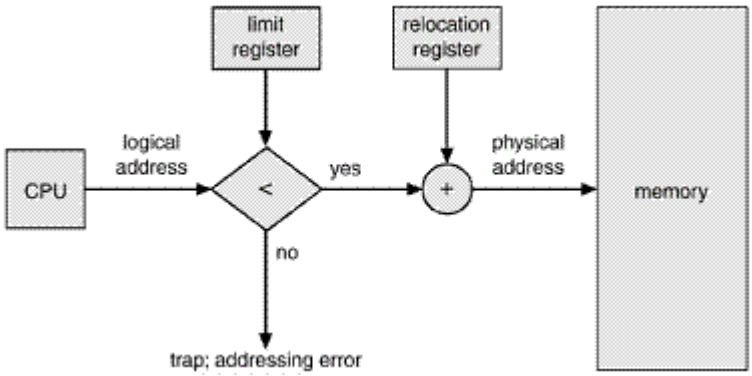
Systèmes d'Exploitation

Gestion de la mémoire

Version 1



Y. Challal, H. Bettahar, M. Vayssade

Table des matières

Objectifs *5*

I - Gestion de la mémoire *7*

A. Organisation de la mémoire : monoprogrammation vs. multiprogrammation.....*7*

B. Multiprogrammation et partitions multiples contigües.....*8*

1. Partitions contiguës fixes.....8
2. Partitions contiguës dynamiques.....10
3. Partitions contiguës Siamoises (Buddy system).....11
4. Ré-allocation et protection.....11
5. Va et vient (Swap).....12
6. Fragmentation et Compactage.....13

C. Multiprogrammation et partitions multiples non contigües.....*14*

1. Pagination.....14
2. Segmentation.....17
3. Segmentation paginée.....18

D. Mémoire virtuelle.....*19*

1. Concept de mémoire virtuelle.....19
2. Overlays (segments de recouvrement).....20
3. Pagination à la demande.....20
4. Algorithmes de remplacement de page.....22
5. Gestion Mémoire dans le sytème Linux [bouzefrane03].....24

Objectifs

- Analyser la fonction de gestion de la mémoire
- Analyser la mise en œuvre de la mémoire virtuelle

Gestion de la mémoire

I

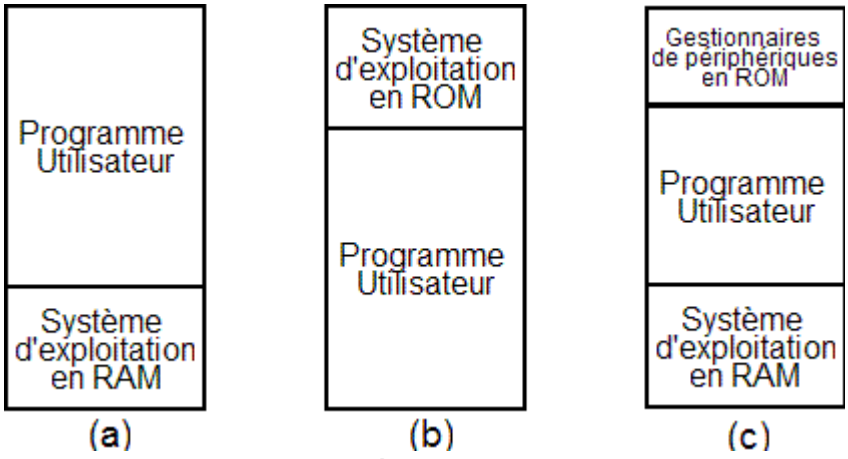
Organisation de la mémoire : monoprogrammation vs. multiprogrammation	7
Multiprogrammation et partitions multiples contigües	8
Multiprogrammation et partitions multiples non contigües	14
Mémoire virtuelle	19

La gestion de la mémoire est l'une des tâches fondamentales d'un système d'exploitation. Un programme ne peut s'exécuter que si ses instructions et ses données sont en mémoire physique. Donc, si on désire exécuter plusieurs programmes simultanément dans un ordinateur, il faudra que tous ces programmes soient chargés dans la mémoire. Le système d'exploitation devra donc allouer à chaque programme une zone de la mémoire où celui-ci sera chargé. 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 inconnus du système à l'avance. Chaque fois qu'un utilisateur va demander le lancement d'un programme, le système devra trouver une place dans la mémoire pour le charger. Le fait de travailler en multiprogrammation implique donc que les programmes d'application sont chargés dans des zones de la mémoire dont la localisation n'est pas connue d'avance. Le système prend à sa charge la gestion de la mémoire principale, il assure sa protection, et il fournit l'information qui permet à chaque programme de s'exécuter sans savoir à l'avance dans quelle zone mémoire il sera au moment de son exécution (cette zone peut d'ailleurs changer entre différentes exécutions et même en cours d'exécution). Le but d'une bonne gestion de la mémoire est d'augmenter le rendement global du système. La question principale à laquelle le système devra répondre chaque fois qu'un nouveau programme demande à être exécuté sera donc "à quel endroit dans la mémoire ?"

A. Organisation de la mémoire : monoprogrammation vs. multiprogrammation

Organisation de la mémoire en monoprogrammation

A la base, le rôle du plus simple gestionnaire de mémoire est d'exécuter un seul programme à la fois, en partageant la mémoire entre le programme et le système d'exploitation. Trois variantes de l'organisation possible sont montrées à la figure (cf. 'Organisation de la mémoire en monoprogrammation' p 8) suivante (cf. 'Organisation de la mémoire en monoprogrammation' p 8) *[tanenbaum03]*.



Organisation de la mémoire en monoprogrammation

Autrefois utilisé dans les mainframes et les mini-ordinateurs, le premier modèle (fig. (cf. 'Organisation de la mémoire en monoprogrammation' p 8)(a)) est aujourd'hui rare. Le deuxième modèle (fig. (cf. 'Organisation de la mémoire en monoprogrammation' p 8)(b)) équipe plusieurs ordinateurs de poche ou systèmes embarqués. Enfin, le troisième modèle (fig. (cf. 'Organisation de la mémoire en monoprogrammation' p 8)(c)) était exploité sur les premiers ordinateurs personnels (ceux que faisait fonctionner MS-DOS), dans lesquels la partie du système d'exploitation en ROM s'appelait le BIOS. Quand le système est organisé de cette manière, un seul processus peut s'exécuter à la fois. Dès que l'utilisateur tape une commande, le système d'exploitation copie le programme demandé depuis le disque vers la mémoire et l'exécute.

Gestion de la mémoire et multiprogrammation

Excepté sur des systèmes embarqués simples, on n'a plus tellement recours à la monoprogrammation de nos jours. La plupart des systèmes modernes autorisent l'exécution de processus multiples en même temps, ce qui implique le séjour de plusieurs programmes en même temps en mémoire et c'est cette technique qui a donné naissance à la gestion moderne de la mémoire. Pour supporter la multiprogrammation et donc l'existence de plusieurs processus en mémoire principale, on peut distinguer deux grandes stratégies d'allocation mémoire: l'allocation de partitions contiguës et l'allocation de partitions non-contiguës.

B. Multiprogrammation et partitions multiples contigües

L'espace mémoire est divisé en partitions. Chaque partition peut être allouée à un programme. La taille des partitions et donc leur nombre peuvent être fixe ou variable.

1. Partitions contiguës fixes

Table de description des partitions

Dans cette stratégie la mémoire est divisée en n partitions de tailles fixes (si possible inégales). Ce partitionnement se fait au démarrage du système. Le système d'exploitation maintient une table de description des partitions.

Exemple : Table de description des partitions

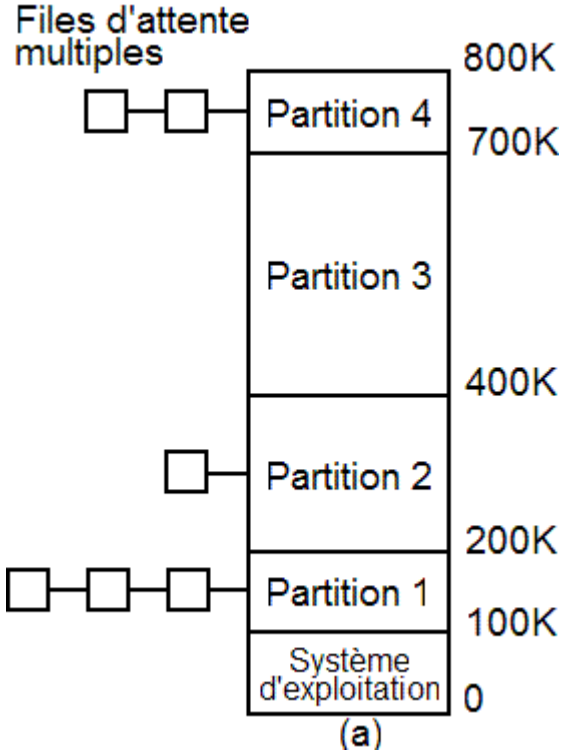
Le tableau suivant est un exemple de table de description de partitions:

Adresse	Taille	Etat
312k	8K	allouée
320k	32K	allouée
352k	32K	libre
384k	120K	libre
504k	520K	allouée

Tableau 1 : Table de description de partitions

Méthode : Allocation de partitions fixes : files d'attente séparées

Quand un processus arrive, il peut être placé dans la file d'attente des entrées de la plus petite des partitions assez larges pour le contenir. La figure (cf. 'Partitions contigües fixes : files d'attente séparées' p 9) illustre ce type de systèmes à partitions fixes.

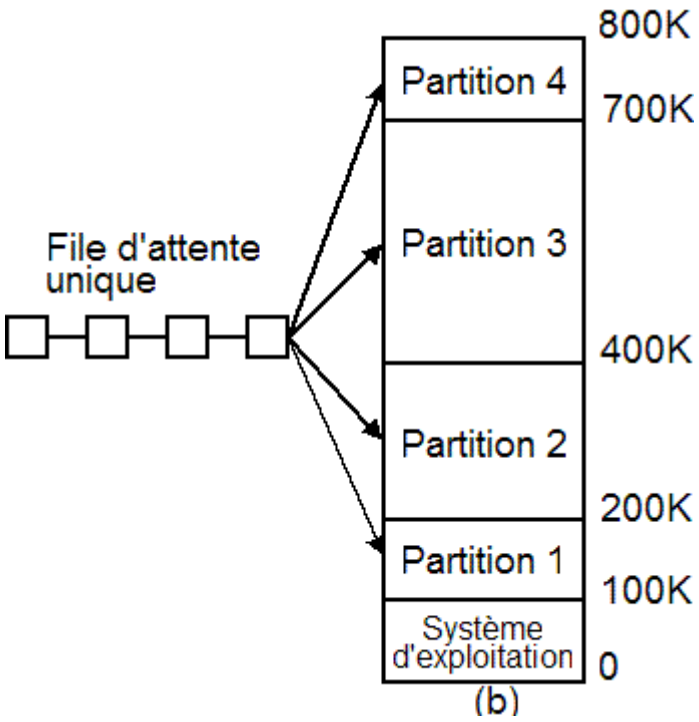


Partitions contigües fixes : files d'attente séparées

Le tri des processus qui arrivent en différentes files d'attente présente des inconvénients lorsque la file d'attente pour une grande partition est vides tandis que celle d'une petite partition est pleine, comme l'illustrent les partitions 1 et 3 de la figure (cf. 'Partitions contigües fixes : files d'attente séparées' p 9) [fig:memoire:partitions-fixes-a] (cf. 'Partitions contigües fixes : files d'attente séparées' p 9) *[tanenbaum03]*.

Méthode : Allocation de partitions fixes : file d'attente commune

Une autre organisation possible consiste à gérer une seule file d'attente (cf. 'Partitions fixes contigües : file d'attente commune' p 10), comme la figure. Dès qu'une partition devient libre, toute tâche placée en tête de file d'attente et dont la taille convient peut être chargée dans cette partition vide et exécutée.



Partitions fixes contigües : file d'attente commune

2. Partitions contiguës dynamiques

Partitions dynamiques

Dans cette stratégie la mémoire est partitionnée dynamiquement selon la demande. Lorsqu'un processus se termine sa partition est récupérée pour être réutilisée (complètement ou partiellement) par d'autres processus.

Méthode : Stratégies d'allocation

Le lancement des processus dans les partitions se fait selon différentes stratégies. Pour cela le gestionnaire de la mémoire doit garder trace des partitions occupées et /ou des partitions libres. On distingue les stratégies de placement suivantes:

1. **Stratégie du premier qui convient (First Fit):** La liste des partitions libres est triée par ordre des adresses croissantes. La recherche commence par la partition libre de plus basse adresse et continue jusqu'à la rencontre de la première partition dont la taille est au moins égale à celle du processus en attente.
2. **Stratégie du meilleur qui convient (Best Fist):** On alloue la plus petite partition dont la taille est au moins égale à celle du processus en attente. La table des partitions libres est de préférence triée par tailles croissantes.
3. **Stratégie du pire qui convient (Worst Fit):** On alloue au processus la partition de plus grande taille.

Exemple : Allocation de partitions dynamiques contigües

Soit une MC dont la table des partitions libres est la suivante :

Adresse	Taille
352K	32K
504K	520K

Tableau 2 : Partitions libres

Soit les demandes suivantes P4(24K), P5(128K), P6(256K) .

Evolution de la table des partitions libres:

First Fit:

init	P4	P5	P6
32K	8K	8K	8K
520K	520K	392K	136K

Best Fit:

init	P4	P5	P6
32K	8K	8K	8K
520K	520K	392K	136K

Worst Fit:

init	P4	P5	P6
32K	32K	32K	32K

Tableau 3 : Allocation de partitions dynamiques contiguës selon stratégies différentes

3. Partitions contiguës Siamoises (Buddy system)

Méthode : Partitions Siamoises

C'est un compromis entre partitions de tailles fixes et partitions de tailles variables. La mémoire est allouée en unités qui sont des puissances de 2. Initialement, il existe une seule unité comprenant toute la mémoire. Lorsque de la mémoire doit être attribuée à un processus, ce dernier reçoit une unité de mémoire dont la taille est la plus petite puissance de 2 supérieure à la taille du processus. S'il n'existe aucune unité de cette taille, la plus petite unité disponible supérieure au processus est divisée en deux unités "siamoises" de la moitié de la taille de l'original. La division se poursuit jusqu'à l'obtention de la taille appropriée. De même deux unités siamoises libres sont combinées pour obtenir une unité plus grande.

Exemple

Action	Mémoire				
Au départ	1Mo				
A, 150 ko	A	256ko			512ko
B, 100 ko	A	B	128ko		512ko
C, 50 ko	A	B	C	64ko	512ko
B se termine	A	128ko	C	64ko	512ko
C se termine	A	256ko			512ko
A se termine	1Mo				

Tableau 4 : Buddy system

4. Ré-allocation et protection

Réallocation et translation d'adresse

La multiprogrammation introduit deux problèmes essentiels à résoudre: la ré-allocation et la protection. Examinons cela à partir de la figure (cf. 'Partitions fixes contigües : file d'attente commune' p 10). Dans cette figure, il apparaît clairement que plusieurs processus s'exécutent à différentes adresses. Quand le programme est lié (programme principal, sous-programmes utilisateur, et sous programmes des bibliothèques ne forment qu'un seul espace d'adressage), l'éditeur de lien doit

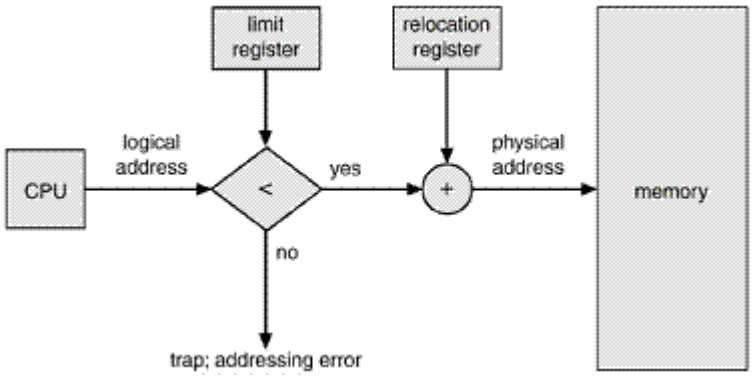
savoir à quelle adresse le programme démarrera en mémoire: supposons que la première instruction corresponde à l'appel d'une procédure à l'adresse absolue 100 dans le fichier binaire produit par l'éditeur de lien. Si le programme est chargé dans la partition 1 (à l'adresse 100K), cette instruction sautera à l'adresse absolue 100, laquelle se trouve dans la zone du système d'exploitation. Ce qui est nécessaire, c'est l'adresse 100K+100. Si le programme est chargé dans la partition 2, il aura besoin d'une adresse à 200K+100, et ainsi de suite. Ce problème est appelé problème de **ré-allocation** ou de **translation d'adresse** *[tanenbaum03]*.

Protection

Dans les systèmes multi-utilisateurs, il est déconseillé d'autoriser des processus à lire ou à écrire dans la mémoire appartenant à d'autres utilisateurs (et a fortiori appartenant au système d'exploitation). Ce problème est appelé problème de **protection**.

Méthode : Registres de base et de limite

Une solution aux deux problèmes de la ré-allocation et de la protection consiste à équiper l'ordinateur avec deux registres matériels particuliers, appelés registres de **base** et de **limite**. Quand un processus est activé, le registre de base est chargé avec l'adresse de départ de la partition, et le registre de limite est chargé avec la longueur de la partition. Les adresses sont comparées avec la valeur du registre de limite, afin d'assurer qu'elles ne référenceront pas une adresse hors de la partition courante. La valeur du registre base est ajoutée à toutes les adresses mémoire générées automatiquement avant qu'elles ne soient envoyées en mémoire. Ainsi, si le registre de base contient la valeur 100K, une instruction comme CALL 100 se transforme effectivement en CALL 100K+100. Le matériel protège les registres de base et de limite afin d'empêcher les programmes utilisateur de les modifier. La figure (cf. 'Registres de base et limite' p 12) illustre ce mécanisme de ré-allocation et de protection.



Registres de base et limite

5. Va et vient (Swap)

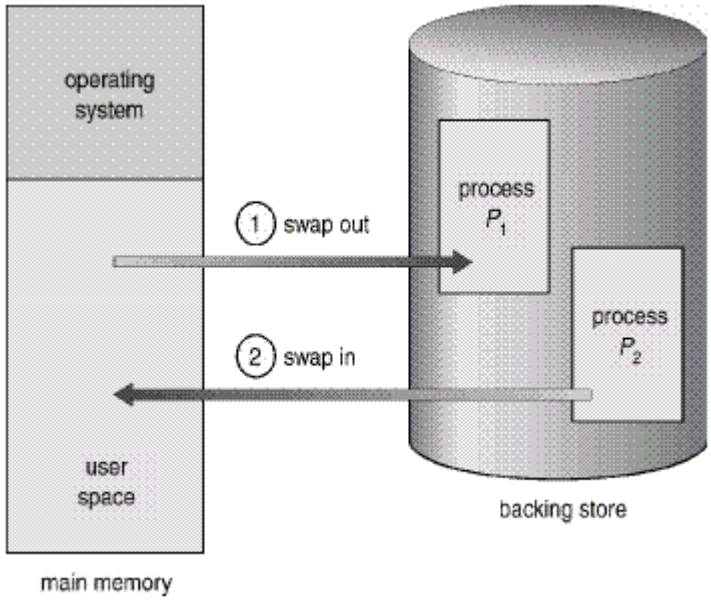
Multi-programmation et limite de la mémoire principale

Dans un système à temps partagé, parfois la mémoire principale est insuffisante pour maintenir tous les processus courants actifs : il faut alors conserver les processus supplémentaires sur un disque et les charger pour qu'ils s'exécutent dynamiquement *[tanenbaum03]*.

Méthode : Swap

La stratégie la plus simple, appelée **va-et-vient** (*swap*) (cf. 'Swap' p 13), consiste à considérer chaque processus dans son intégralité: en cas de réquisition du

processeur, le programme en cours doit être sauvegardé sur disque avant le chargement en mémoire principale de son successeur, dans sa totalité, pour exécution.



Swap

6. Fragmentation et Compactage

Définition : Fragmentation interne

Soient un programme de taille M et une partition de taille N (partition fixe). Si la partition est allouée au programme avec N>M alors la partie non occupée par le programme est appelée fragmentation interne.

Définition : Fragmentation externe

Soit un programme de taille M. Si toute partition libre est de taille Pi telle que Pi<M alors le programme ne peut être chargé dans aucune partition libre alors que :

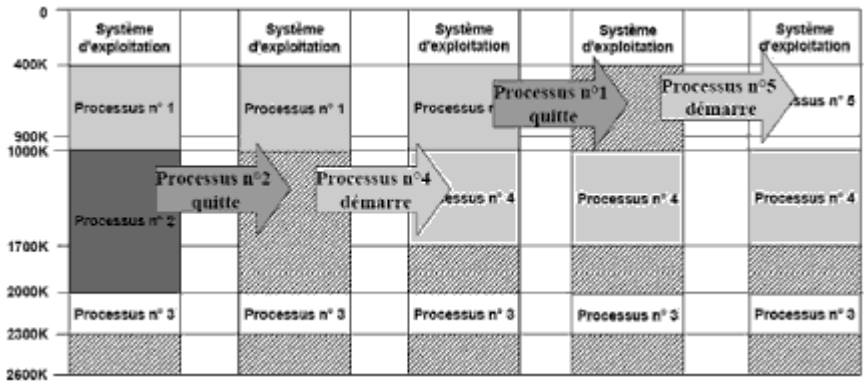
$$\sum_i P_i \geq M$$

Formule 1 : Fragmentation externe

On dit qu'il y a fragmentation externe. La mémoire est fragmentée en un grand nombre de petits trous.

Exemple : Fragmentation de la mémoire

La figure (cf. 'Fragmentation mémoire' p 14) illustre le phénomène de fragmentation de la mémoire.



Fragmentation mémoire

Méthode : Compactage de la mémoire

Les zones libres sont regroupées. C'est une technique très coûteuse, et en plus elle change les adresses de chargement des processus.

Remarque : Compactage et réallocation

Le compactage de la mémoire n'est possible que s'il y a un mécanisme de ré-allocation dynamique des processus (basé sur l'utilisation des registres de translation d'adresse (registre base))

C. Multiprogrammation et partitions multiples non contigües

La traduction dynamique d'adresse réalisée par l'emploi d'un registre de base impose que les programmes soient chargés en mémoire principale dans des cellules contiguës. Nous avons vu que les techniques d'allocation associées conduisent à la fragmentation de la mémoire. Pour l'éviter, il faut pouvoir implanter un programme dans plusieurs zones non contiguës *[crocus75]*. Plusieurs techniques proposent d'allouer des espaces mémoire non-contiguës pour les processus : Pagination et Segmentation.

1. Pagination

Définition : Pagination

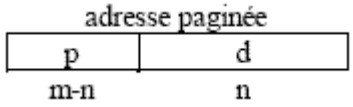
La solution apportée par les mécanismes de pagination consiste à découper l'espace adressable, ou **espace virtuel**, en zones de taille fixe appelée **pages**. La mémoire réelle est également découpée en **cases** (*frame* en anglais) ayant la taille d'une page de sorte que chaque page peut être implantée dans n'importe quelle case de la mémoire réelle *[crocus75]*.

Méthode : Pagination à un niveau

Une adresse est divisée en deux parties : un numéro de page p et un déplacement à l'intérieur de la page d. La taille de la page (et donc de la case) est une puissance de 2 variant généralement entre 512 et 8192 octets selon les architectures. Le choix de tailles de puissance 2 permet de simplifier la division nécessaire pour calculer le numéro de page et le déplacement à partir d'une adresse logique, ainsi :

Si la taille de l'espace d'adressage logique est 2^m et la taille d'une page est 2^n (octets ou mots) (voir figure (cf. 'Structure d'une adresse virtuelle' p 15):

- Les m-n bits de poids fort d'une adresse paginée désignent le numéro de page ; et
- Les n bits de poids faible désignent le déplacement dans le page.



Structure d'une adresse virtuelle

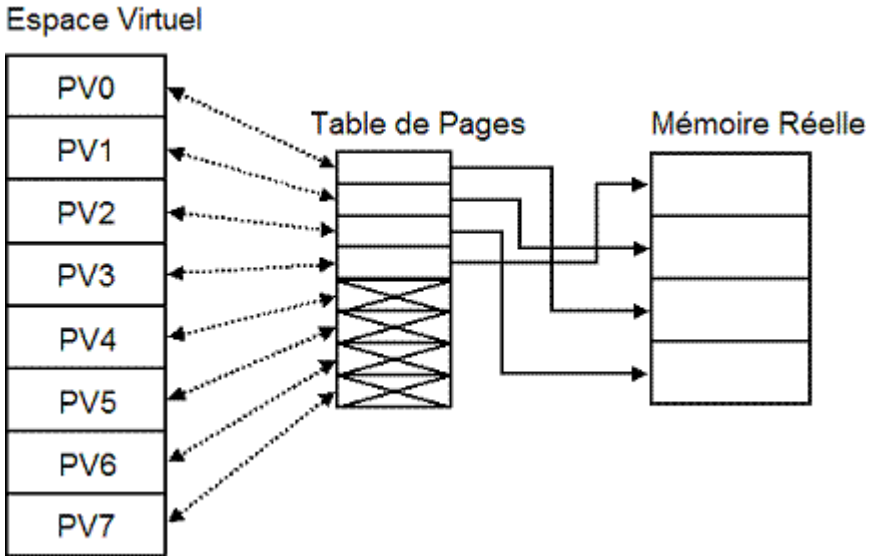
Si T est la taille d'une page et A une adresse logique, l'adresse paginée (p, d) est déduite à partir des formules :

- p = A div T
- d = A modulo T

et l'adresse logique est égale à : A = p*T + d

Méthode : Table des pages

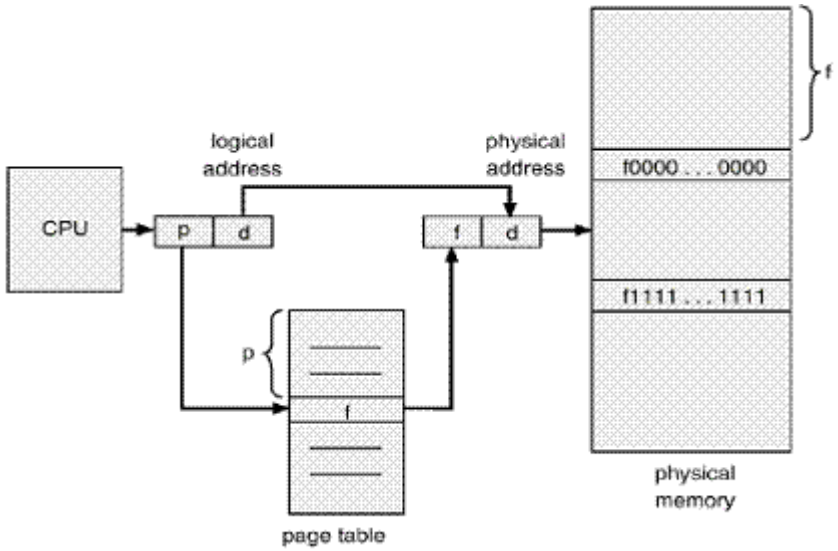
La traduction des adresses utilise une [table des pages] (cf. 'Correspondance entre espace virtuel et espace réel [crocus75]' p 15) qui est située en mémoire centrale ou dans des registres, et dans laquelle les entrées successives correspondent aux pages virtuelle consécutives. La p-ième entrée de la table des pages contient le numéro r de la case où est implantée la pase p, et éventuellement des indicateurs supplémentaires.



Correspondance entre espace virtuel et espace réel [crocus75]

Méthode : Traduction d'adresse

L'adresse réelle (f,d) d'un mot d'adresse virtuelle (p,d) est obtenue en remplaçant le numéro de page p par le numéro de case f trouvé dans la p-ième entrée (voir figure (cf. 'Traduction d'une adresse virtuelle en une adresse réelle' p 16)).



Traduction d'une adresse virtuelle en une adresse réelle

Exemple : Mémoire topographique du CII 10070 [crocus75]

Dans la machine CII 10070, une adresse virtuelle occupe 17 bits, ce qui permet d'adresser 128K mots. La taille maximale de la mémoire réelle est également de 128K mots. L'espace virtuel est découpé en 256 pages de 512 mots. Les 8 bits de gauche de l'adresse virtuelle fournissent le numéro de page et les 9 bits de droite le déplacement dans la page. La traduction dynamique des adresses est réalisée par une "mémoire topographique" constituée de 256 registres (de 8 bits) pouvant contenir chacun un numéro de case. Un verrou d'accès virtuel de 2 bits est associé à chaque entrée de la mémoire topographique; sa signification est la suivante:

- 00 : tout accès autorisé,
- 01 : écriture interdite,
- 10 : lecture autorisée,
- 11 : aucun accès permis.

Remarque : Optimisation de la taille des tables de pages

Pour éviter d'avoir des tables de pages trop longues, des tables de plusieurs niveaux (hiérarchiques (cf. 'Table de pages hiérarchique' p 17)) peuvent être utilisés.

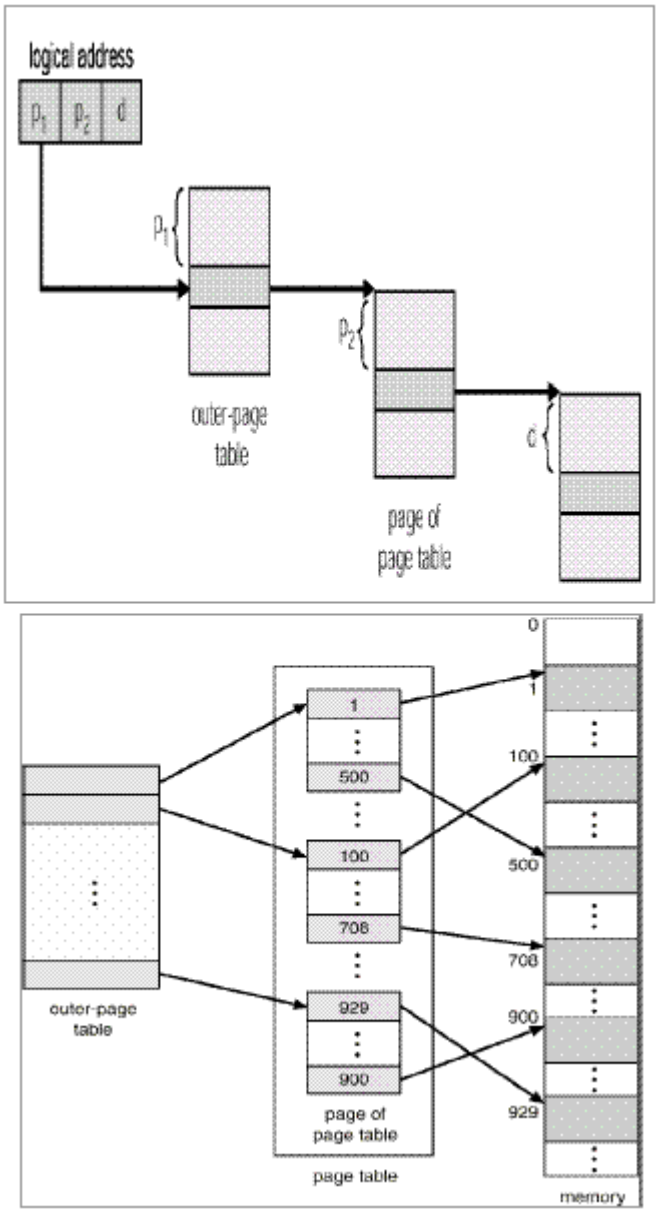


Table de pages hiérarchique

Remarque : Pagination et fragmentation

Avec la pagination, il n y a plus de fragmentation externes (s'il y a suffisamment de mémoire (de pages) non allouée à un programme il les aura), par contre la fragmentation interne persiste: si un programme nécessite 2 pages et 1 octet, il aura 3 pages.

2. Segmentation

Segmentation vs. pagination

La segmentation est analogue à la pagination sauf que la taille d'un segment est variable.

L'espace d'adressage logique est divisé en un ensemble de segments.

L'avantage de la segmentation par rapport à la pagination, est que les segments peuvent refléter une vision logique du programme. Par exemple chaque segment peut représenter un module de programme généré au moment de la compilation.

Exemple : Segmentation

Segment de code / Segment de données / Segment de pile.

Méthode : Table de segments

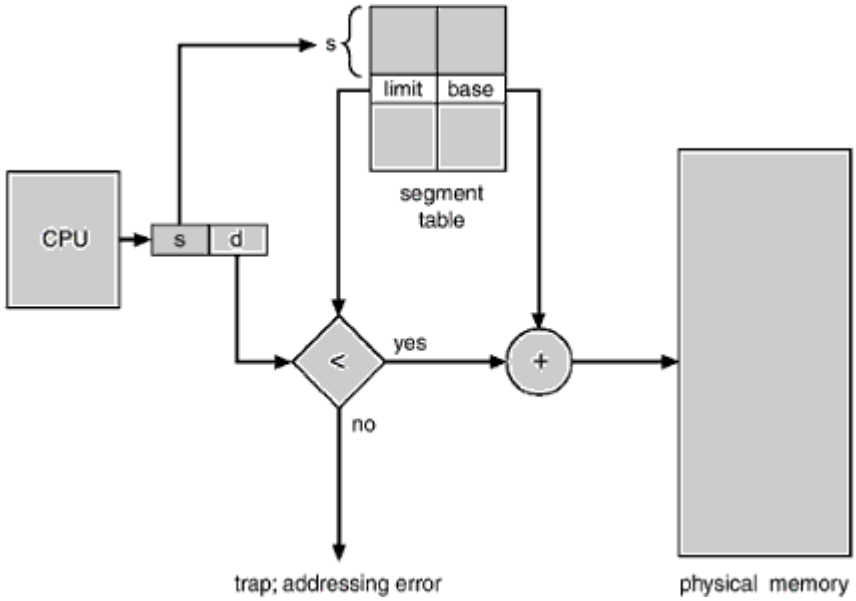
L'espace d'adressage physique correspondant peut ne pas être contigu. Une table de segment spécifie pour chaque segment deux valeurs :

- base : adresse début du segment en mémoire centrale
- limite : taille du segment

Une adresse logique est dite segmentée. Elle comprend un numéro de segment (s) et un déplacement dans le segment (d). (s) est utilisé comme index dans la table de segments. La table de segments est généralement implantée par des registres rapides. Si d > limite : alors erreur de débordement (fameux Segmentation fault).

Méthode : Traduction d'adresse

La figure (cf. 'Mémoire segmentée' p 18) (cf. 'Mémoire segmentée' p 18)illustre le mécanisme de traduction d'adresses segmentées et le mécanisme de protection.



Mémoire segmentée

Remarque : Segmentation et fragmentation externe

Avec la segmentation la taille des segments est variable ce qui peut engendrer une fragmentation externe. Ceci peut être résolu en combinant la segmentation et la pagination.

3. Segmentation paginée

Adressage en segmentation paginée

L'adressage est composé de deux niveaux :

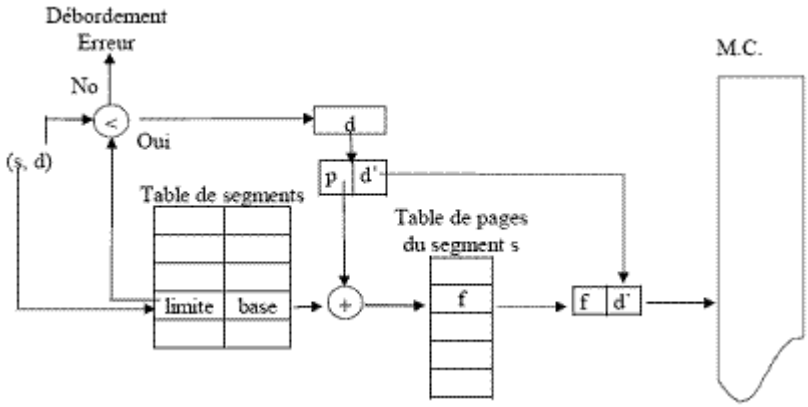
- Le niveau segment et
- Le niveau page

Chaque segment est un espace linéaire d'adressage logiques. Il est découpé en pages. Une adresse est formée de trois parties : (s, p, d')

- s : numéro du segment
- p : numéro de page
- d' : déplacement dans la page

Méthode : Table de page

La table de page est divisée en portions (cf. 'Segmentation paginée' p 19). Chaque portion concerne un segment bien précis. La base permet de retrouver la portion concernant le segment donné, et p donne le numéro de page dans cette portion.



Segmentation paginée

D. Mémoire virtuelle

La mémoire virtuelle est une technique permettant d'exécuter un programme partiellement chargé en mémoire centrale ; la taille du programme étant nettement supérieure à celle de la mémoire centrale.

1. Concept de mémoire virtuelle

Définition : Mémoire virtuelle

Pour un processus, la mémoire virtuelle est le support de l'ensemble des informations potentiellement accessibles. C'est donc plus précisément, l'ensemble des emplacements dont l'adresse peut être engendrée par le processeur.

L'allocation de mémoire consiste à concrétiser cette mémoire virtuelle par des supports physiques d'information tels que mémoire principale (RAM), disques magnétiques, etc. En dernier ressort, l'accès d'un processus à une information (de son espace virtuel) est concrétisé par l'accès d'un processeur physique à un emplacement de mémoire principale (RAM) adressable par ce processeur *[krakowiak87]*.

Fondamental

La **mémoire virtuelle** repose sur le principe suivant : la taille de l'ensemble formé par le programme, les données et la pile peut dépasser la capacité disponible de mémoire physique. Le système d'exploitation conserve les parties de programme en cours d'utilisation dans la mémoire principale, et le reste sur le disque.

Exemple

Un programme de 16 Mo peut s'exécuter sur une machine de 4 Mo de mémoire si les 4 Mo à garder en mémoire à chaque instant sont choisis avec attention, et que des parties du programme passent du disque à la mémoire, à la demande.

Remarque : Mémoire virtuelle et ES

Quand un programme attend le chargement d'une partie de lui-même, il est en attente d'E/S et ne peut s'exécuter; le CPU peut par conséquent être donné à un autre processus, de la même façon que pour tout autre système multiprogrammé *[tanenbaum03]*.

Méthode : Implémentation de la mémoire virtuelle

Il existe deux techniques permettant d'implanter une mémoire virtuelle. Les "overlays" et la pagination à la demande. Dans les deux cas le manque en mémoire centrale est compensé par la mémoire secondaire. Le principe des deux méthodes consiste à ne maintenir en mémoire centrale que les instructions et les données nécessaires à l'exécution d'un programme à un instant t.

2. Overlays (segments de recouvrement)

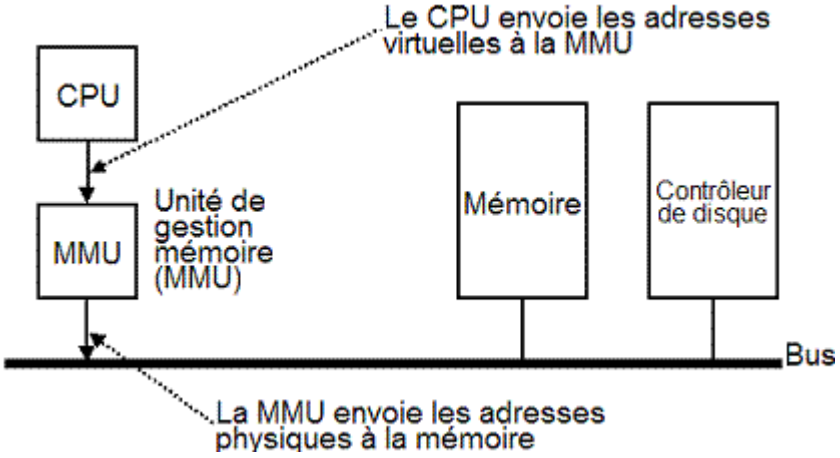
Méthode : Segments de recouvrement

Au début de l'informatique, il arrivait que les programmes soient trop importants pour être supportés par la mémoire disponible. La solution la plus courante consistait à diviser le programme en parties, appelées segments de recouvrement} ("overlays"). Le segment 0 était exécuté en premier. Quand cela était réalisé, il pouvait appeler un autre segment. Les segments de recouvrement étaient conservés sur le disque et chargés en mémoire par le système d'exploitation, dynamiquement à la demande *[tanenbaum03]*.

3. Pagination à la demande

Méthode : Memory Management Unit (MMU)

Dans un système paginé, les adresses générées par un programme sont appelées des adresses virtuelles et elles forment l'espace d'adressage virtuel. Ces adresses virtuelles ne vont pas directement sur le bus mémoire mais dans une unité de gestion mémoire (MMU, *Memory Management Unit*) (cf. 'Unité de Gestion Mémoire (MMU)' p 21) qui fait correspondre les adresses virtuelles à des adresses physiques, en se basant sur une table de page. (cf. 'Unité de Gestion Mémoire (MMU)' p 21)



Unité de Gestion Mémoire (MMU)

Fondamental : Traduction d'adresse et bit de présence

La figure (cf. 'Correspondance entre espace virtuel et espace réel [crocus75]' p 15) donne un exemple très simple de la manière dont se fait le travail de correspondance dans une MMU à travers une table de pages.

Cependant, cette capacité de mettre en correspondance les 8 pages virtuelles sur les 4 cases physiques en initialisant correctement la MMU ne résout pas le problème posé par un espace d'adressage virtuel plus grand que la mémoire physique. Puisqu'il n'y que 4 cases physiques, seuls les 4 des pages virtuelles de la figure (cf. 'Correspondance entre espace virtuel et espace réel [crocus75]' p 15) sont mises en correspondance avec la mémoire physique. Les autres, illustrées par des croix dans la figure, ne sont pas mises en correspondance. En terme de matériel, un **bit de présence/absence** conserve la trace des pages qui se trouvent physiquement en mémoire.

Fondamental : Défaut de page

Que se passe-t-il si le programme essaye par exemple de faire appel à une page non présente ? La MMU remarque que la page est absente (ce qui est indiqué par le bit de présence) et fait procéder le CPU à un déroutement, c'est-à-dire que le processeur est restitué au système d'exploitation. Ce déroutement, appelé **défaut de page** est réalisé de la manière suivante :

- le système d'exploitation sélectionne une case peu utilisée et écrit son contenu sur le disque;
- il transfère ensuite la page qui vient d'être référencée dans la case libérée,
- modifie la correspondance et
- recommence l'instruction déroutée.

[tanenbaum03]

Méthode : Pagination à a demande

L'algorithme suivant résume la technique de pagination à la demande.

```
Le programme gère une adresse logique paginée (p,d)
SI p est valide
ALORS générer adresse physique
SINON /* Défaut de page */
  Lire p de la mémoire secondaire /* swap in */
  SI il y une case libre en mémoire centrale
  ALORS
    allouer cette case à p
```

```
    mettre à jour la table de page
    générer l'adresse physique
  SINON /* algorithme de remplacement */
    choisir une page présente en mémoire centrale
    l'écrire en mémoire secondaire /* swap out */
    mettre à jour le bit de validité
    affecter la case libérée à la nouvelle page
      et mettre à jour la table de page
    générer l'adresse physique
  FSI
FSI
```

4. Algorithmes de remplacement de page

Remplacement de page (local / global)

Lors d'un défaut de page, s'il n'y a pas de case libre pour charger la page demandée comment choisir la page à remplacer ? Un algorithme de remplacement peut être :

- Local : si sa victime est choisie parmi les pages allouées au programme
- Global : si sa victime est choisie parmi l'ensemble de toutes les pages présentes en mémoire centrale.

Définition : Chaîne de références

On appelle chaîne de références la séquence des numéros de pages référencées par un programme durant une exécution. Un bon algorithme de remplacement doit diminuer le nombre de défauts de pages engendrés par une chaîne de référence définie par l'exécution d'un processus.

Méthode : Algorithme optimal

Pour une chaîne de référence donnée, on peut montrer que l'algorithme suivant minimise le nombre total de défauts de pages : lors d'un défaut de page, choisir comme victime une page qui ne fera l'objet d'aucune référence ultérieure, ou, à défaut, la page qui fera l'objet de la référence la plus tardive. Cet algorithme suppose une connaissance de l'ensemble de la chaîne de référence; il est donc irréalisable en temps réel. Il permet d'évaluer, par comparaison, les autres algorithmes *[krakowiak87]*.

Méthode : Tirage Aléatoire

La victime est choisie au hasard (loi uniforme) parmi l'ensemble des pages présentes en mémoire. Cet algorithme n'a aucune vertu particulière, car il ne tient aucun compte du comportement observé ou prévisible du programme; il sert surtout de point de comparaison *[krakowiak87]*.

Méthode : Ordre Chronologique de Chargement (FIFO)

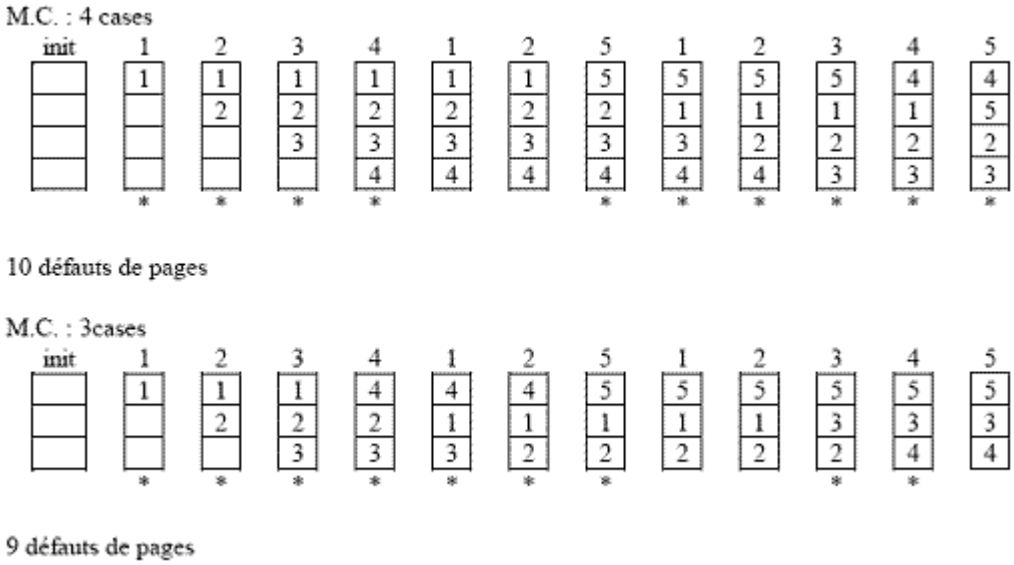
Cet algorithme choisit comme victime la page la plus anciennement chargée. Son principal intérêt est sa simplicité de réalisation : il suffit d'entretenir dans une file FIFO les numéros des cases où sont chargées les pages successives *[krakowiak87]*.

Remarque : Anomalie de Belady

Logiquement s'il on a plus de cases, on aura moins de défauts de pages. Belady a montré que ce n'est pas le cas.

Exemple : Anomalie de Belady

On considère un programme qui utilise 6 pages et la chaîne de référence suivante: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Puis nous déroulons l'algorithme de remplacement de pages FIFO en utilisant une mémoire à 4 cases puis à 3 cases. On remarque qu'avec 3 cases on obtient moins de défauts de pages qu'avec 4 cases. La figure (cf. 'Anomalie de Belady' p 23) illustre le déroulement de l'algorithme dans ces deux situations.



Anomalie de Belady

Attention : FIFO et anomalie de Belady

L'algorithme FIFO souffre de l'anomalie de Belady

Méthode : Ordre Chronologique d'utilisation (LRU: Least Recently Used)

Cet algorithme tente d'approcher l'algorithme optimal, en utilisant la propriété de localité. Son principe s'explique comme suit: puisque les pages récemment utilisées ont une probabilité plus élevée que d'autres d'être utilisées dans un futur proche, une page non utilisée depuis un temps élevé a une probabilité faible d'être utilisée prochainement. L'algorithme choisit donc comme victime la page ayant fait l'objet de la référence la plus ancienne. La réalisation de l'algorithme impose d'ordonnancer les cases selon la date de dernière référence de la page qu'elles contiennent. Pour cela, une information doit être associée à chaque case et mise à jour à chaque référence. Cette référence peut être une date de référence; une solution plus économique consiste à utiliser un compteur incrémenté de 1 à chaque référence; la case dont le compteur a la valeur la plus faible contient la victime. En raison de son coût, cette solution n'a été utilisée que sur des installations expérimentales. Si la taille du compteur est réduite à un bit, on obtient l'algorithme suivant, dont le coût est acceptable *[krakowiak87]*.

Méthode : Algorithme de la seconde chance

Cet algorithme est une approximation très grossière de LRU. A chaque case est associé un bit d'utilisation U, mis à 1 lors d'une référence à la page qu'elle contient. Les cases sont ordonnées dans une file circulaire et un pointeur ptr désigne la dernière case chargée. Le pointeur progresse jusqu'à la première case dont le bit U est à zéro; les cases rencontrées en route, dont le bit U est à 1, reçoivent une seconde chance (de n'être pas prises comme victime) et leur bit U est mis à 0. Cet algorithme est également appelé algorithme de l'horloge "Clock", la progression du

pointeur étant analogue à celle de l'aiguille d'une horloge. La mise à 1 du bit U lors d'une référence peut être réalisée par un mécanisme matériel ou logiciel *[krakowiak87]*.

5. Gestion Mémoire dans le sytème Linux [bouzefrane03]

Pagination hiérarchique

Dans le système Linux, la gestion mémoire est conçue selon un modèle de pagination à trois niveaux reposant sur les tables suivantes :

- la table globale (page global directory) : elle contient les numéros des pages contenant des tables intermédiaires;
- la table intermédiaire (page middle directory) : elle contient les numéros des pages contenant des tables des pages;
- la table des pages (page table) : elle contient les adresses des pages mémoire contenant le code ou les données utilisés par le processus.

Pagination sous Linux avec un Pentium

Ce modèle est réduit à une pagination à deux niveaux sur les machines basées sur la famille de processus Pentium, en faisant disparaître les tables intermédiaires. En effet, ces processeurs, qui possèdent un adressage sur 32 bits, peuvent adresser un espace virtuel de 4 Go. Avec des pages mémoires de 4 Ko, la table des pages, dans une pagination à un niveau, contiendrait 2^{20} entrée. Pour éviter une aussi grande table des pages, la mémoire est gérée avec une pagination à deux niveaux.

Comme on trouve dans une table des pages 1024 entrées de 4 octets, une table de pages tiendra exactement dans une page mémoire et décrit un espace de 1024 pages de 4 Ko, soit 4 Mo.

Ainsi un programme de taille inférieure ou égale à Nx4 Mo nécessitera N tables de pages. Mais un programme chargé en mémoire n'aura pas nécessairement ses N tables en mémoire : en effet, il lui suffit pour s'exécuter d'avoir sa table des hyperpages et la table des pages correspondant aux pages en cours d'utilisation. Le bit de présence P dans l'entrée k de la table des hyperpages indique si la table des pages k est chargée en mémoire ou non.