Gestion de la mémoire

Plus encore que la gestion du processeur, la gestion de la ressource mémoire est un point fondamental pour les performances globales de l'ordinateur. Le système d'exploitation doit :

- Connaître les zones libre de mémoire physique
- Allouer de la mémoire aux processus (sans gaspillage et avec sécurité)
- Transformer les adresses virtuelles référencées dans un programme en adresses réelles de la mémoire physique
- Offrir une mémoire virtuelle plus grande que la mémoire physique (va-et-vient, pagination)
- Récupérer la mémoire qui se libère lorsqu'un processus se termine

I – Partitions

Comment découper la mémoire centrale pour qu'elle puisse contenir un maximum de programmes? (partitions RAM, mémoire vive)

1) Partitions de taille fixe

La méthode la plus simple consiste à partager la mémoire physique en partitions de tailles fixes (pas nécessairement de tailles identiques), fixées à l'avance, une fois pour toutes. Lors de sa création, un processus dispose de l'un de ces espaces. Si un autre processus est présent en mémoire en même temps que lui, celui-ci disposera d'une autre partition disjointe de la précédente. La protection consiste alors simplement à interdire au processus de construire une adresse en dehors de sa partition (le processeur peut être doté par exemple de deux registres accessibles uniquement en mode noyau : un registre de base contenant la limite inférieure de la partition, un registre de limite contenant la limite supérieure de la partition), le processeur se déroutera vers le système si un processus génère une adresse en dehors de sa partition). L'allocation de la mémoire physique est simple, puisqu'un processus reçoit une portion de mémoire physique de la même taille que son espace d'adressage.

La place occupée par un programme étant inférieure ou égale à la taille de la partition, il reste souvent des espaces libres inoccupés. Il existe plusieurs algorithmes d'attribution des partitions aux processus et le système peut gérer plusieurs queues de processus en attente, en triant les processus selon leur taille, ou une file d'attente unique.

Lorsqu'une partition est libre, le système recherche un processus de la file qui peut se satisfaire de la taille de cette partition, et le charge en mémoire dans cette partition. La recherche peut s'arrêter sur le premier trouvé, ou rechercher celui qui a le plus gros besoin d'espace, tout en étant inférieur à la taille de la partition ; notons que ceci peut entraîner la famine d'un processus qui ne demanderait qu'un tout petit espace.

On voit que des partitions peuvent être libres, alors que d'autres seront surchargées, d'où une utilisation de la mémoire non optimale. Ce type de système appelé MFT (Multiprogrammation avec un nombre Fixe de Tâches) était utilisé sur des mainframes IBM (OS/360 années 80), aujourd'hui très peu de systèmes l'utilisent.

N.B.: on parle de :

- Fragmentation interne quand de la mémoire allouée ne peu pas être utilisée (par exemple lorsqu'un job de 1ko tourne sur une partition de 3ko, il y a 2ko qui ne peuvent être utilisés.
- Fragmentation externe quand de la mémoire non allouée ne peut pas être allouée à un job (par exemple lorsqu'il n'y a en attente que des processus nécessitant de grands espaces mémoire, plusieurs partitions peuvent rester libres alors que des jobs sont en attente.

2) Notion de va-et-vient

Lorsque la mémoire physique n'est pas assez grande pour contenir tous les processus, il faut déplacer sur le disque certains de ces processus, puis les ramener en mémoire centrale pour pouvoir les exécuter. Ce mouvement des processus entre la mémoire centrale et le disque est appelé va-et-vient ou swapping.

3) Partitions de taille variable

Une alternative aux partitions de taille fixe consiste à créer des partitions de taille variable, correspondant à la taille des processus lors de leur chargement. Le nombre, la taille et la position des partitions varient donc au cours du temps. Cela ne résoud le problème que lors de la phase initiale, car dès qu'un processus se termine, il libère sa place mémoire, il apparaît alors rapidement de nombreux trous, pas nécessairement assez grands pour être utilisables. Lorsque la mémoire devient trop fragmentée, on peut effectuer un compactage en déplaçant tous les processus vers le bas de la mémoire. Pour cela le système dispose des éléments suivants :

- Registre de base
- Registre borne (limite supérieure du programme)
- Dispositif de calcul de l'adresse effective pour toute référence à la mémoire
- Dispositif de vérification d'appartenance de l'adresse effective à la zone mémoire du processus

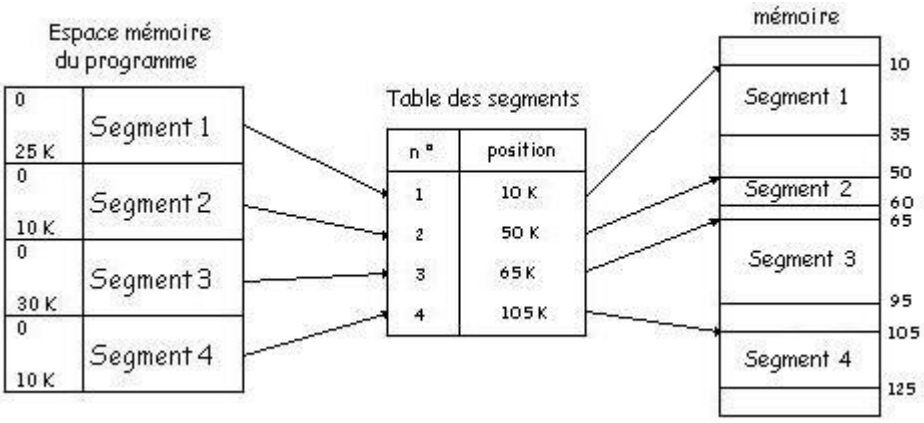
Cette technique est appelée réallocation (relocation) ou translation dynamique (dynamic relocation), c'est une opération coûteuse. Un autre problème vient de la contiguïté des partitions, car l'espace mémoire nécessaire à un processus à tendance à augmenter avec le temps, on peut alors décider d'allouer un espace légèrement plus grand que sa taille lors du chargement.

II – Segmentation

Allouer à chaque processus une zone mémoire d'un seul tenant est une condition souvent difficile à satisfaire d'où l'idée de pouvoir charger un programme dans des zones mémoires non contiguës.

1) Définition – table des segments

La segmentation permet de définir plusieurs espaces d'adresses séparés. Chaque processus dispose de son propre espace mémoire constitué d'un ensemble de segments, chaque segment étant un espace linéaire de taille limitée. Un segment regroupe en général des objets de même nature (instructions, données ou constantes, partagées ou propres, etc.). A chaque processus est associé une table des segments (segments table) contenant les adresses de chargement des segments du processus. L'adresse d'une donnée contient alors deux champs : le numéro de segment, et le déplacement à l'intérieur du segment (offset) c'est à dire l'adresse relative au début du segment. Pour obtenir l'adresse effective il faut donc ajouter l'offset à l'adresse de chargement du segment qui est stockée dans la table des segments.



Exercice : Trouver l'adresse physique correspondant à (2, 3601).
Réponse : 50 Ko + 3601 = 50 x 1024 + 3601 = 54801.
(Le n° 2 dans la table des segments correspond à 50 Ko, et 1 Ko = 1024 Ko)

Le contenu de la table des segments peut varier selon les systèmes, elle contient en général les informations suivantes :

- L'adresse de début qui indique où commence le segment dans l'espace linéaire
- La taille du segment qui indique l'étendu de l'espace mémoire du segment. Le processus doit donner des déplacements dans le segment qui soient inférieurs à cette taille
- Les droits qui indiquent les opérations du processus sur le segment qui sont autorisées

Lorsque le processeur interprète pour le compte d'un processus, une adresse segmentée (s, d) où s désigne le segment, d le déplacement, il lit en mémoire le descripteur du segment et contrôle que les droits sont suffisants. Il compare ensuite d avec le champ taille de ce descripteur. Si la valeur est acceptable, il ajoute d à l'adresse de début du segment et obtient ainsi l'adresse de l'emplacement dans la mémoire linéaire. En général, pour éviter un accès mémoire supplémentaire pour obtenir le descripteur de segment, le processeur dispose de registres spécialisés qui contiennent les descripteurs des derniers segments accédé par le processus.

Le passage systématique par la table des segments permet de gérer facilement le déplacement d'un segment dans la mémoire, puisqu'il suffit ensuite de modifier le champ adresse de début du descripteur pour permettre au processus de continuer son fonctionnement après ce déplacement.

Voici quelques exemples de processeurs disposant de l'adressage segmenté :

- Dans l'intel i286 (années 90), la table des descripteurs de segments d'un processus est divisée en deux parties, une table des segments communs à tous les processus, et une table propre à chaque processus. Chacune de ces tables peut avoir 8192 segments, et chaque segment peut avoir jusqu'à 64 Ko. La mémoire linéaire est la mémoire physique, et peut atteindre 16 Mo
- L'intel i386 est assez voisin de l'i286, du point de vue adressage segmenté, si ce n'est que les segments peuvent atteindre 4 Go. La mémoire linéaire peut de plus être paginée
- Le Pentium a 16 K segments indépendants atteignant au plus 4 Go, la mémoire peut être de plus paginée.

2) Algorithmes d'allocation

Un des problèmes du système d'exploitation est l'allocation des segments libres de la mémoire physique pour y loger les segments des processus. Il existe plusieurs algorithmes d'allocation des segments :

- Première zone libre (first fit) : le système parcourt la liste des segments jusqu'à trouver un segment libre assez grand, il prend le 1er qu'il trouve. S'il reste de l'espace libre, la zone est divisée en deux parties : une pour le processus, l'autre comme nouvel espace mémoire libre
- Zone libre suivante (next fit) : c'est le même principe que l'algorithme précédent, sauf qu'il démarre la recherche à partir de l'endroit où il s'était arrêté la fois précédente, au lieu de recommencer la recherche au début
- Meilleur ajustement (best fit) : il recherche parmi tous les segments libres le plus petit qui convient, pour réduire la taille des trous. En fait à l'usage, cet algorithme est moins bon que les précédents car il est plus lent et génère des trous inutilisables
- Plus grand résidu (worst fit) : il recherche parmi tous les segments libres le plus grand qui convient, pour augmenter la taille des trous et les rendre aussi utilisable. Des simulations ont montré que ce n'est pas non plus la meilleure solution
- Placement rapide (quick fit) : le système gère deux listes, une pour les segments occupés, l'autre pour les segments libres, elle-même divisée en listes séparées pour les tailles les plus demandées. La recherche est plus simple, mais la fusion des blocs libérés est plus compliquée.

Exemple :
On considère une mémoire ayant les zones libres suivantes (dans l'ordre) 2 Ko, 10 Ko, 8 Ko, 5 Ko, 12 Ko, 13 Ko.
Indiquer quelle zone va être occupée lors d'une requête de 7 Ko suivant les différents algorithmes ci-dessus (excepté le dernier), pour le « next fit », on suppose que lors de la précédente requête on est resté sur 5 Ko.
Solution : first fit => 10 Ko ; next fit => 12 Ko ; best fit => 8 Ko ; worst fit => 13 Ko

III – Pagination

L'idée de mémoire virtuelle (virtual memory) est de donner l'illusion d'une mémoire sans limite. Le rôle de la mémoire virtuelle est de permettre l'exécution de plusieurs processus qui nécessiteraient un espace mémoire total supérieur à l'espace mémoire réel (mémoire vive et cache). Notons que dans ce but, il faut quand même avoir suffisamment de mémoire secondaire (disques) pour stocker les programmes et les données en entier. Au début de l'informatique, lorsqu'un programme était trop grand pour l'espace mémoire physique disponible, il fallait le découper en modules (overlays) chargeables séparément, ce découpage à base de recouvrement était effectué par le programmateur et représentait une tâche complexe. Aujourd'hui avec la pagination, c'est le système d'exploitation qui permet l'exécution d'un processus plus grand que la mémoire physique disponible.

1) Pagination

La mémoire physique est découpée en blocs de taille fixe (puissance de 2, de 512 octets à 64 Ko), appelés pages réelles (page frames) ou cases. La mémoire linéaire des processus, mémoire virtuelle, est découpée en blocs de même taille fixe, appelés pages. Chaque page peut alors être placée dans une case quelconque. Une table de correspondance (une par processus), appelée table des pages (page table) est gérée par le système d'exploitation.

Le mécanisme de traduction des adresses virtuelles en adresses réelles doit associer à chaque numéro de page virtuelle le numéro de case réelle qui contient cette page, si elle existe. Cette traduction est assurée par le MMU (Memory Management Unit). Un registre spécialisé du processeur contient l'adresse de la table des pages des processus. En général, l'adresse de cette table est une adresse physique. Chaque entrée de cette table contient les informations suivantes :

- Le bit de présence qui indique s'il y a une case allouée à cette page
- Le numéro de case (numéro de case réelle alloué à la page)
- Les bits de protection (indique les opérations autorisées sur cette page par le processus)
- Le bit de page référencée (bit R) positionné à 1 lors d'un accès à la page en lecture/écriture
- Le bit de page modifié (bit M) positionné à 1 lors d'un accès en écriture (page modifié)

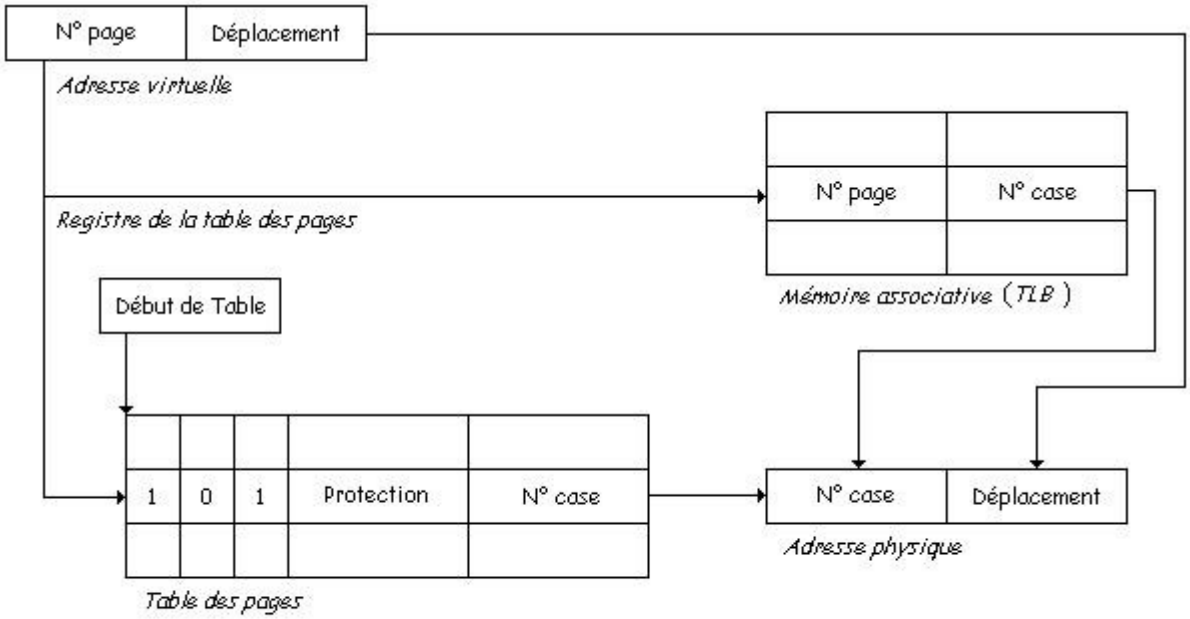
Ces deux derniers indicateurs servent à la gestion de la mémoire physique, le bit R sert pour un des algorithmes que nous verrons par la suite. Ces bits seront remis à zéro par le système d'exploitation lorsque la page modifiée aura été récrite pour le bit M par exemple, au bout d'un certain intervalle d'horloge pour le bit R.

On pourra donc avoir tous les cas suivants :

R	0	0	1	1
M	0	1	0	1

Lorsque le processeur traduit une adresse virtuelle, il en isole le numéro de page virtuelle qu'il utilise comme déplacement dans cette table des pages pour en trouver l'entrée correspondante. Si l'indicateur de présence est positionné, il contrôle la validité de l'accès vis à vis des indicateurs de protection. Si l'accès est accepté, il concatène le numéro de case au déplacement virtuel dans la page pour obtenir l'adresse en mémoire physique de l'emplacement recherché. Pour éviter l'accès mémoire supplémentaire à la table des pages, une mémoire associative appelée aussi TLB (Translation Lookaside Buffer) contient les entrées de la table correspondant aux derniers accès.

La figure suivante montre le fonctionnement de ce mécanisme.



Fonctionnement de la pagination à un niveau

Remarque 1 : Une page réelle peut correspondre à deux pages virtuelles pour permettre à deux programmes différents de partager les données ou le code.
Remarque 2 : La pagination à un niveau d'un espace virtuel de grande taille peut conduire à des tailles importantes de la table des pages. Dans la pagination à deux niveaux, la mémoire virtuelle est divisée en hyperpages de taille fixe, chaque hyperpage étant découpée en pages de même taille qu'une page réelle de mémoire physique. Cette pagination réduit la représentation de la table des pages d'un processus (à chaque hyperpage, on associe une table de pages, et la table des hyperpages permet de localiser la table des pages de chaque hyperpage).

2) Pagination à la demande

La mémoire virtuelle peut être de taille beaucoup plus grande que la mémoire physique. Il faut alors mettre en mémoire physique uniquement les pages de mémoire virtuelle dont les processus ont besoin pour leur exécution courante, les autres étant conservées sur mémoire secondaire (disque).

Lorsqu'un processus accède à une page qui n'est pas en mémoire physique, l'instruction est interrompue, et un déroutement au système est exécutée. On dit qu'il y a défaut de page. Le système doit alors charger cette page, cette méthode qui consiste à charger une page uniquement lorsqu'elle est demandée s'appelle pagination à la demande (demand paging).

Lorsqu'un défaut de page se produit, s'il y a une case libre, le système peut allouer cette case. Sinon il faut réquisitionner une case occupée par une autre page. On dit qu'il y a remplacement de page. Si la page remplacée a été modifiée, il faut auparavant la récrire en mémoire secondaire. Il existe plusieurs algorithmes de remplacement de pages.

- L'algorithme optimal consisterait à enlever la page qui sera accéder dans un avenir le plus lointain possible. C'est un algorithme théorique puisqu'il n'est pas possible de prévoir les accès futur des processus. Il sert d'algorithme de référence pour évaluer la performance des autres algorithmes.
- L'algorithme de fréquence d'utilisation (LFU : Liste Frequency Used) consiste à remplacer la page qui a été la moins fréquemment utilisée. Il faut maintenir une liste des numéros de page ordonné par leur fréquence d'utilisation.
- L'algorithme chronologique d'utilisation (LRU : Last Recently Used) consiste à remplacer la page qui n'a pas été utilisée depuis le plus longtemps. Il faut maintenir une liste des numéros de pages ordonnées par leur moment d'utilisation.
- L'algorithme chronologique de chargement (FIFO) consiste à remplacer la page en mémoire depuis le plus longtemps. Il suffit d'avoir une file des numéros de page dans l'ordre où elles ont été amenées en mémoire.
- L'algorithme de seconde chance. C'est un dérivé de l'algorithme FIFO lors d'un remplacement si la page la plus ancienne a son bit référencé à 1 on ne l'enlève pas (on lui donne une seconde chance), sa date de chargement est remplacée par l'heure actuelle et son bit R est mit à 0. L'algorithme continue et recommence avec la nouvelle dernière page jusqu'à en trouver une qui ait son bit R à 0, elle sera alors remplacée. Cela évite de remplacé des pages utilisées sur de longue périodes.
- L'algorithme de l'horloge, c'est le même principe que l'algorithme précédent seule l'implémentation (la façon dont c'est programmé) change

Exemple : On considère une machine ayant 4 cases. Pour chaque page on donne le moment du chargement, le moment du dernier accès, le bit R et le nombre de fois où la page a été utilisée depuis le chargement (les temps sont en tops d'horloge).

Page	Chargement	Dernière référence	Bit R	Nombre d'utilisation
0	136	280	1	2
1	203	265	0	2
2	167	270	0	4
3	165	285	1	1

Si un défaut se produit, indiquer la page qui sera remplacée suivant l'algorithme FIFO, LRU, LFU, horloge.
Réponse : FIFO : 1, LRU : 0, LFU : 3, horloge : 2.

3) Espace de travail

On appelle ensemble de travail ou espace de travail (working set) l'ensemble des pages dont a besoin un processus à un instant donné pour s'exécuter. Dans la pagination à la demande, aucune page n'est chargée à l'avance. Une alternative consiste à précharger les pages correspondant à l'ensemble de travail, on appelle cela le préchargement (preparing). Cette approche est appelé modèle de l'ensemble de travail (working set model).

Si l'espace de travail nécessaire est plus grand que le nombre de pages réelles allouées, les défauts de pages seront fréquents. Lorsqu'un processus provoque des défauts de pages trop fréquents on dit qu'il s'écroule (trashing). Il peut donc être intéressant d'évaluer la taille de l'espace de travail d'un processus. Si $T(P_i)$ est la taille de l'espace de travail d'un processus P_i, il faut déterminer l'ensemble E des processus tel que : $\sum_{P_i \in E} T(P_i) \leq M$ où M est la taille totale de la mémoire disponible.

4) Pagination et segmentation

On peut combiner segmentation et pagination. L'espace des adresses virtuelles est découpé en segments, les segments sont découpés en pages (on aura alors trois champs dans une adresse : le numéro de segment, le numéro de page, le déplacement dans la page). Chaque processus a sa table des segments, chaque segment a sa table des pages.

Dans le Pentium d'Intel, par exemple, il y a une table des descripteurs globaux (GDT) partagée par tous les programmes qui décrit les segments système et des tables des descripteurs locaux (LDT), une par programme, qui décrivent les segments locaux de chaque programme. Puis il y a une pagination (si elle est activée) à deux niveaux : chaque segment a un répertoire des pages qui a selon la taille une ou plusieurs entrées, chaque entrée pointant sur une table des pages.

5) Conclusion

Les trois problèmes à résoudre sont la définition de l'espace d'adresses des processus, la protection de leurs objets propres et l'attribution d'une partie de la mémoire physique à chacun d'eux.

Dans le partitionnement, les processus reçoivent une partie contiguë de la mémoire physique qui constitue leur espace d'adresses.

La mémoire segmentée fournit aux processus un espace d'adresses à deux dimensions leur permettant de regrouper ensembles des objets de même nature. Cela simplifie la gestion de leur mémoire virtuelle.

La pagination est un mécanisme qui permet de plaquer la mémoire virtuelle sur la mémoire réelle, en les découpant respectivement en pages et en cases de même taille, et en mettant les pages dans n'importe qu'elles cases sans devoir respecter la contiguïté. La pagination peut être à un niveau ou à plusieurs niveaux. La pagination à la demande consiste à n'allouer une case à une page que lorsqu'un processus en a besoin. Cela implique de mettre en oeuvre un algorithme de remplacement lorsque toutes les cases sont occupées.

La segmentation et la pagination peuvent être utilisées conjointement.

Chaque processus a besoin d'un ensemble de pages pour s'exécuter, et qui constitue son espace de travail. Pour un fonctionnement optimal, le nombre de cases dont dispose un processus doit être égal à la taille de cet espace de travail. S'il est plus grand, la mémoire est mal utilisée. S'il est plus petit, le nombre de défaut de pages augmente, ce qui peut conduire à l'écroulement du système.