Module 7: Mémoire virtuelle

1 Introduction

La technique qui permet d'exécuter des programmes dont la taille dépasse la taille de la mémoire principale s'appelle la **mémoire virtuelle**.

La mémoire virtuelle et la mémoire principale sont structurées en unités d'allocations. Ces unités sont appelées des pages dans le cas de la mémoire virtuelle et des cadres dans le cas de la mémoire principale.

Les transferts entre la mémoire virtuelle et la mémoire principale sont gérés par un composant intégré au processeur qui s'appelle MMU (Memory Management Unit), à l'aide de tables de correspondances entre espace virtuel et espace réel. De son côté, la segmentation autorise un découpage logique de l'espace mémoire d'un programme en fonction de son utilisation pour rendre plus efficace son positionnement en mémoire physique [1].

Notez que les adresses virtuelles référencées pat l'instruction en cours d'exécution doivent être traduites en adresse physique. 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.

2 Principe de la mémoire virtuelle

L'espace d'adressage de la mémoire virtuelle est stocké sur disque [2]. Pour pouvoir exécuter un processus, le système d'exploitation charge en mémoire uniquement quelques pages (ou segments) incluant celle qui contient le début du programme à exécuter.

Lorsqu'un processus commence à être exécuté, seule une partie de l'espace d'adressage du processus se trouve en mémoire principale. Si le processus fait référence à une partie qui n'existe pas en mémoire, on suspend son exécution et le processus passe à l'état **bloqué**. Une demande sera donc faite auprès des entrées/sorties par le système d'exploitation pour aller chercher la partie manquante. Une fois la partie est récupérée, le processus passe à l'état **prêt**.

La figure 1 un exemple d'espace des adresses virtuelles et espace physique [3].

1. Texte tiré du livre de l'architecture de l'ordinateur.
2. Disque veut dire disque dur.
3. Figure tirée de `http://www.groupes.polymtl.ca/inf2610/documentation/notes/chap10.pdf`

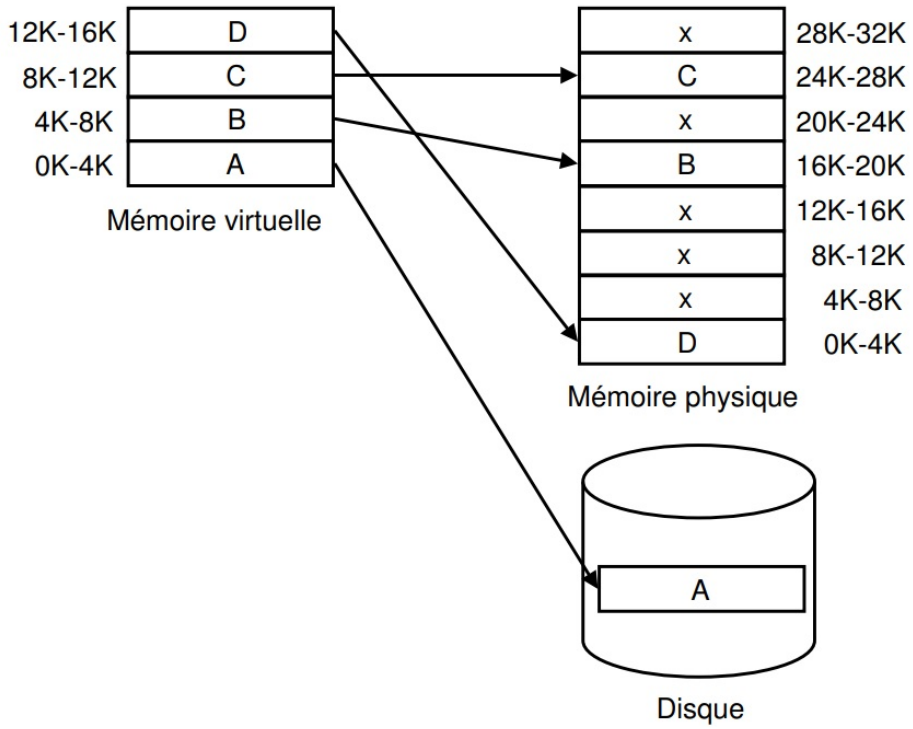

FIGURE 1 – Exemple d'espace des adresses virtuelles et espace physique. Le processus dans son espace virtuel continu comprend quatre pages : A, B, C et D. Trois blocs sont situés dans la mémoire physique et le quatrième bloc est sur le disque dur.

2.1 Pagination

Comme nous l'avons mentionné plus haut, lorsqu'un processus n'est présent que partiellement en mémoire principale, il faut donc prévoir un autre support de mémorisation pour stocker le reste du programme. Ce support est le disque dur.

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. Une page possède une taille fixe qui est égale à celle d'un cadre. Elle varie généralement entre 2 Ko et 16 Ko. La valeur de 4 Ko est une valeur typique assez répandue.

La figure 2 montre un exemple de trois processus qui s'exécutent en même temps [4].

4. Figure tirée du livre de l'architecture de l'ordinateur.

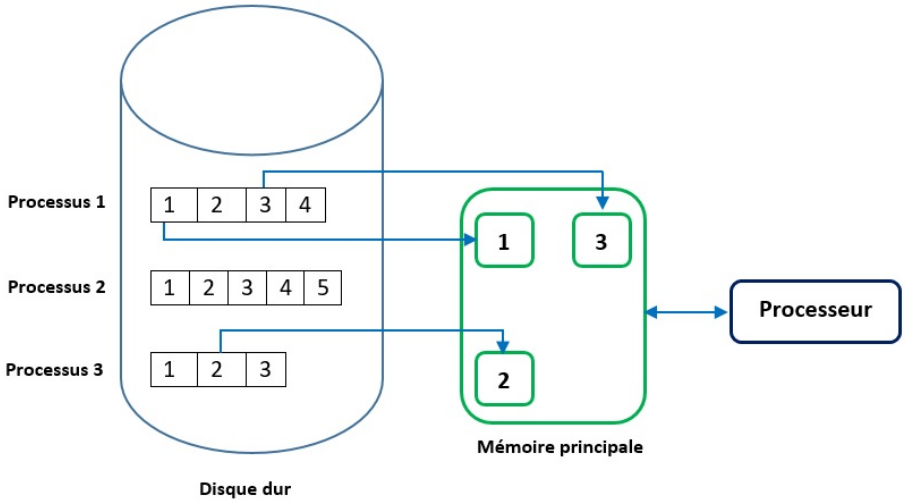

FIGURE 2 – Principe de stockage virtuel et le concept de pages.

Les caractéristiques de ces trois processus sont :
— Processus 1 : il a un espace mémoire composé de 4 pages. Deux pages sont déjà présentes en mémoire principale.
— Processus 2 : il n'a aucune page en mémoire principale. Il ne peut pas être exécuté tant qu'il n'a pas une de ses pages en mémoire principale.
— Processus 3 : il a une seule page en mémoire, qui pourrait probablement être celle en cours d'exécution.

2.1.1 Adressages virtuel et réel

Nous savons maintenant que chaque processus a un espace mémoire. Et que cet espace mémoire et la mémoire principale sont totalement disjoints. Cela veut dire que nous avons de adressages différents :

1. Un adressage logique au niveau de l'espace mémoire de processus.
2. Un adressage physique au niveau de la mémoire principale.

Notez que les pages de chaque processus pourraient se situer n'importe où en mémoire principale. Par conséquent, pour pouvoir récupérer une donnée située dans un boîtier mémoire, il faut, tout d'abord, transformer les adresses virtuelles (pages) en adresses physiques dites réelles.

La transformation des adresses virtuelles en adresses réelles se fait par le composant MMU comme le montre la figure 3 [5].

5. Figure tirée du livre de l'architecture de l'ordinateur.

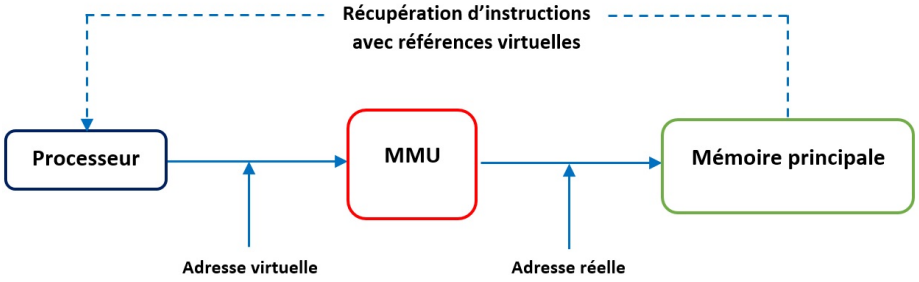

FIGURE 3 – Traduction d'adresse virtuelle en adresse réelle.

2.2 Algorithme d'accès à la mémoire

Nous avons dit auparavant que l'espace virtuel du processus est découpé en pages de quelques kilo-octets (4 Ko en général). Ces pages représentent donc l'unité de transfert entre le disque dur et la mémoire principale.

Pareillement, la mémoire physique est découpée en emplacements qui ont la même taille d'une page. Ces emplacements sont appelés cadres. Par conséquent, chaque cadre pourrait contenir une page virtuelle.

Lors des opérations d'accès à la mémoire, c'est-à-dire, récupération d'une donnée par exemple, ou la lecture de la prochaine instruction, le processeur effectue ces opérations à l'aide d'une adresse virtuelle. Des opérations doivent donc être effectuées par le processeur et qui sont résumées dans les points suivants :

1. L'adresse virtuelle est envoyée à l'unité de gestion de la mémoire (MMU).
2. Si la page virtuelle se trouve déjà en mémoire physique, l'unité de gestion de la mémoire effectue une transformation de l'adresse virtuelle en adresse réelle et produit donc l'accès mémoire.
3. Si la page virtuelle n'existe pas dans la mémoire physique, il y a un défaut de page.
4. Dans le cas d'un défaut de page, l'unité de gestion de la mémoire donne la main au système d'exploitation pour qu'il puisse aller chercher la page sur le disque et la mettre en mémoire physique.
5. Le processeur effectue donc une autre tentative d'accès à la mémoire.

La pagination est un mécanisme efficace qui permet de bien séparer la gestion de l'espace mémoire d'un processus et la gestion de la mémoire réelle. Cependant, ce mécanisme pourrait être lent car le processeur doit effectuer un accès au disque dur (support de stockage secondaire) pour récupérer des pages virtuelle (étape qui peut prendre 10 ms) et l'amener en mémoire physique. Notez que l'accès à la mémoire physique est beaucoup plus rapide (environ 50 ns) que l'accès au disque dur. Cela pourrait donc pénaliser les performances des systèmes.

3 Unité de gestion de la mémoire

L'unité de gestion de la mémoire utilise une table pour mémoriser les correspondances entre les adresses virtuelles et les adresses physiques. Cette table est appelée **table de pages**.

Le nombre d'entrée dans la table de pages égale au nombre de pages virtuelles. Comme nous l'avons vu dans la figure 1, la table de pages d'un processus doit être en totalité ou en partie en mémoire principale lors de l'exécution du processus. Cette table est nécessaire à l'exécution des processus car elle assure la conversion des adresses virtuelles en adresses réelles.

3.1 Structure de la table de pages

La table de pages est un grand tableaux indexé par le numéro de page virtuelle, et qui donne le numéro de page physique où elle est située comme illustré dans la figure 4 [6]. Il existe un bit de validité (V) qui indique si l'information qui se trouve dans la ligne correspondante est correcte ou non.

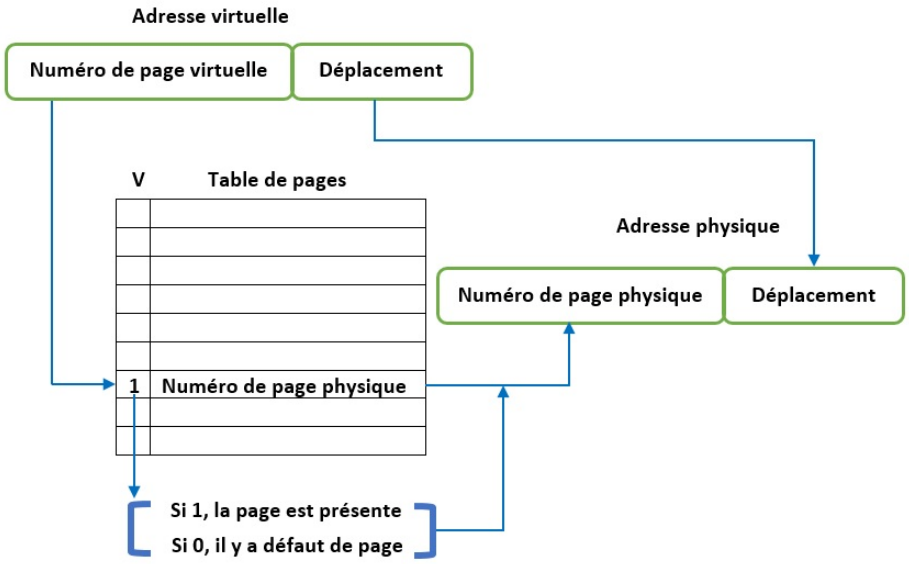

FIGURE 4 – Structure de la table de pages.

Notez que cette table de pages est stockée en mémoire principale et son adresse physique est mémorisée dans un registre spécial de l'unité de gestion de la mémoire. Cela veut dire, que lors d'une transformation d'adresse virtuelle, l'unité de gestion de la mémoire effectue donc un premier accès mémoire pour

6. Figure tirée du livre de l'architecture de l'ordinateur.

récupérer d'abord l'entrée dans la table de pages avant de faire l'accès mémoire proprement dit.

Chaque entrée dans la table de pages est composée de plusieurs champs détaillés comme suit :

— Le bit de présence.
— Le bit de référence (R).
— Les bits de protection.
— Le bit de modification (M).
— Le numéro de case correspondant à la page.

3.2 Fonctionnement d'un MMU

Le principe de base d'un MMU est qu'il reçoit en entrée une adresse virtuelle, et envoie en sortie une adresse physique. Pour comprendre ce principe de fonctionnement, prenons l'exemple suivant [7] :

Dans cet exemple (voir figure 5 [8]), nous supposons que l'adresse virtuelle est codée sur 16 bits, et que le nombre de pages dans cet exemple est 16 numérotée entre 0 et 15. Donc :

— Les 4 bits de poids fort indiquent le numéro de page, comprise entre 0 (0000_2) et 15 (1111_2).
— Les autres bits donnent le déplacement dans la page, entre 0 (000000000000_2) et 4095 (111111111111_2).

7. Exemple tiré du cours `http://www.groupes.polymtl.ca/inf2610/documentation/notes/chap10.pdf`
8. Figure tirée du cours `http://www.groupes.polymtl.ca/inf2610/documentation/notes/chap10.pdf`

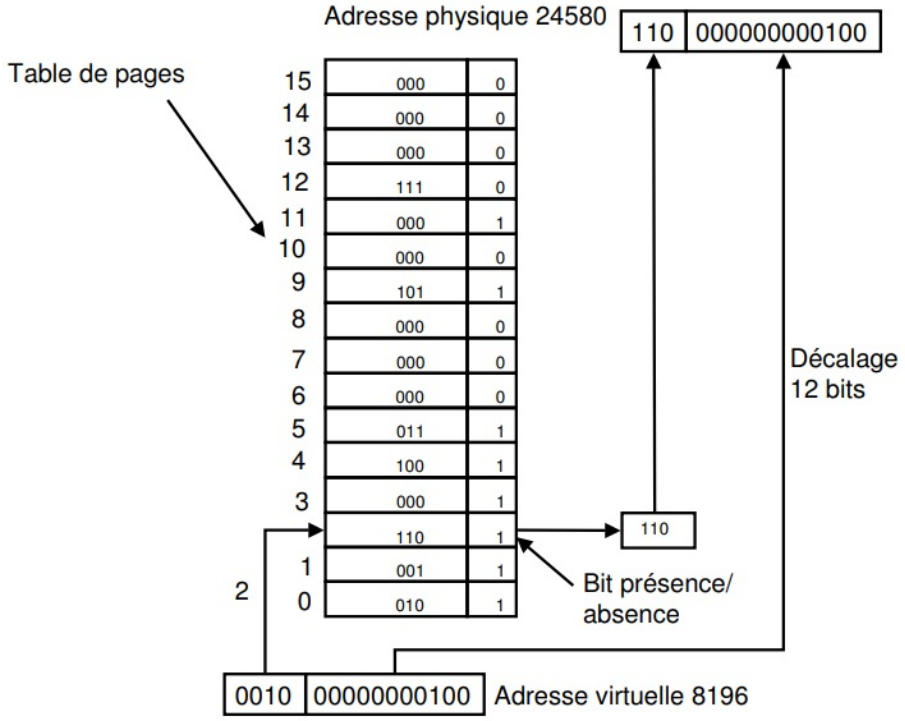

FIGURE 5 – Exemple de fonctionnement d'un MMU.

Le MMU examine l'entrée de la table de pages comme montrée dans la figure 5, dans ce cas l'entrée numéro 2. Si le bit de validation (appelé aussi bit de présence/absence) est à 0, la page n'est pas en mémoire, donc il y a un défaut de page. Si le bit de validation est à 1, alors le MMU 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 5.

3.3 Table de pages à plusieurs niveaux

Une table de pages pourrait avoir une taille très grande. À titre d'exemple, si nous utilisons un adressage virtuel sur 32 bits, nous aurons plus de 2^{20} (20 = 32 - 12 (déplacement)) entrées dans cette table. Et si chaque entrée est formée de 3 octets, nous aurons donc besoin de 3 Mo pour stocker cette table, ce qui est déjà considérable. Donc, pour éviter d'avoir des tables de cette taille dans la mémoire, d'autres méthodes ont été proposées et qui permettent d'utiliser des

tables de pages à plusieurs niveaux. La figure 6 [9] montre un exemple de table de pages à plusieurs niveaux.

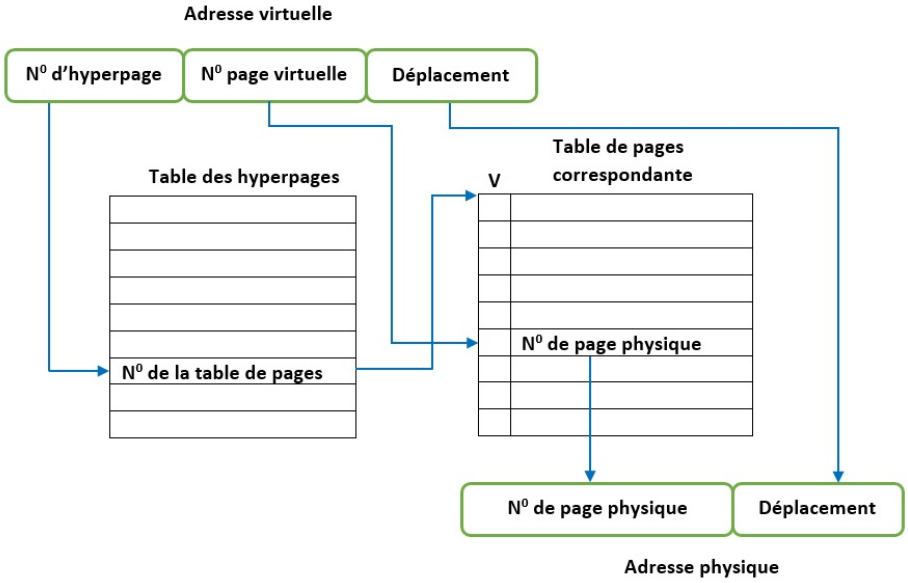

FIGURE 6 – Exemple de table de pages à plusieurs niveaux.

Si nous reprenons l'exemple précédent, pour un adressage virtuel sur 32 bits, et des pages de 4 Ko, nous pourrons avoir 1025 tables de 1024 entrées. De cette façon, nous avons la possibilité de charger uniquement les tables qui nous intéressent, les autres tables n'auront pas besoin d'être chargées en mémoire.

L'idée des tables de pages à plusieurs niveaux est de découper les numéros de pages virtuelles en deux ou plus. Dans ce cas, les bits de poids fort indiquant un numéro d'hyperpage, et ceux de poids faible indiquant la page virtuelle dans cette hyperpage. Par conséquent, ce découpage permet de charger uniquement la table des hyperpages de taille moins importante. La table de pages correspondante sera chargée en mémoire à la demande.

Bien qu'avec les tables de pages à plusieurs niveaux, nous gagnons d'espaces, l'inconvénient de cette méthode est qu'elle ralentit l'accès à la mémoire car le MMU doit consulter deux tables au lieu d'une avant de pouvoir effectuer la conversion de l'adresse virtuelle en adresse physique.

3.4 Algorithme de défaut de page

Nous avons vu précédemment qu'un défaut de page se produit lorsque le processus fait référence à une page virtuelle qui n'est pas encore disponible en

9. Figure tirée du livre de l'architecture de l'ordinateur.

mémoire physique. L'absence de page est toujours signalée à l'aide du bit de validité. Dans ce cas, le MMU demande au système d'exploitation de récupérer sur le disque la page virtuelle et de l'amener en mémoire physique et relancer l'instruction ayant provoqué ce défaut de page. Les étapes de traitement d'un défaut de page sont résumées dans les points suivants :

1. Arrêt de l'instruction en cours d'exécution.
2. Une interruption de défaut de page est levée.
3. Le système d'exploitation prend la main.
4. Le système d'exploitation suspend donc l'exécution du processus.
5. Le système d'exploitation vérifié que la référence mémoire faite par le processus est valide.
6. Avant de récupérer la page virtuelle, le système d'exploitation choisit un cadre de page libre dans la mémoire pour recevoir la page virtuelle.
7. Le système d'exploitation effectue une opération de lecture sur le disque afin de récupérer la page virtuelle en question et la mettre en mémoire. Pendant cette opération de lecture, le système peut donner la main à un autre processus pour poursuivre sont exécution.
8. Une nouvelle interruption prévient le système d'exploitation de la fin de transfert de la page virtuelle en mémoire physique. Le système d'exploitation peut maintenant mettre à jour la table de pages et débloquer le processus interrompu.

3.5 Remplacement de page

Comme nous l'avons vu dans le cadre de la mémoire cache, il existe également des algorithmes de remplacement de page dans le cadre de la mémoire virtuelle. La question qui se pose, comment choisir la page à remplacer ?

Les algorithmes de remplacement vus dans le cadre de la mémoire cache sont également utilisés pour le remplacement de pages. Ces algorithmes sont résumés dans les points suivants :

— **Remplacement FIFO** : la page la plus ancienne sera remplacée.
— **Remplacement LRU** : la page dont la dernière utilisation est la plus ancienne sera remplacée.
— **Remplacement LFU** : la page la moins utilisée sera remplacée.
— **Remplacement MFU** : la page qui a été la plus utilisée sera remplacée.

3.6 Accélération de la traduction

L'unité de gestion de la mémoire doit traduire une adresse virtuelle en adresse physique à chaque référence mémoire. Cela est très coûteux particulièrement si la table de pages est de grande taille.

Dans cette section, nous allons voir comment nous pouvons améliorer les performances du processus de traduction des adresses virtuelles.

3.6.1 Table inverse

Rappelons que la taille de la table de pages est directement proportionnelle au nombre de pages virtuelles. Cela veut dire que lorsque les adresses sont codées sur 32 bits, la table pourrait se placer entièrement en mémoire. Mais, si le codage est sur 64 bits et plus, le stockage de la table en mémoire devient impossible.

Une solution simple pour ce problème est de prévoir une table dite **inverse** qui donne, pour chaque cadre de page physique, la page virtuelle qui y est stockée. L'idée de cette solution est que la recherche à partir des pages virtuelles est plus compliquée car il faut parcourir toutes les pages de la table pour vérifier si cette page se trouve dans un cadre physique.

L'autre avantage de la table inverse est que sa taille est fonction du nombre de cadres physiques et non pas du nombre de pages virtuelles. La figure 7 [10] montre un exemple de table inverse.

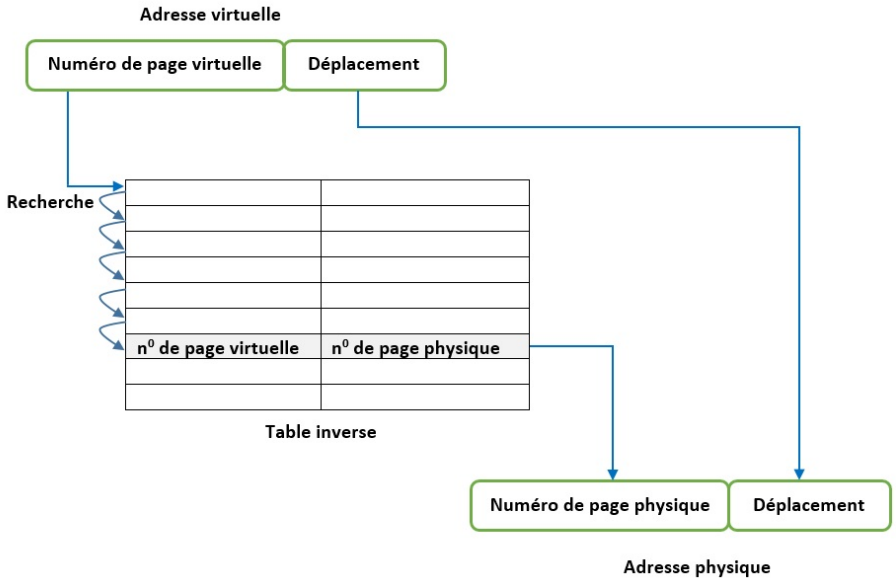

FIGURE 7 – Exemple de fonctionnement de la table inverse.

Notez que le MMU utilise une fonction de hachage à partir de l'adresse virtuelle pour sélectionner une entrée où peut se retrouver la correspondance.

3.6.2 Tampon de traduction anticipé

Nous avons vu que pour trouver l'adresse physique, il faut toujours effectuer un accès mémoire peu importe le type de la table de pages utilisé (plusieurs ni-

10. Figure tirée du livre de l'architecture de l'ordinateur.

veaux, inverse, avec hachage). Cette table est stockée dans la mémoire physique à une adresse connue pour le MMU. Cela veut dire que nous avons besoins de deux accès : un premier accès mémoire pour la table, et un deuxième accès pour chercher l'information, ce qui double encore le temps.

Pour éviter ce ralentissement, le MMU inclut un mécanisme pour accélérer le processus de traduction des adresses virtuelles. Le rôle de ce mécanisme est d'éviter le double accès à la mémoire pour consulter la table de pages. L'idée de ce mécanisme est de garder les derniers couples utilisés dans un tampon de traduction anticipé (TLB : Translation Lookaside Buffer). Ce tampon est doté d'une mémoire associative rapide et de petite taille. La taille de cette mémoire varie entre 256 Ko à quelques Kilo-octets [11].

De cette façon, la traduction d'adresse virtuelle peut donc se conclure de trois façons [12] :

— La recherche dans le TLB est concluante (TLB hit) : la MMU génère rapidement l'adresse physique équivalente et la pénalité d'accès est minime.

— Le TLB ne contient pas la correspondance (TLB miss, échec TLB), mais la page virtuelle est en mémoire physique : un accès à la table de pages permet d'avoir son adresse réelle et le temps d'accès est doublé.

— La page virtuelle n'est pas encore en mémoire physique (et il y a forcément un échec TLB) : le MMU soulève une interruption de défaut de page, le système d'exploitation prend la main pour effectuer un accès disque afin de ramener la page virtuelle en mémoire, au prix d'une pénalité d'accès importante.

4 Segmentation

La segmentation est une technique qui peut être combinée avec la pagination. Chaque segment est composé d'un ensemble de pages. Les adresses générées prennent la forme des triplets comme suit :

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

La segmentation consiste donc à découper l'espace mémoire d'un processus en segments, répertoriés par numéro et stockés n'importe où en mémoire principale comme le montre la figure 8 [13]. Ces segments peuvent avoir des tailles différentes.

11. Tiré du livre de l'architecture de l'ordinateur.
12. Tiré du livre de l'architecture de l'ordinateur.
13. Figure tirée du livre de l'architecture de l'ordinateur.

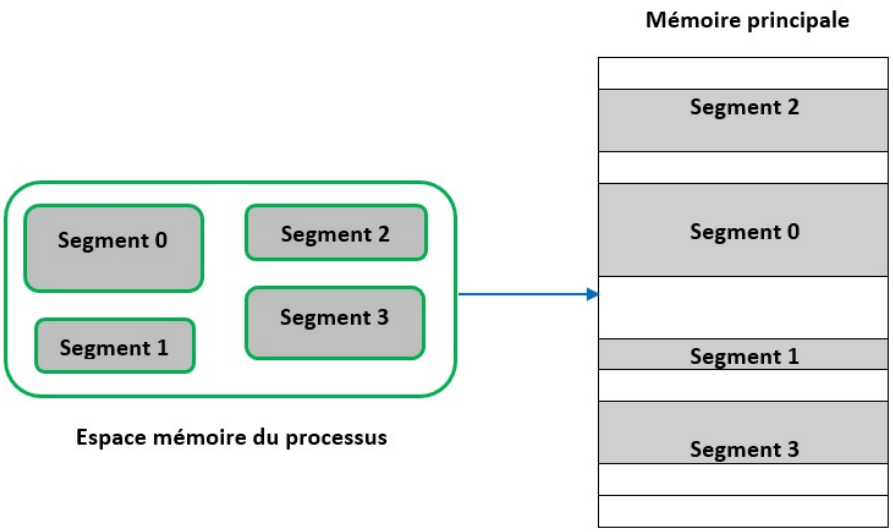

FIGURE 8 – Segmentation de l'espace mémoire.