

# Allocation

Le système d'exploitation, les processus systèmes et utilisateurs se trouvent dans la mémoire, il faut donc un mécanisme de protection pour isoler les processus. Ce mécanisme, c'est le MMU vu plus tôt.

## Mono-programmation

Lorsqu'un seul processus s'exécute à la fois en même temps que le système d'exploitation, c'est le cas sur les systèmes très rudimentaires comme MS-DOS.

Étant donné qu'il n'y a qu'un seul processus à la fois, le système d'exploitation n'a pas beaucoup à faire, car il n'y a pas besoin d'isoler la mémoire (sauf pour la petite partie réservée au système d'exploitation).

Le BIOS réalise une gestion des périphériques.

## Multi-programmation

### Partitions fixes

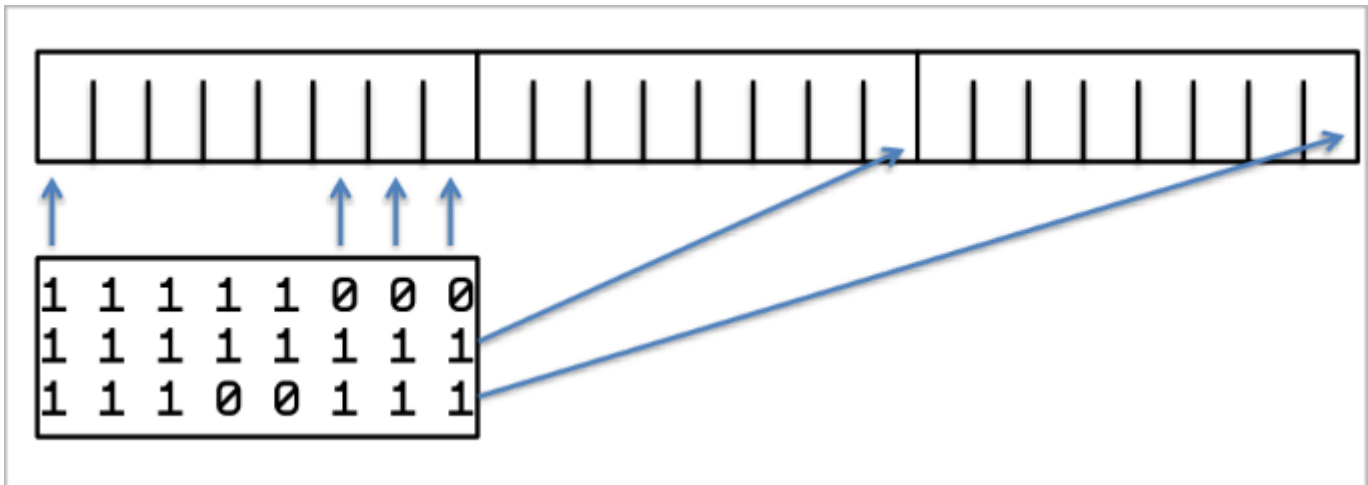
On peut pré-découper la mémoire en morceaux de taille variable (les partitions). On va donc tenter de placer chaque processus dans la plus petite partition qui peut la contenir.

L'un des problèmes de ce système est qu'il va y avoir beaucoup de restes (fragmentation interne) car si un a besoin de quatre unités, mais que la seule partition que l'on peut prendre en contient 8, on a quatre unités non utilisées.

### Partitions variables

On alloue l'espace **selon les besoins des processus**, on va créer des partitions en fonction des demandes faites par les processus.

### Table de bits



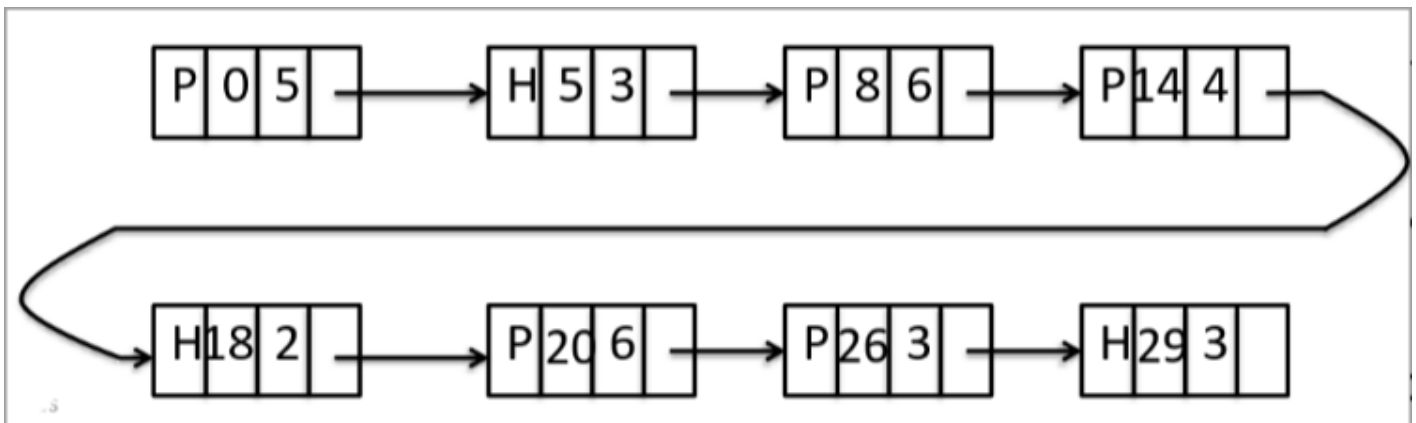
La table de bits correspond à une cartographie de la mémoire découpée en blocs d'allocations de taille fixe. Pour chaque bloc, on va noter un 1 si c'est occupé, ou un 0 si le bloc est libre dans la table de bits.

Ainsi, il suffit de seulement stocker 1 bit par unité d'allocation. Cependant, le problème est que si les unités d'allocations sont petites, alors la table sera grande, mais ce sera plus précis.

À l'inverse, si les unités d'allocations sont grandes, la table sera plus petite, mais sera moins précise (ce qui implique donc un plus grand gaspillage de mémoire).

Un autre problème est que plus la taille est grande, moins l'allocation sera rapide, car il faudra parcourir toute la table jusqu'à trouver le segment d'unités libre recherché.

## Liste chaînée



Une autre méthode correspond à conserver une liste chaînée des partitions mémoire. Cette liste est généralement triée suivant les adresses des partitions libres, ainsi les partitions qui se suivent dans la mémoire se suivront dans la liste.

Chaque élément de la liste chaînée n'a alors qu'à stocker quatre informations : si le bloc est libre ou pas, la position du début de la partition, la longueur de la partition et le lien avec la partition suivante. Par exemple **P86** signifie qu'il y a un processus sur la partition en position huit d'une longueur de six unités.

Ce qui implique également de faciliter la fusion de blocs libre, si un programme libère une partition, cette partition pourra être facilement fusionnée avec les éventuelles partitions libres adjacentes pour former une partition plus grande.

L'avantage est de rendre l'allocation beaucoup plus rapide, le désavantage est que cela crée de la fragmentation.

Pour en savoir plus, vous pouvez regarder sur [ce site](#).

## Choix de la partition

Lorsque l'on recherche la partition de mémoire qui pourra accueillir un programme, il y a plusieurs algorithmes possibles.

- L'algorithme **first-fit** qui correspond à prendre la première partition libre trouvée étant suffisamment grande pour contenir le programme. C'est souvent cet algorithme qui est utilisé, car il est très rapide.
- L'algorithme **best-fit** qui correspond à parcourir toute la mémoire à la recherche de la meilleure partition. Cependant, cette méthode est assez peu efficace et crée beaucoup de fragmentation externe en créant des bouts de partitions trop petites pour être utilisées
- L'algorithme **worst-fit** qui correspond à de nouveau parcourir toute la mémoire, mais cette fois-ci à la recherche de la plus grande zone disponible afin d'éviter de créer trop de *fragmentation externe* comme le best-fit. Cet algorithme est particulièrement rapide pour les listes triées par taille.

## Fragmentations

- **La fragmentation interne** survient lorsque la mémoire est divisée en unité d'allocation de taille fixe. Elle provient de la mémoire allouée par le SE, mais pas demandée au processus. Le SE alloue donc un surplus de mémoire par rapport à la demande du processus.
- **La fragmentation externe** survient suite aux allocations et libérations successives, la mémoire ressemble alors à un gruyère.

## Solutions à la fragmentation

Pour rassembler les zones mémoires et résoudre ce problème de fragmentation, on peut utiliser le **compactage**. Cela consiste à rassembler les zones mémoires occupées et les zones libres ensembles de façon à créer de grands espaces libres.

Cela est cependant uniquement possible dans le cas de la translation à l'exécution et c'est un mécanisme assez coûteux en ressource.

Une autre solution est d'utiliser de la **pagination** que nous allons voir en détail dans la section suivante.

# Swapping

Pour pouvoir s'exécuter un processus doit se trouver entièrement en mémoire, alors lorsqu'il y a beaucoup de processus en mémoire et peu de mémoire disponible, on peut mettre un processus qui ne s'exécute pas (état ready) sur le disque dur pour libérer de la mémoire (via une partition sur le disque dur ou via un fichier "swap file").

Un processus peut être swappé lorsque son quantum de temps a expiré (retour de running à ready), si le quantum est petit il y aura beaucoup de changement de processus et donc beaucoup d'accès au disque en revanche si le quantum est grand, il y aura moins de changements de processus et moins d'accès au disque.

Les performances de ce mécanisme sont de ce fait déterminées par le temps de transfert mémoire ↔ disque.

On ne peut swap que les processus auquel on n'a pas besoin d'accéder à la mémoire, c'est-à-dire les processus en état **ready** car pour les processus en état **waiting**, le système d'exploitation doit pouvoir accéder à la mémoire du processus pour y faire les entrées-sorties.

---

Revision #7

Created 3 January 2024 10:25:12 by SnowCode

Updated 6 January 2024 19:13:15 by SnowCode