Chapitre 04:

Gestion de la mémoire

(Memory management)

1. Définitions :

- **Mémoire vive (centrale ou interne, RAM) :** un type de mémoire qui permet de stocker des informations provisoires. Son avantage majeur est sa capacité de lecture très rapide par rapport au disque dur et qui permet une utilisation fluide de l'ordinateur (son but est d'y accéder rapidement et provisoirement). Quand le système s'éteint, la RAM se vide.
- **Mémoire de masse (physique ou interne):** est une mémoire de grande capacité qui peut être lue et écrite par un système (Bande magnétique, disque dur, disque optique (CD, DVD, Blu-ray), disque magnéto-optique et mémoire flash).
- **Mémoire d'échange (SWAP) :** est une partie de la mémoire de masse d'un ordinateur utilisée par le système d'exploitation pour stocker des données qui, du point de vue des applications, se trouvent en mémoire vive. Son intérêt est de simuler sans coût une mémoire vive plus grande, avec une perte de vitesse limitée. C'est utile lorsque la mémoire vive est pleine.
- **Mémoire virtuelle :** Espace du disque dur interne d'un ordinateur qui vient seconder la mémoire vive, Elle se concrétise par un fichier d'échanges (fichier swap), lequel contient les données non sollicités constamment. La mémoire virtuelle, comme son nom l'indique, sert à augmenter artificiellement la mémoire vive.
- **Mémoire morte (ROM) :** Type de mémoire dont le contenu est accessible en lecture et non en écriture. Elle conserve les données en l'absence de courant électrique. Elle stocke les informations nécessaires au démarrage d'un système (BIOS, des équipements embarqués, des tables de constantes,...).
- **Mémoire flash :** est une mémoire de masse à semi-conducteurs réinscriptible, c'est-à-dire une mémoire possédant les caractéristiques d'une mémoire vive mais dont les données ne disparaissent pas lors d'une mise hors tension.
- **Mémoire cache :** est une mémoire plus rapide et plus proche du matériel informatique (processeur, disque dur). Son rôle est de stocker les informations les plus fréquemment utilisées par les logiciels et applications lorsqu'ils sont actifs. C'est

cet accès direct qui détermine les performances d'un programme car il économise des échanges incessants entre le processeur et la mémoire vive.

2. Rôle d'un gestionnaire de mémoire :

- création d'un processus.
- activation/désactivation d'un processus.
- suppression d'un processus.
- partage la mémoire disponible entre les processus (protection).
- cartographie la mémoire.
- alloue/dés-alloue de la mémoire dynamiquement pour les besoin d'un processus.
- assure la cohérence de la mémoire.
- optimise l'utilisation de la mémoire.

3. Organisation de la mémoire :

Une application ne peut s'exécuter que si ses instructions et ses données sont en mémoire physique (vive). Donc, si on désire exécuter plusieurs programmes simultanément dans un système, il faudra que tous ces programmes soient chargés dans la mémoire. L'OS devra donc allouer à chaque programme une zone de la mémoire où celui-ci sera chargé.

Le système prend à sa charge la gestion de la mémoire principale:

- il assure sa protection,
- il fournit l'information qui permet à chaque programme de s'exécuter (à quel endroit dans la mémoire ? → l'adresse et la capacité).

Le but d'une bonne gestion de la mémoire est d'augmenter le rendement global du système.

3.1. Monoprogrammation :

Le rôle le plus simple d'un gestionnaire de mémoire est d'exécuter un seul programme à la fois, en partageant la mémoire entre le programme et le système d'exploitation.

Figure 4.1. Architecture d'une mémoire physique avec monoprogrammation.

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.

3.2. Multiprogrammation :

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.

3.2.1. 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 fixes ou variables.

A. Partitions contiguës fixes: la mémoire est divisée en *n* partitions de tailles fixes (si possible inégales). Ce partitionnement se fait au démarrage du système. Quand un processus arrive, il peut être placé dans la file d'attente.

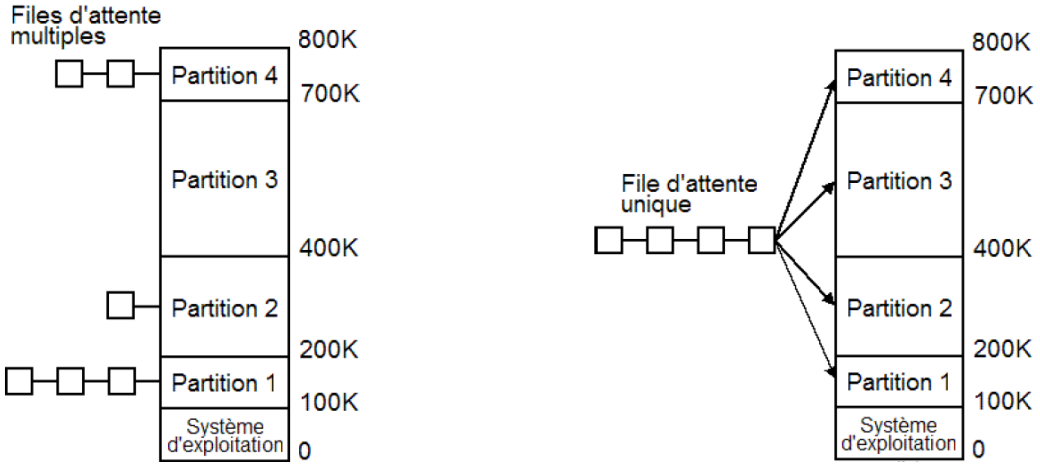
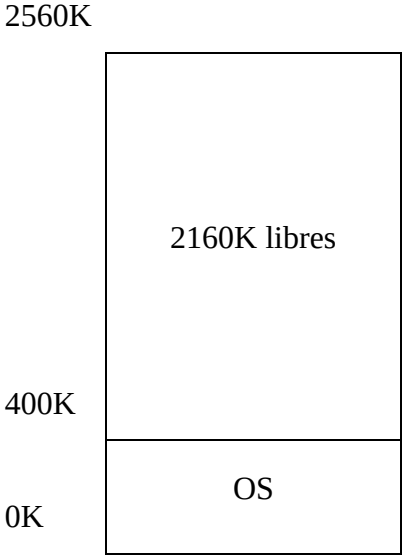


Figure 4.2. Files d'attente séparées et communes.

L'inconvénient des files d'attente séparées se présente lorsque la grande partition est vide tandis que celle d'une petite partition est pleine.

La solution → *file d'attente commune* : Dès qu'une partition devient libre (**trou**), toute tâche placée en tête de file d'attente et dont la taille convient peut être chargée dans cette partition vide et exécutée.

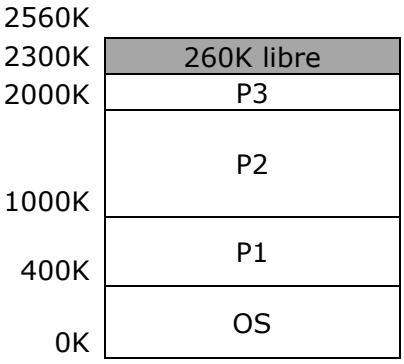
Exemple : L'état de la mémoire d'un système est décrit par la figure suivante. Le système a une file de processus décrit par le tableau suivant (État de la mémoire):



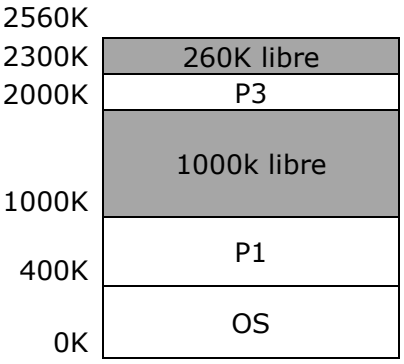
Processus	Mémoire	Temps
P1	600 K	10
P2	1000 K	5
P3	300 K	20
P4	700 K	8
P5	500 K	15

Les figures suivantes montre les différents états successifs de la mémoire après les entrées et les sorties de processus.

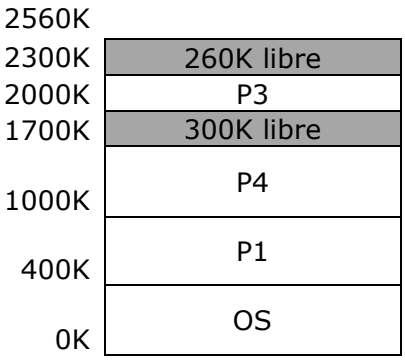
P1, P2 et P3 entrent:



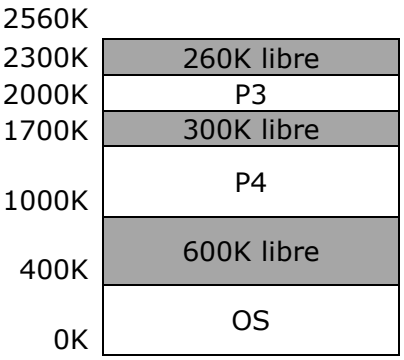
P2 termine:



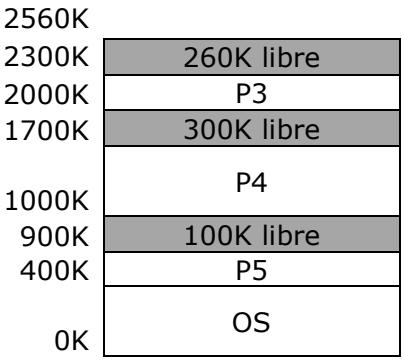
P4 entre:



P1 termine:



P5 entre:



B. Partitions contiguës dynamiques

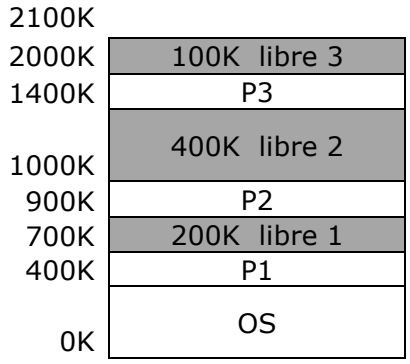
Dans cette stratégie la mémoire est partitionnée dynamiquement selon la demande. Lorsqu'un processus se termine sa partition est récupérée pour être réutilisée (complètement ou partiellement) par d'autres processus. Le gestionnaire de la mémoire doit garder trace des partitions occupées et/ou des partitions libres. On distingue les stratégies de placement suivantes:

- **Stratégie du premier qui convient (FF : First Fit):** On alloue au processus la première partition suffisamment grande.
- **Stratégie du meilleur qui convient (BF : Best Fit):** On alloue la plus petite partition dont la taille est au moins égale à celle du processus en attente.
- **Stratégie du pire qui convient (WF : Worst Fit):** On alloue au processus la partition de plus grande taille.

Remarque: les algorithmes BF et WF nécessitent le tri des partitions libres par ordre des adresses croissantes.

Exemple:

L'état de la mémoire d'un système est décrit par la figure suivante. On suppose qu'un processus P4 demande un espace mémoire de 80K.



En fonction de l'algorithme choisi, P4 occupe :

- Algorithme FF : Partition 1.
- Algorithme BF : Partition 3.
- Algorithme WF : Partition 2.

Exercice:

Dans un système de gestion mémoire à partitions variables, on constate que la liste des partitions libres "trous" est la suivante (dans l'ordre des adresses mémoire croissantes) :
10K 4K 20K 18K 7K 9K 12K 15K

On veut placer successivement des processus de volumes respectifs P1 (12K) P2 (10K) P3 (9K) dans la mémoire.

Indiquer, dans l'ordre des adresses croissantes, la nouvelle liste des trous en utilisant chacune des stratégies de placement : FF, BF, WF ?

Solution:

1) First Fit : utilisation de la première zone libre :

État initial	10K	4K	20K	18K	7K	9K	12K	15K
Placement de P1(12K)	10K	4K	P1+8K	18K	7K	9K	12K	15K
Placement de P2(10K)	P2	4K	P1+8K	18K	7K	9K	12K	15K
Placement de P3(9K)	P2	4K	P1+8K	P3+9K	7K	9K	12K	15K

2) Best Fit : meilleur ajustement :

État initial	10K	4K	20K	18K	7K	9K	12K	15K
Placement de P1(12K)	10K	4K	20K	18K	7K	9K	P1	15K
Placement de P2(10K)	P2	4K	20K	18K	7K	9K	P1	15K
Placement de P3(9K)	P2	4K	20K	18K	7K	P3	P1	15K

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

État initial	10K	4K	20K	18K	7K	9K	12K	15K
Placement de P1(12K)	10K	4K	P1+8K	18K	7K	9K	12K	15K
Placement de P2(10K)	10K	4K	P1+8K	P2+8K	7K	9K	12K	15K
Placement de P3(9K)	10K	4K	P1+8K	P2+8K	7K	9K	12K	P3+6K

C. Va et vient (Swap)

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.

La stratégie *swapping* consiste à considérer chaque processus dans son intégralité. Le programme en cours doit être sauvegardé sur disque avant le chargement en mémoire principale de son successeur pour exécution.

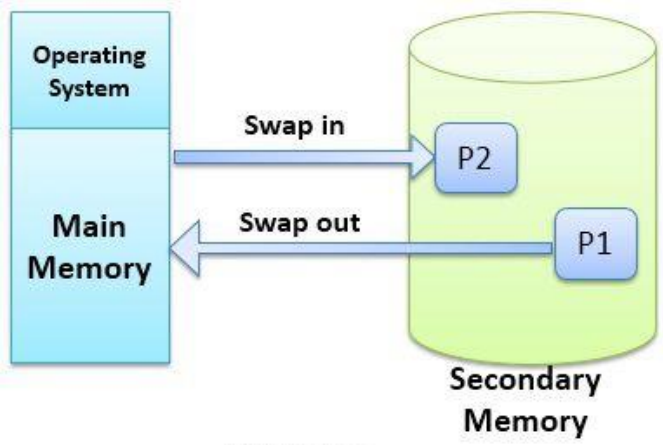
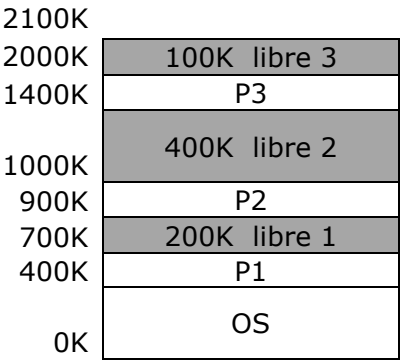


Figure 4.3. Principe de Swapping.

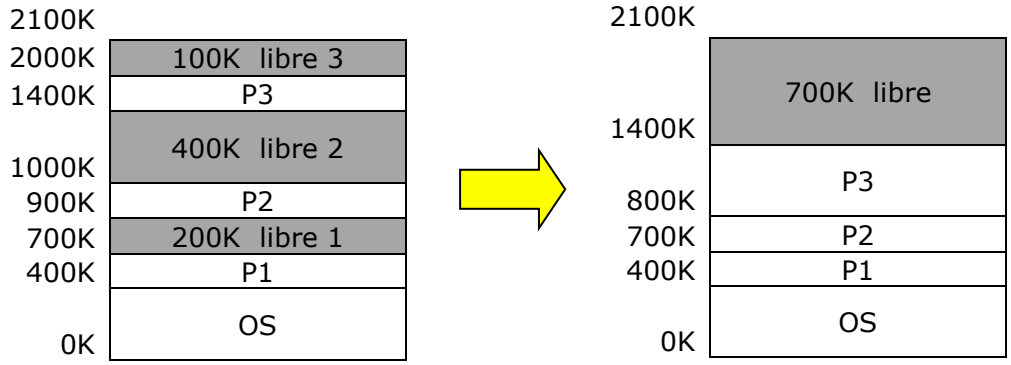
Problème de Fragmentation :

Les algorithmes d'allocation de la mémoire contiguë précédents, produisent la fragmentation de la mémoire. En effet, suite aux différentes entrées et sorties de processus, des fragments (fractions) séparées se forment dans la mémoire.

Si le processus P4 désire entrer dans le système en occupant 500K, il ne pourrait pas bien que l'espace total disponible est de 700 K.



Solution→ Défragmentation par compactage: le principe est de rassembler tous les trous en un seul bloc.



Inconvénient : L'opération de compactage est une opération très coûteuse pour l'OS.

3.2.2. Multiprogrammation et partitions multiples non contigües :

Nous avons vu que les techniques d'allocation associées conduisent à la fragmentation de la mémoire (de nombreuses petites zones libres inexploitables). Pour l'éviter, il faut pouvoir implanter un programme dans plusieurs zones non contiguës. Plusieurs techniques proposent d'allouer des espaces mémoire non-contiguës pour les processus : Pagination et Segmentation.

A. Pagination :

La pagination consiste à découper l'espace adressable, ou espace **virtuel**, en zones de taille fixe appelée **pages**. La mémoire réelle (RAM) 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.

- 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.
- Si la taille de l'espace d'adressage logique est 2^m et la taille d'une page est 2^n :
  - Les *m-n* bits de poids fort d'une adresse paginée désignent le numéro de page,
  - Les *n* bits de poids faible désignent le déplacement dans le page.

Adresse paginée

P	d
m-n bits	n bits

Table des pages : La traduction des adresses utilise une table des pages qui est située en mémoire centrale ou dans des registres. L'adresse réelle (*f*,*d*) d'un mot d'adresse virtuelle/logique (*p*,*d*) est obtenue en remplaçant le numéro de page *p* par le numéro de case *f* trouvé dans la $p^{ième}$ entrée.

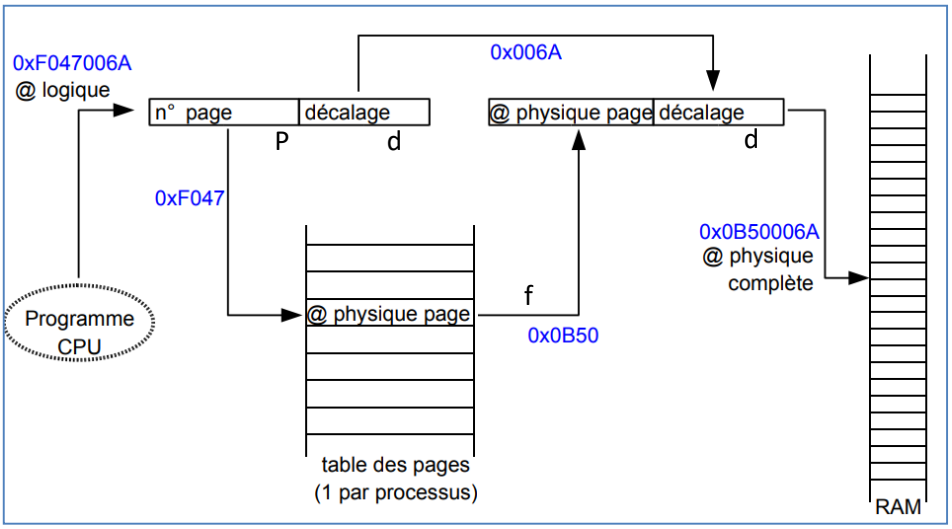


Figure 4.4. Traduction de l'adresse logique (virtuelle).

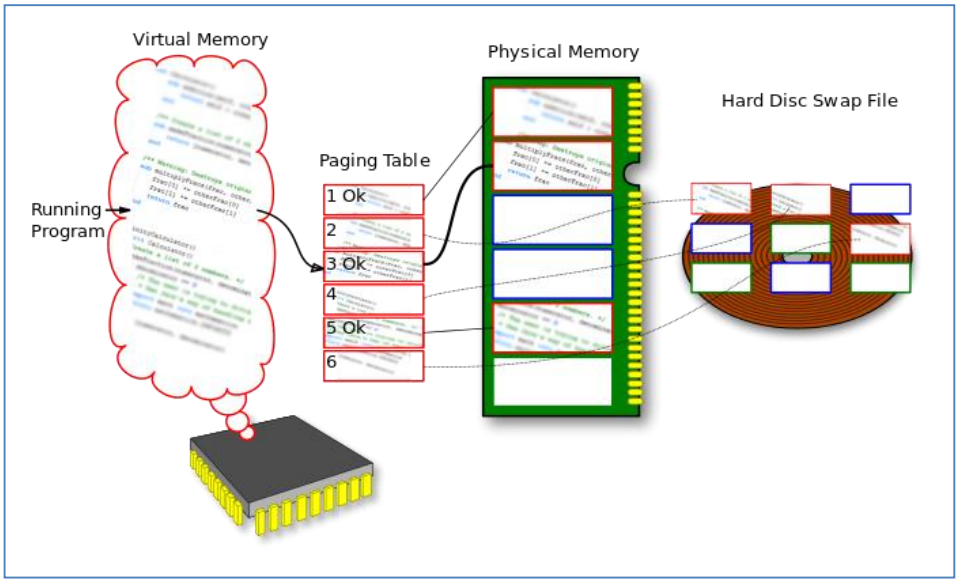


Figure 4.5. Principe de la pagination.

B. Segmentation :

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.

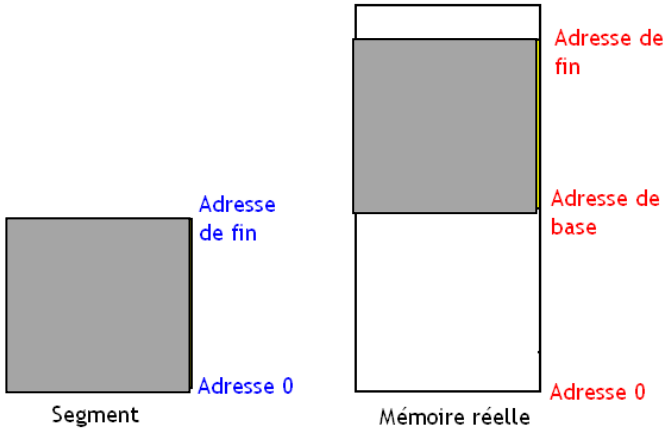


Figure 4.6. Principe de la segmentation.

Table de segments : la table de segment spécifie pour chaque segment deux valeurs :

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

Une adresse logique (virtuelle) 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 (famous Segmentation fault).

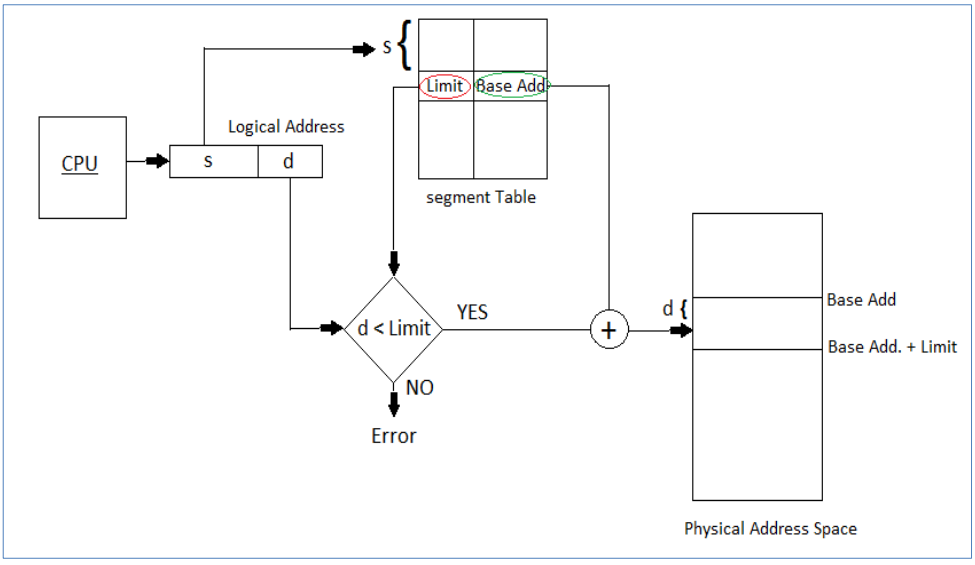


Figure 4.7. Traduction d'adresse d'une mémoire segmentée en utilisant la table des segments.

4. Gestion de la mémoire virtuelle:

La mémoire virtuelle est une technique autorisant l'exécution de processus pouvant ne pas être complètement en mémoire. La mémoire logique est plus grande que la mémoire physique.

→ Possibilité d'exécuter des programmes dont la taille est bien supérieure à celle de la mémoire physique.

→ Sur les ordinateurs actuels, l'espace mémoire virtuel projeté pour un processus peut atteindre plusieurs Go.

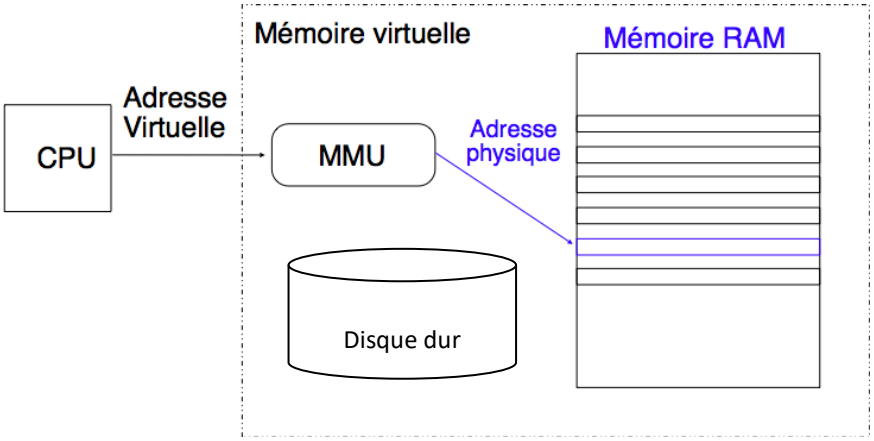


Figure 4.8. Principe de fonctionnement de la mémoire virtuelle.

L'allocation de mémoire consiste à concrétiser cette mémoire virtuelle par des supports physiques d'information tels que la mémoire principale (RAM), les 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.

Principe: 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.

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**) qui fait correspondre les adresses virtuelles à des adresses physiques, en se basant sur une table de page.