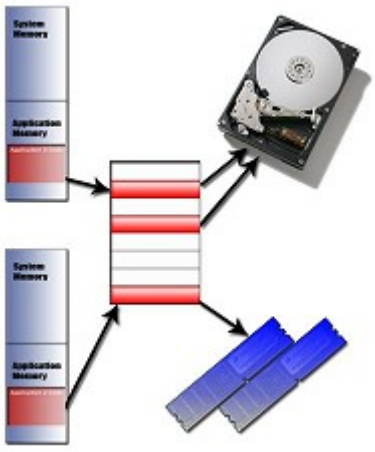
Gestion de la mémoire

Table des matières

Le gestionnaire de mémoire est un sous-ensemble du système d'exploitation. Son rôle est de partager la mémoire entre l'OS 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).



1. Introduction

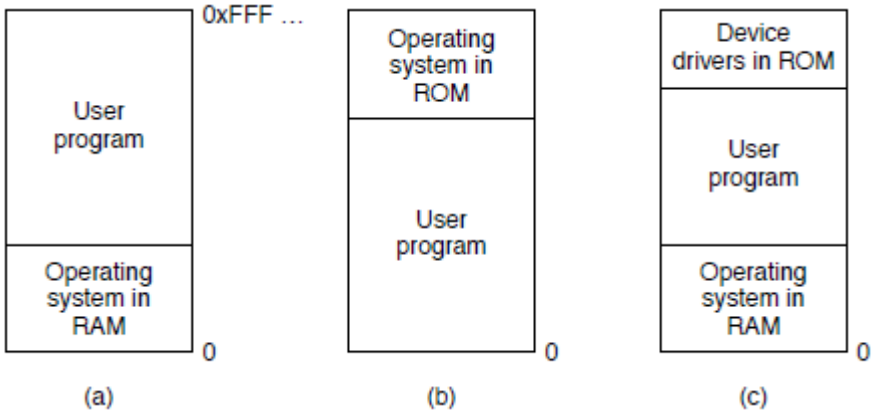
On peut subdiviser les systèmes de gestion de la mémoire en deux catégories :

- les systèmes qui peuvent déplacer les processus en mémoire secondaire pour trouver de l'espace de swap[1],
- les systèmes qui n'utilisent pas la mémoire secondaire.

1.1. Monoprogrammation sans va-et-vient ni pagination

L'approche la plus simple pour gérer la mémoire consiste à n'accepter qu'un seul processus à la fois (monoprogrammation) auquel on permet d'utiliser toute la mémoire disponible en dehors de celle qu'utilise le système.

Cette approche était utilisée, par exemple, dans les premiers micro-ordinateurs IBM PC utilisant MS-DOS comme système d'exploitation (c).

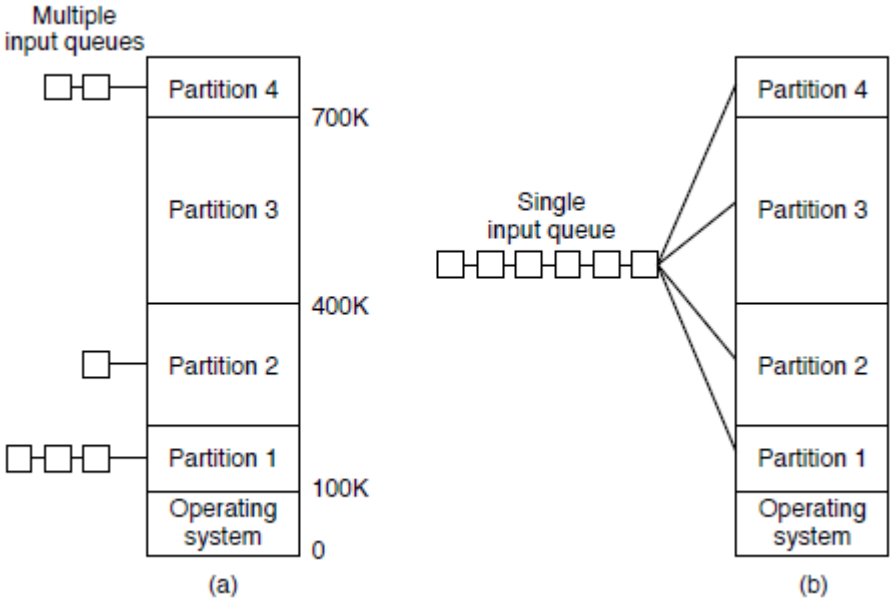


1.2. Multiprogrammation avec des partitions fixes

Dans un système multiprogrammé, il faut gérer la répartition de l'espace entre les différents processus. Cette partition peut être faite une fois pour toute au démarrage du système par l'opérateur de la machine, qui subdivise la mémoire en partitions fixes.

Chaque nouveau processus est placé dans la file d'attente de la plus petite partition qui peut le contenir (a). Cette façon de faire peut conduire à faire attendre un processus dans une file, alors qu'une autre partition pouvant le contenir est libre. L'alternative à cette approche consiste à n'utiliser qu'une seule file d'attente : dès qu'une partition se libère, le système y place le premier processus de la file qui peut y tenir (b).

1 va-et-vient



2. Swaping

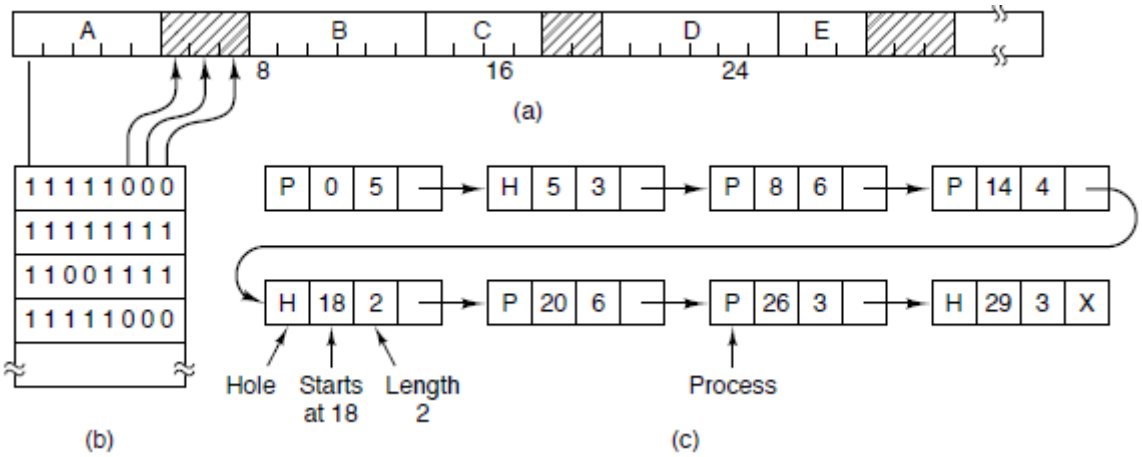
Dans un système qui peut déplacer les processus sur le disque quand il manque d'espace, le mouvement des processus entre la mémoire et le disque (va-et-vient) risque de devenir très fréquent, si on n'exploite pas au mieux l'espace libre en mémoire principale. Sachant que les accès au disque son très lents, les performances du système risquent alors de se détériorer rapidement. Il faut alors exploiter au mieux l'espace libre en mémoire principale, en utilisant un partitionnement dynamique de la mémoire.

Le fait de ne plus fixer le partitionnement de la mémoire rend plus complexe la gestion de l'espace libre. Celui-ci doit cependant permettre de trouver et de choisir rapidement de la mémoire libre pour l'attribuer à un processus. Il est donc nécessaire de mettre en œuvre des structures de données efficaces pour la gestion de l'espace libre.

2.1. Gestion de la mémoire par tables de bits

La mémoire est divisée en unités (quelques mots mémoire à plusieurs kilo-octets), à chaque unité on fait correspondre dans une table de bits :

- la valeur 1 si l'unité mémoire est occupée ;
- la valeur 0 si l'unité mémoire est libre.



(a) Une partie de la mémoire occupée par cinq processus (A,B,C,D et E) et trois zones libres. Les régions hachurées sont libres. (b) La table de bits (0 = libre, 1 = occupée). (c) Liste chainée des zones libres.

Pour allouer k unités contiguës, le gestionnaire de mémoire doit parcourir la table de bits à la recherche de k zéro consécutifs. Cette recherche s'avère donc lente pour une utilisation par le gestionnaire de la mémoire et est rarement utilisée à cet effet.

2.2. Gestion de la mémoire par listes chaînées

Une autre solution consiste à chaîner les segments libres et occupés. La figure ci-dessus montre l'exemple d'un tel chaînage, les segments occupés par un processus sont marqués (P) les libres sont marqués (H). La liste est triée sur les adresses, ce qui facilite la mise à jour.

Lorsqu'on libère la mémoire occupée par un segment, il faut fusionner le segment libre avec le ou les segments adjacents libres s'ils existent :

Ex : Quatre combinaisons de voisins possibles d'un processus X qui termine et libère le segment qu'il occupe

3. La mémoire virtuelle

Le principe de la mémoire virtuelle consiste à considérer un espace d'adressage virtuel supérieur à la taille de la mémoire physique, sachant que dans cette espace d'adressage, et grâce au mécanisme de va-et-viens sur le disque, seule une partie de la mémoire virtuelle est physiquement présente en mémoire principale à un instant donné. Ceci permet de gérer un espace virtuel beaucoup plus grand que l'espace physique sans avoir à gérer les changements d'adresses physiques des processus après un va-et-vient : car même après de multiples va-et-vient un processus garde la même adresse virtuelle. C'est notamment très utile dans les systèmes multiprogrammés dans lesquels les processus

qui ne font rien (la plupart) occupent un espace virtuel qui n'encombre pas nécessairement l'espace physique.

3.1. Pagination

La plupart des architectures actuellement utilisées reposent sur des processeurs permettant de gérer un espace virtuel paginé : l'espace d'adressage virtuel est divisé en pages, chaque page occupée par un processus est soit en mémoire physique soit dans le disque (va-et-vient).

La correspondance entre une page virtuelle et la page physique à laquelle elle peut être associée (si elle est en mémoire physique) est réalisée en utilisant une table de pages :

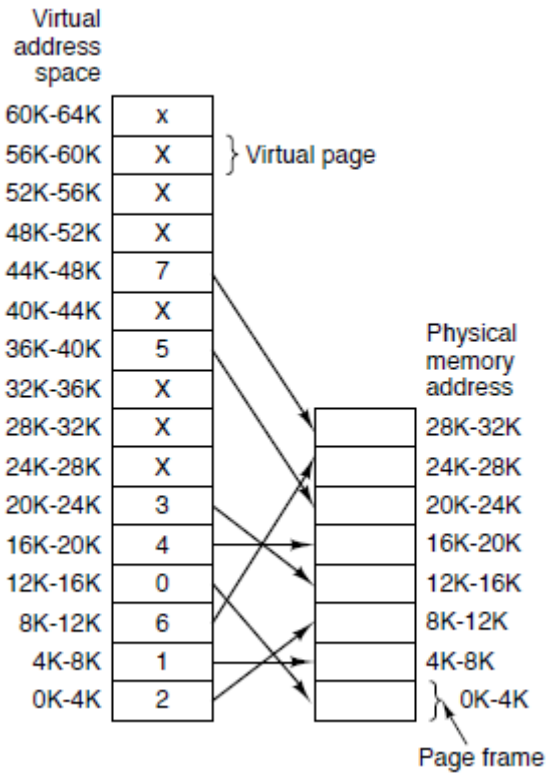


Table de pages

Lorsque le processeur doit exécuter une instruction qui porte sur un mot mémoire donné dont il a l'adresse virtuel, il cherche dans la table des pages l'entrée qui correspond à la page contenant le mot. Si la page est présente en mémoire il lit le mot, sinon il déclenche un déroutement de type défaut de page. A la suite de ce déroutement, le système de gestion de la mémoire est appelé afin de charger la page manquante à partir du disque (va-et-vient).

Chaque entrée de table de pages consiste en un descripteur de page qui contient généralement ces informations (au moins) :

- Numéro de la page en mémoire physique (cf. figure 5.5) si celle-ci est présente en mémoire.
- Un bit qui indique la présence (ou non présence) de la page en mémoire.
- Un bit de modification : qui mémorise le fait qu'une page a été modifiée, ceci permet au gestionnaire de mémoire de voir s'il doit la sauver sur le disque dans le cas ou il veut récupérer l'espace qu'elle occupe pour charger une autre page.
- Un bit de référencement : qui est positionné si on fait référence à un mot mémoire de la

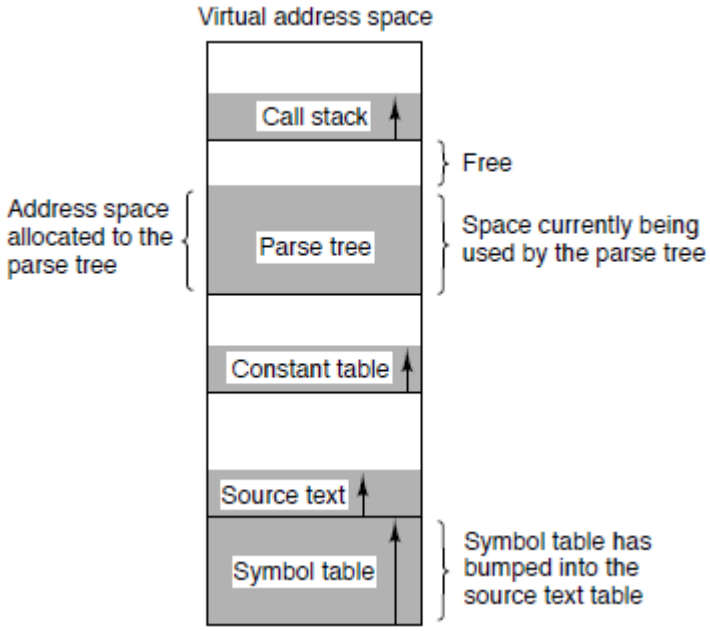
page, ceci permet au système de trouver les pages non référencées qui seront les meilleures candidates pour être retirées de la mémoire physique s'il y a besoin et manque de mémoire physique.

4. La segmentation

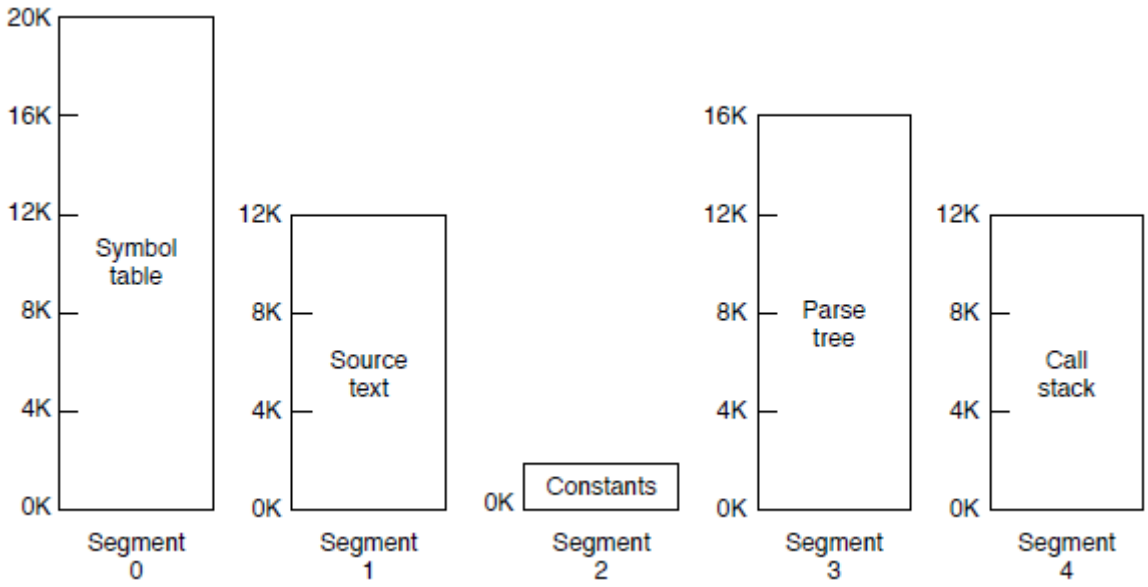
4.1. Implantation de segments purs

La segmentation de la mémoire permet de traiter la mémoire non plus comme un seul espace d'adressage unique, mais plutôt comme un ensemble de segments (portions de la mémoire de taille variable), ayant chacune son propre espace d'adressage. Ceci permet aux processus de simplifier considérablement le gestion de mémoire propre.

Ainsi un processus qui utilise différentes tables : table des symboles, code, table des constantes, etc. doit se préoccuper de la position relative de ses tables et doit gérer les déplacements de tables quand celles-ci croissent et risquent de se recouvrir :



Alors que dans une gestion de la mémoire segmentée, il lui suffit d'allouer un segment pour chaque table, la gestion de la position des segments dans l'espace d'adressage de la mémoire physique devient à la charge du système :



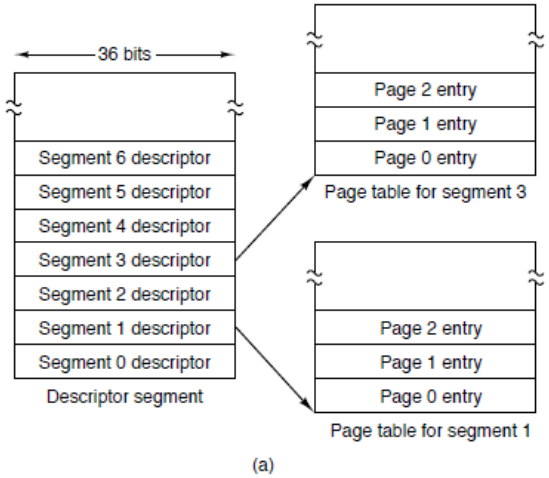
La segmentation permet aussi de faciliter le partage de zones mémoire entre plusieurs processus.

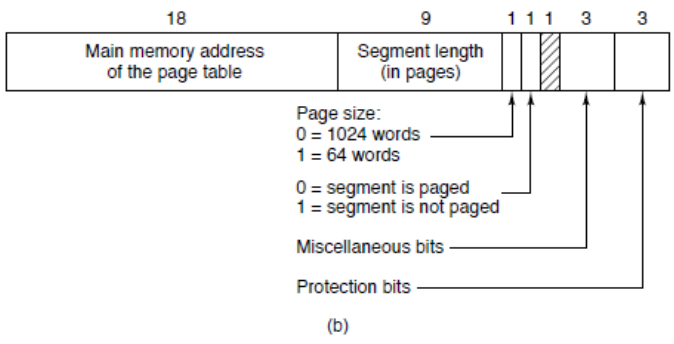
L'implémentation d'un système utilisant la segmentation pure (sans pagination) pose le problème de la fragmentation de la mémoire. La fragmentation est due au fait que les segments (contrairement aux pages) sont de taille non fixe.

4.2. Segmentation avec pagination

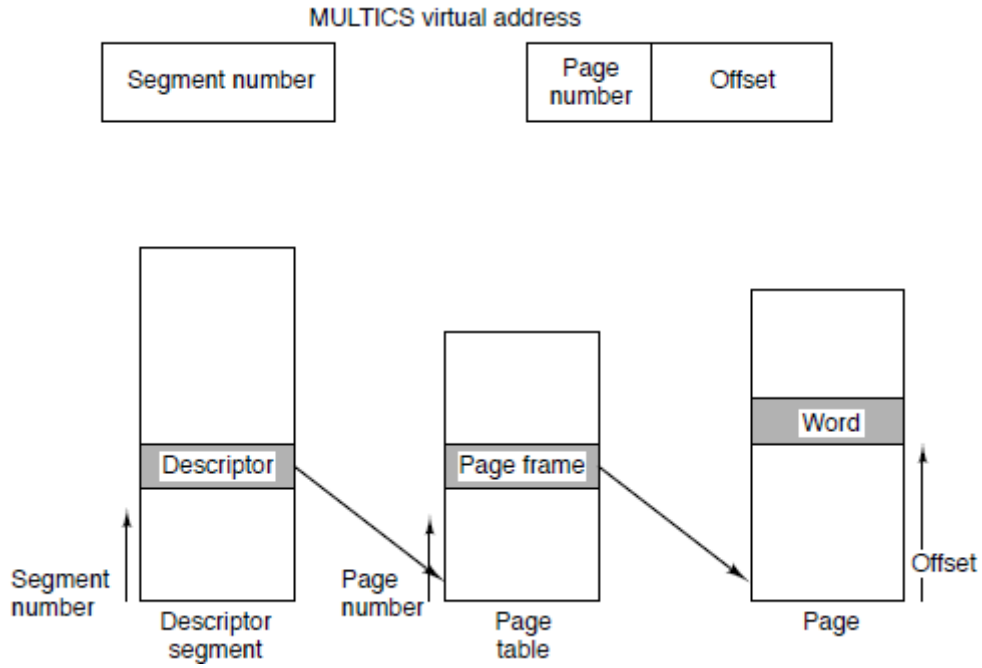
L'apparition du système d'exploitation MULTICS dans les années soixante a permis d'introduire divers concepts novateurs encore utilisés dans les systèmes les plus récents. Parmi ces idées, on trouve le principe de la segmentation avec pagination. En combinant segmentation et pagination MULTICS combine les avantages des deux techniques en s'affranchissant des principaux défauts qu'ils ont : fragmentation de la mémoire pour la segmentation, et espace d'adressage unique pour un système utilisant un adressage virtuel paginé.

Sous MULTICS chaque segment possède son propre espace d'adressage paginé. Ainsi chaque segment possède sa propre table de pages (a). Le descripteur de chaque segment contient, en plus de l'adresse de la table de pages qui lui est associée, la taille du segment, la taille des pages (1024 mots ou 64 mots), un indicateur permettant de paginer ou non l'espace d'adressage du segment, des bits de protection, ainsi que d'autres informations (b).





Ainsi, pour trouver une case mémoire, MULTICS doit consulter d'abord la table des segments, puis la table des pages, et enfin accéder à la page physique si elle est présente en mémoire :



Afin d'accélérer le processus, une mémoire associative permet de conserver les numéros des pages physiques correspondants aux pages les plus récemment référencées à partir de leur numéro de segment et de page :

Comparison field					Is this entry used?
Segment number	Virtual page	Page frame	Protection	Age	
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1