Système partie 1 - cours 4

Pastille 1 - le swapping

N. de Rugy-Altherre - Vincent Colotte

Introduction

Pour s'exécuter, les instructions d'un programme doivent être en mémoire centrale. Le système doit gérer l'allocation de la mémoire aux programmes utilisateurs. Dans des systèmes multiprogrammés, plusieurs processus peuvent se trouver en mémoire centrale, prêts à être exécutés. Plusieurs questions se posent alors :

- Comment gérer la répartition de la mémoire entre les différents processus ?
- Que faire lorsqu'il n'y a plus de place en mémoire centrale ?
- Comment gérer la libération de la mémoire par un processus ?
- Comment gérer la protection d'accès à la mémoire ?
- ...

Lorsque qu'un processus est chargé en mémoire, les systèmes actuels utilisent deux stratégies pour gérer la mémoire : le va-et-vient (swapping) et la mémoire virtuelle.

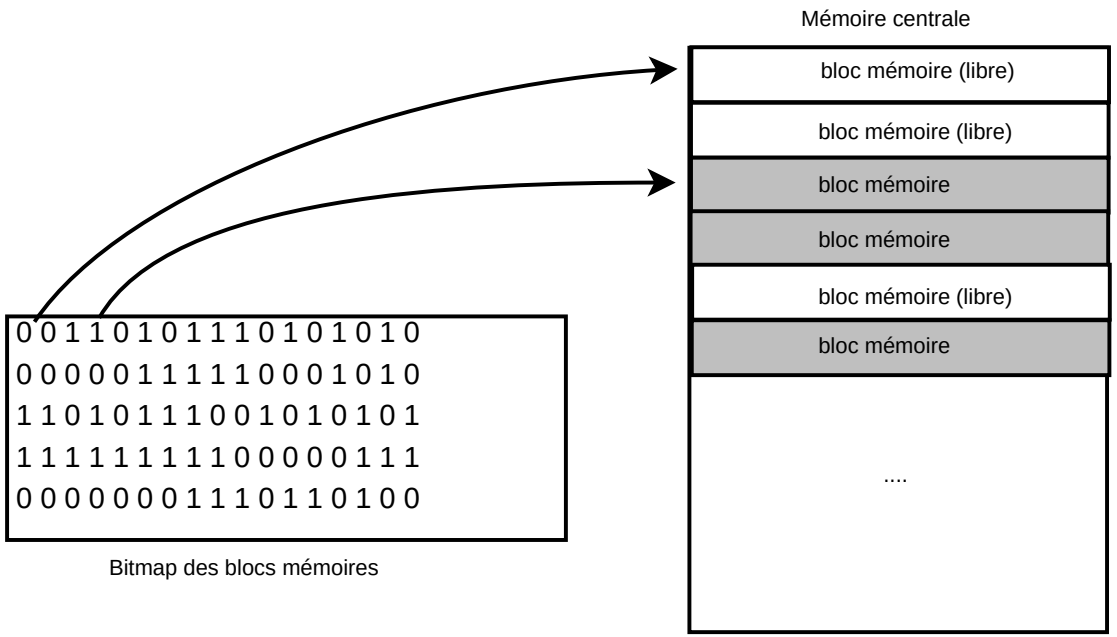
Swapping

Cette technique consiste à charger un processus en mémoire en recherchant l'espace libre qui lui est nécessaire dans la mémoire centrale. À la fin de l'exécution du processus, il libère la place mémoire. Le transfert d'un processus de la mémoire vers le disque peut aussi être nécessaire si le système a besoin de place pour un autre processus.

- Charger un processus du disque vers la mémoire
- **Recherche de l'espace libre**
- Transfert vers le disque si besoin

BitmapMC

Trois méthodes peuvent être utilisées pour gérer alors la mémoire (pour connaitre les espaces libres) : le bitmap, les listes chaînées, le *buddy* système.
Le principe consiste à découper la mémoire en bloc de plusieurs octets et de garder en mémoire une table avec un 0 pour dire que ce bloc est libre et un 1 pour occupé.



BitmapMC

Bien que le principe soit simple, il porte plusieurs inconvénients :

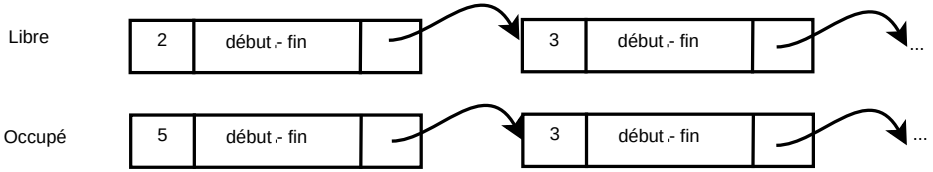
- la recherche de n blocs libres est coûteuse en temps.
- Pour accélérer la recherche et prendre moins de place en mémoire : définir un bloc sur plus d'octets mais pertes de place lors de l'allocation.

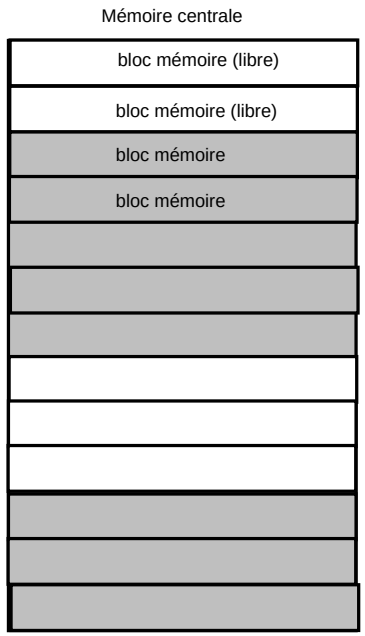
Exercice

Votre DD possède 10 blocs, nommés $D0, D1, \ldots, D9$. Votre MC possède 3 blocs nommés $R0, R1, R2$. Imaginons qu'un processus ait besoin dans l'ordre et séquentiellement des blocs $D0, D3, D8, D2$ puis $D0$.
Dessiner l'évolution de la bitmap, en supposant que la MC commence vide.

Listes chainées

Une autre méthode consiste à décrire la mémoire comme une liste chaînée d'objets alternant des plages de mémoire libre et de mémoire occupée avec l'information de la taille du segment. Il est apparu plus intéressant de gérer les plages libres et les plages occupées par deux listes chaînées distinctes.





Listes chainées

Recherche d'un plage pour un processus :

- *first fit* : le premier (emplacement) qui convient
- *best fit* : le meilleur ajustement
- *worst fit* : le pire ajustement ou le plus grand résidu

Par simulation, *first fit* meilleur des trois (le moins couteux).

Exercice

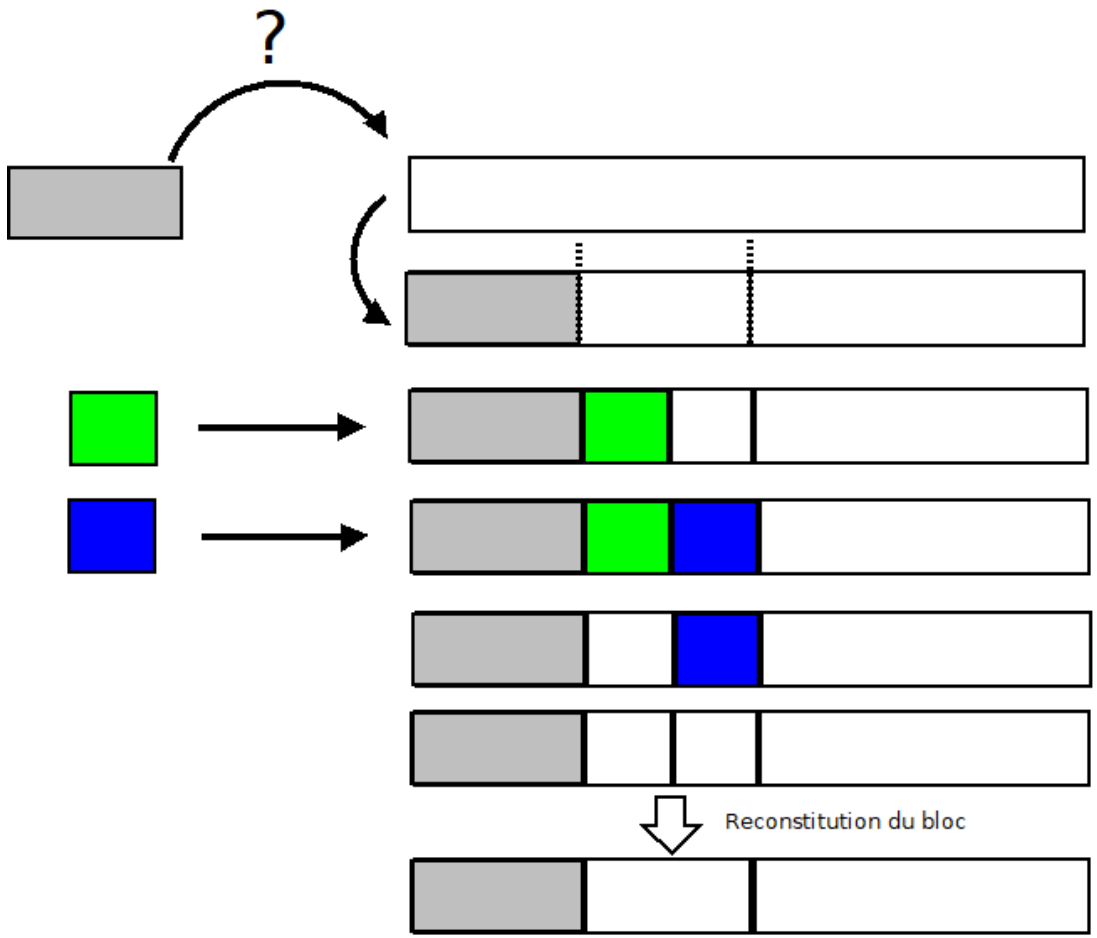
Votre DD possède 10 blocs, nommés $D0, D1, \ldots, D9$. Votre MC possède 3 blocs nommés $R0, R1, R2$. Imaginons qu'un processus ait besoin dans l'ordre et séquentiellement des blocs $D0, D3, D8, D2$ puis $D0$.
Dessiner l'évolution des listes chaînées first fit, en supposant que la MC commence vide.

Buddy : blocs compagnons

Le système repose sur le principe de blocs compagnons (*buddy*). La mémoire est constituée de blocs de taille d'une puissance de deux. Lors de l'allocation, le système recherche un bloc de la taille demandée T. S'il n'y a pas de bloc de cette taille le système divise en 2 un bloc de taille $2T$ (si ce dernier n'existe pas il cherche alors à diviser un bloc de taille $2 \times 2T$ pour obtenir ce bloc de $2T$ ...). Pour limiter la fragmentation, à la libération, le(s) bloc(s) de taille supérieur est reconstitué si le bloc compagnon à côté du bloc libéré est libre.

Buddy : blocs compagnons

Les blocs sont découpés en 2 si besoin (et remis ensemble à la libération)



Exercice

Votre DD possède 10 blocs, nommés $D0, D1, \ldots, D9$. Votre MC possède 3 blocs nommés $R0, R1, R2, R4$. Imaginons qu'un processus ait besoin dans l'ordre et séquentiellement des blocs $D0, D3, D8, D2$ puis $D0$.
Dessiner l'évolution de la MC en supposant qu'elle commence vide.

Récupération

Toutes ces techniques de va-et-vient implique l'apparition d'une fragmentation de la mémoire plus ou moins important pénalisant l'allocation pouvant entraîner une perte de place mémoire importante. Le principe est de compacter périodiquement la mémoire utilisée, c'est-à-dire déplacer les plages de mémoires libres pour les regrouper en une seule plage. Cependant cette technique est lourde à mettre en place et très coûteuse en temps.

- Quelque soit la technique : fragmentation
- Compacter régulièrement
- ... mais couteux en temps. ($\Rightarrow$*daemon*)