La gestion de la mémoire

1. Rôle

Le gestionnaire de mémoire est un sous-ensemble du système d'exploitation. Son rôle est de partager la mémoire entre l'O.S. et les diverses applications. Le terme « mémoire » fait surtout référence à la mémoire principale, c'est à dire à la RAM, mais la gestion de celle-ci demande la contribution de la mémoire auxiliaire (mémoire de masse, spacieuse mais lente) et à la mémoire cache (rapide mais de taille restreinte).

Voici les fonctions qu'on attend du gestionnaire de mémoire :

- **L'allocation de la mémoire aux processus**

  - Répertorier les emplacements libres de la mémoire
  - Allouer la mémoire nécessaire aux nouveaux processus
  - Récupérer la mémoire des processus qui s'achèvent

  Cette récupération peut nécessiter une réallocation des processus en cours pour optimiser l'emploi de la mémoire. La zone mémoire attribuée à un processus peut donc changer au cours de son exécution.

- **La protection**

  Il faut s'assurer que les adresses générées par chaque processus ne concerne que la zone mémoire qui lui est impartie, sans quoi, l'intégrité du système d'exploitation et des autres processus n'est pas garantie.
  Certaines zones mémoire doivent pourtant servir simultanément à plusieurs processus : le code de fonctions servant à plusieurs applications qui tournent en parallèle ou les données utilisées simultanément par divers processus

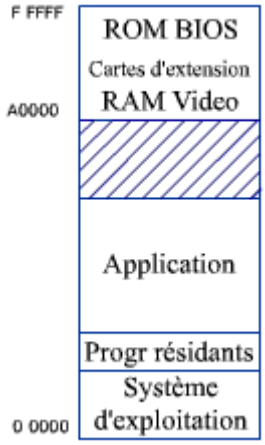
- **La segmentation de l'espace d'adressage**

  Les programmes sont subdivisés en segments : le code, les données modifiables, celles qui ne le sont pas, la pile. On attend donc du gestionnaire de mémoire qu'il permette la segmentation de l'espace d'adressage des programmes pour les raisons suivantes :
  - pouvoir coder les segments séparément et les paramétrer en fonction de l'application
  - permettre des degrés de protection différents selon les segments(lecture seule, exécution…)
  - accepter le partage de certains segments.

- **La mémoire virtuelle**

  Elle offre aux applications une mémoire de taille supérieure à celle de la mémoire principale.
  L'espace d'adressage que représente la mémoire centrale est parfois insuffisant. Les disques suppléent à cette insuffisance en fournissant une mémoire auxiliaire plus vaste mais plus lente et qui n'est pas directement accessible au processeur.

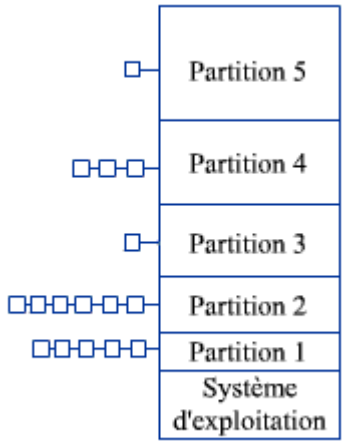
2. Gestion de la mémoire pour systèmes monotâches



Dans le cas des systèmes monotâches, la gestion de la mémoire est assez simple. Il suffit de réserver une partie de la mémoire au système d'exploitation. L'application est ensuite casée dans l'espace restant qui est libéré sitôt que l'application sera terminée.
Cela se complique un peu si l'application nécessite plus d'espace que ce que peut fournir la mémoire vive. On segmente alors l'application en *segments de recouvrements* ou *«Overlays»*. Cette technique n'a plus cours maintenant. Elle était utilisée à l'époque du DOS pour des applications volumineuses. Le programmeur devait prévoir le découpage de son application en imaginant comment ces overlays serait chargés en mémoire les uns à la suite des autres pour qu'au cours de son exécution l'application puisse atteindre toutes les fonctions nécessaires.
C'était en quelque sorte comme cela qu'on concevait la mémoire virtuelle à l'époque des systèmes d'exploitation monotâches.
Notez que la zone mémoire occupée par le système d'exploitation n'était pas protégée par ce type de gestion de mémoire.

3. Gestion de la mémoire pour systèmes multitâches

Plusieurs processus doivent se partager la mémoire sans empiéter sur l'espace réservé au système d'exploitation ni aux autres processus. Quand un processus se termine, le S.E. doit libérer l'espace mémoire qui lui était alloué pour pouvoir y placer de nouveaux processus.



3.1. Partition de la mémoire

Partitions fixes

Le plus simple est de diviser la mémoire en partitions fixes dès le démarrage du système. Les partitions sont de différentes tailles pour éviter que de grandes partitions ne soient occupées que par de petits processus. Le gestionnaire de mémoire, en fonction de la taille des processus, décide quelle partition lui allouer pour ne pas gaspiller trop de mémoire.

Une file d'attente est associée à chaque partition. Quand vient une nouvelle tâche, le gestionnaire détermine quelle est la plus petite partition qui peut la contenir puis place cette tâche dans la file correspondante.

Le fait d'éviter d'allouer une partition trop grande à un petit processus conduit parfois à des aberrations. Il arrive que des partitions plus grandes restent inutilisées alors que se forment ailleurs des files interminables de petits processus. La mémoire est donc mal utilisée.

Une autre solution est de créer une file unique. Lorsqu'une partition se libère, on consulte la file pour trouver la tâche qui l'occuperait de manière optimale.

Le risque est que les petites tâches soient pénalisées. Une parade est de conserver une petite partition au moins qui ne sera accessible qu'aux petites tâches. Une autre solution, serait de dire qu'un processus ne peut être ignorer qu'au maximum un certain nombre de fois. Après n refus, il prendra place dans une partition même si la partition est bien plus grande qu'il ne faut.

Partitions variables

Une autre manière d'éviter les emplacements mémoires inoccupés en fin de partitions est d'allouer aux processus des espaces qui correspondent exactement à l'espace qui leur est utile. Au fur et à mesure que les processus se créent et se terminent, des partitions s'allouent et se libèrent laissant des zones mémoires morcelées et inutilisables.
La mémoire se fragmente et est de plus en plus mal employée. Il faudrait la compacter en déplaçant régulièrement les processus mais cette tâche supplémentaire ralentit le système.

Conclusion

La partition de la mémoire que ce soit avec des partitions de tailles fixes ou de tailles variables, ne permet pas d'utiliser la mémoire au mieux.

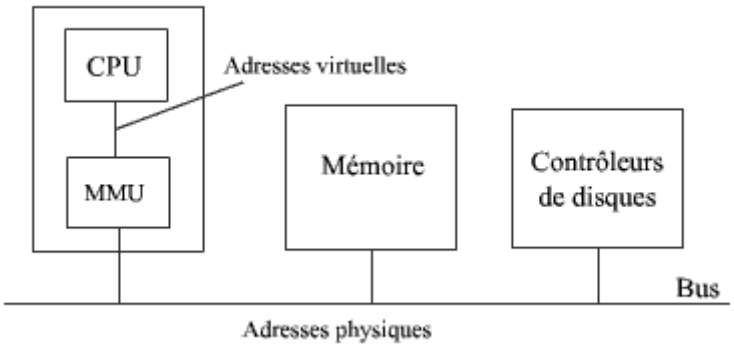
3.2. La pagination

Les processus requièrent des espaces d'adresses continus. On a vu que cela est difficilement réalisable en découpant la mémoire en partions dont les tailles correspondent à celles des processus. La pagination est une technique d'allocation de la mémoire bien plus efficace. Elle fournit aux processus des ***espaces d'adresses*** séquentiels à partir d'***espaces mémoire*** discontinus.

La pagination consiste à diviser la mémoire et les processus en blocs de mêmes tailles appelés pages. Les ***pages mémoire*** sont souvent appelées *"page frames"* ou *"cadres de pages"* tandis que les ***pages de processus*** sont simplement appelées *"pages"*.

Les pages (de processus) ne sont pas toutes simultanément actives ; elles ne sont donc pas nécessairement toutes présentes simultanément dans la mémoire principale. Les pages inactives attendent sur le disque. L'***espace d'adressage*** est donc ***virtuel*** sa taille peut être supérieure à celle de la mémoire réelle.

Les processeurs disposent actuellement d'un dispositif, le MMU *« Memory Manager Unit »* qui permet de placer des processus en mémoire sans nécessairement placer les *pages de processus* dans des *cadres de pages* contigus. On distingue les ***adresses logiques*** qui se réfèrent aux pages de processus des ***adresses physiques*** qui se réfèrent aux cadres de pages.



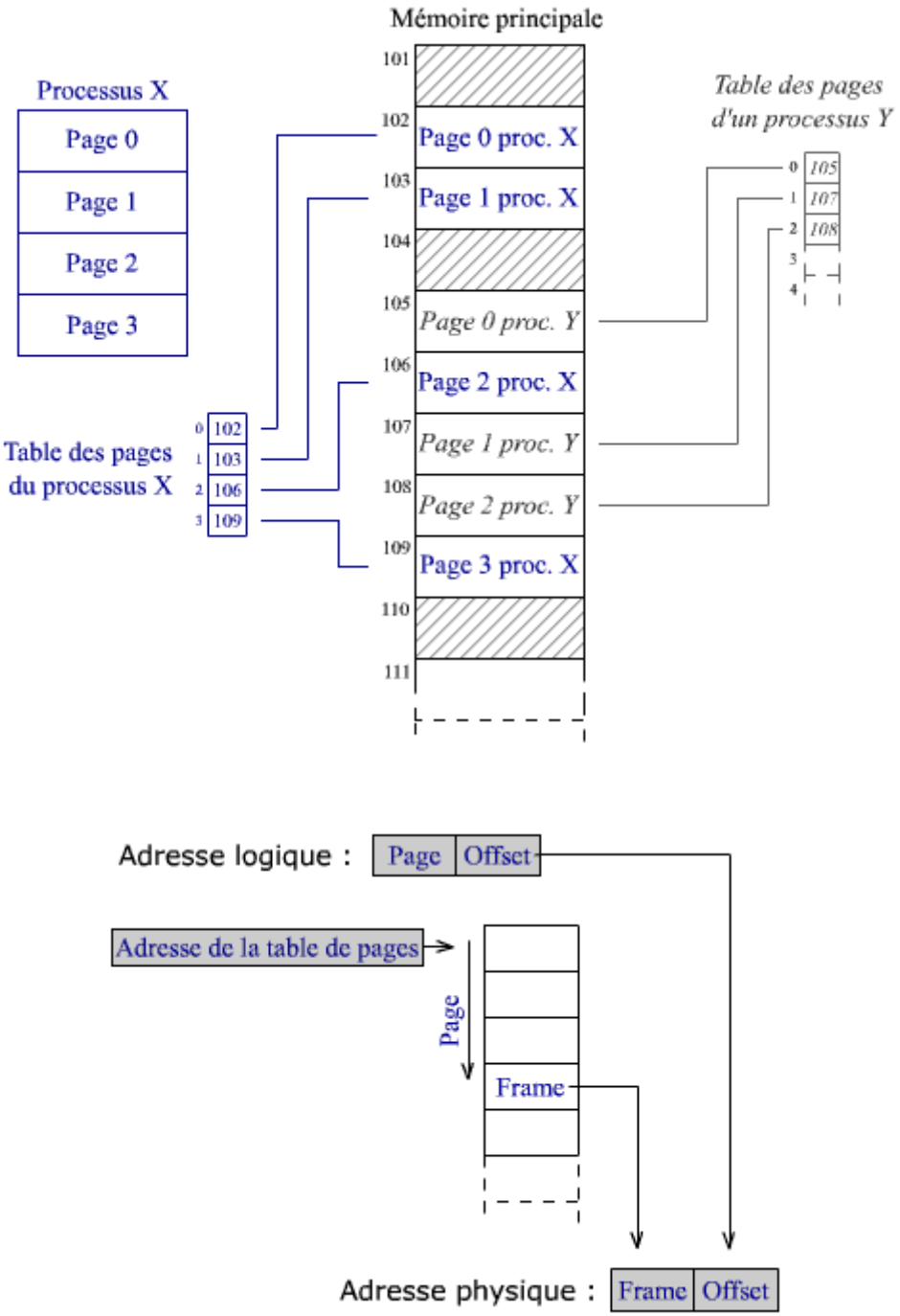
Voyons à présent comment l'unité de gestion mémoire (MMU) met en correspondance les adresses physiques et logiques. Elle contient pour ce faire une *table de pages* où sont inscrits les numéros des cadres de pages.

L'adressage se fait au moyen de *numéros de pages* et d'*offsets*. L'offset (= déplacement ou décalage) est la position relative au début de la page.

L'adresse logique est composée du numéro de page de processus et d'un offset.

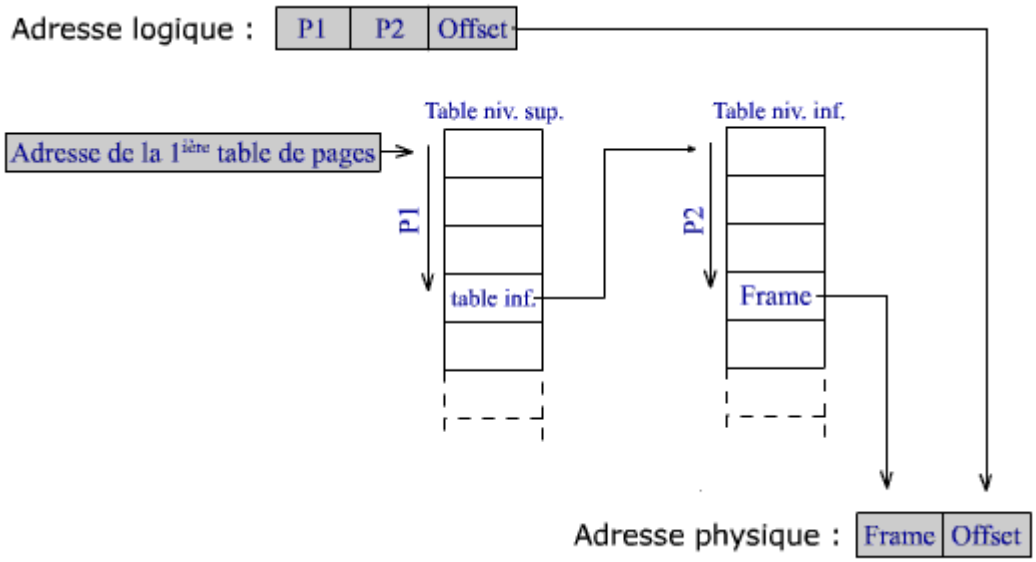
L'adresse physique correspondante est formée à partir du numéro du cadre de page où est chargé la page de processus et du même offset que celui de l'adresse logique.

Le numéro du cadre de page est consigné dans la table de pages On le retrouve en se servant du numéro de page de processus comme d'un index.



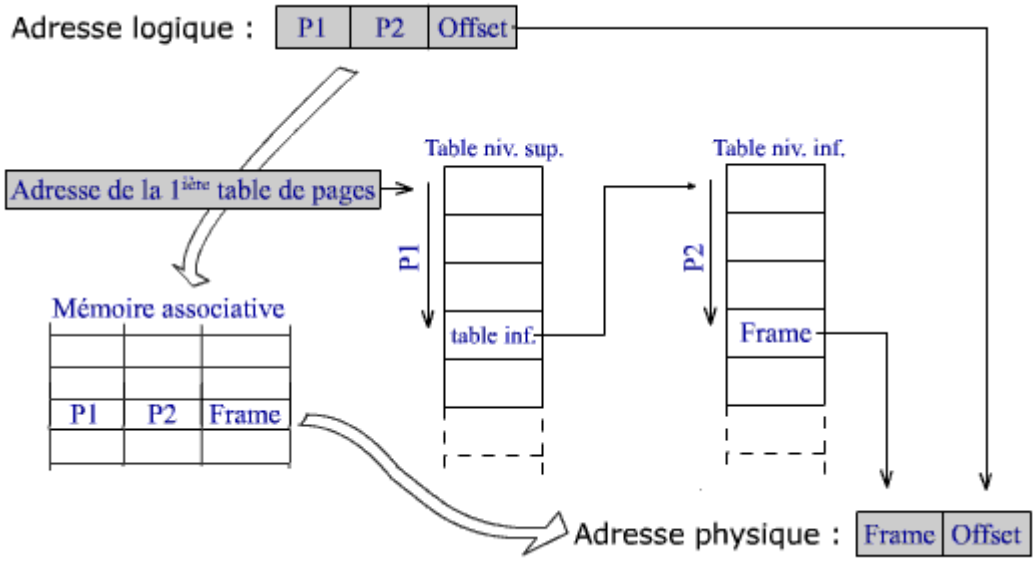
Pagination simple

Le nombre de pages étant souvent très grand les tables des pages deviennent volumineuses et peuvent même occuper … plusieurs pages. On les fractionne donc en plusieurs niveaux : une table de page de niveau supérieur dont chaque élément pointe vers une table de niveau inférieur. L'adresse logique contient dès lors deux nombres pour aboutir au numéro de page. Le premier sert d'index dans la table de niveau supérieur, le second sert d'index dans la table du niveau suivant.



Tables de pages multiniveaux

Ces accès multiples à différentes pages pour aboutir à l'adresse finale ralentissent fortement l'adressage. On évite de répéter ces recherches en notant les correspondances trouvées entre les adresses logiques et les adresses physiques dans une mémoire associative. Ce qui permet ensuite de retrouver presque immédiatement les correspondances les plus récentes.



NB. L'espace d'adressage est perçu par le programmeur comme une suite continue d'octets. La subdivision de l'adresse en numéros de page et d'offset est transparente. Elle est prise en charge par le matériel.

3.3. La segmentation

Chaque processus est constitué d'un ensemble de segments. Chaque segment est un espace linéaire.

Les segments sont des espaces d'adressages indépendants, de différentes longueurs, et qui peuvent même varier en cours d'utilisation. Ils correspondent à des subdivisions logiques déterminées par le programmeur ou par le compilateur.

Les segments contiennent des informations de même nature : le code, les données, la pile, des tables, etc. Il est dès lors possible d'attribuer des protections adaptées à chaque type de segment : un segment de code peut être déclaré en exécution seule, une table de constantes en lecture seule mais pas en écriture ni en exécution. Certaines zones de code en exécution seule peuvent être partagées par plusieurs processus ; cela se fait par exemple pour des bibliothèques de sous-programmes.

L'accès aux segments se fait via une *table de segments*.
Chaque entrée de la table comporte l'adresse de départ du segment et sa taille.
L'adresse logique est constituée du numéro de segment et d'un offset. Contrairement aux pages dont le fonctionnement est transparent pour le programmeur, les segments sont des entités logiques qu'il connaît et manipule. Il distingue les deux informations contenues dans l'adresse : le numéro du segment et l'offset.

Le numéro de segment sert d'index pour retrouver l'adresse du début du segment dans la table de segment. Cet offset doit être inférieur à la taille du segment consignée, elle aussi, dans la table de segments. Si ce n'est pas le cas, une erreur est générée qui provoque l'abandon du programme.. L'offset est ensuite ajouté à l'adresse de début de segment pour former l'adresse physique.

3.4. Segmentation avec pagination

La segmentation et la pagination concernent des problèmes différents. Ce sont deux techniques qui peuvent se combiner.

- La segmentation découpe les processus en zones linaires pouvant être gérées différemment selon que ces segments sont propres au processus, qu'ils sont partagées, lus, écrits ou exécutées et de manière à protéger les processus entre eux.

- La pagination découpe la mémoire en pages non contiguës mais de même taille. Elle procure aux processus des espaces d'adresse continus (nécessaires aux segments). Les pages mémoires peuvent n'être allouées que lorsqu'un processus en a besoin. On obtient de la sorte une mémoire virtuelle de taille supérieure à la mémoire réelle.

Les systèmes d'exploitation qui gèrent ces deux techniques simultanément administrent *une table de segments* et *plusieurs tables de pages*.
Un segment peut contenir plusieurs pages mais toutes ne doivent pas nécessairement être présentes en mémoire à tout moment. On ne garde en mémoire que celles qui sont réellement utilisées. Chaque segment a sa propre table de pages.