SR02: Gestion de la mémoire 15/05

Introduction

- Avec la multi-programmation, nous avons donc donc plusieurs processus qui accèdent à la RAM.
- La taille de la mémoire est restreinte, comment l'optimiser?
- Si un processus à une taille supérieure à la place en mémoire centrale, comment l'exécuter?

Rappel:
On a 2 catégories de mémoire, la mémoire centrale interne (RAM) et la mémoire de masse.

Les éléments de caractérisation d'une mémoire sont :

- Mode de localisation de l'info
- Volume
- Tps d'accès
- Tps de cycle
- Débit
- Volatilité => Si non alimenté électriquement, on perds les infos? => La RAM est volatile.

Mémoire cache

- plus rapide que la RAM, mais aussi plus cher.
- Débit mémoire centrale très lent par rapport au processur.
- La mémoire cache est **l'intermédiaire** entre le processeur et la mémoire centrale.
- Pour des informations rapides et peu gourmandes, le processeur les prendra dans le cache.
- **Problème de l'inconsistance des données**: si on change les données dans la mémoire RAM, il faut aussi les changer dans le cache!

Type d'accès à la mémoire

- Accès séquentiel => lent
- Accès direct ou aléatoire => chaque info a une adresse
- Accès semi-séquentiel

Multiprogrammation: les programmes d'application sont dans des zones de la mémoire dont la localisation n'est **PAS CONNUE** d'avance.
Système va fournir l'info qui permet a chaque programme de s'exécuter sans savoir a l'avance dans quelle zone mémoire il sera pdt l'exé.
Bonne gestion de la mémoire = **Augmenter le rendement global du système.**

Question que le système doit résoudre => **"A quel endroit dois-je stocker?"**

Espace d'adressage logique ou physique

L'UC manipule des **adresses logiques**. Les programmes ne connaissent que des adresses logiques ou virtuelles. L'espace d'adressage logique est générable par un programme.

Le système ne travaille qu'avec des adresses logiques.

Orga de la mémoire en monoprogrammation

Exécution *d'un programme à la fois*, avec partage de la mémoire entre le programme et le S.E.
Dès que l'utilisateur tape une cmd, le S.E copie le prog demandé depuis le disque vers la mémoire et l'exécute.

Gestion très simple de la mémoire, mais pas très performant

Gestion de la mémoire en Multiprogrammation

Plusieurs programmes en même temps en mémoire, 2 stratégies d'allocation mémoire:

- Allocation de partitions contiguës
- Allocation de partitions non contiguës

Partitions contiguës fixes

Table de description des partitions

- Mémoire divisée en *n* partitions de tailles fixes. Impossible de changer la taille. Partitionnement au démarrage du système. Le S.E maintient une table de description de partitions (Adresse, Taille, Etat (Libre, allouée)).

- Pour savoir le début et la fin des partitions, on définit 2 registres:

    - *registre de base avec la limite inférieure de la partition*
    - *registre de limite avec la limite supérieure de la partition*

Attention le processeur se déroutera si l'adresse est en dehors des limites
Attention Un processus étant de taille inférieure ou égale à la partition, il reste souvent de l'espace libre. *voir diapo 16 pour + d'explications sur les problèmes que l'on peut rencontrer*

Allocation de partitions fixes =>files d'attente séparée

- File d'attente pour chaque partition => Quand le process arrive il est placé dans la file d'attente des entrées de la plus petite des partitions assez large pour le contenir.
  Attention Il peut y avoir des inconvénients si une grande partition a une file vide alors qu'une petite partition a une file pleine. Ou si on alloue une partition trop grande à un petit process.

En général, on perd de la place dans chaque partition: **FRAGMENTATION INTERNE**

- Une autre solution, créer une **file unique** pour répondre au problèmes des files d'attente (files d'attente de partition de grande taille vide à l'inverse des partitions de petit volume pleines).

Allocation de partitions fixes =>file d'attente commune

voir diapo19

Partitions contiguës dynamiques

- Si un fragment reste, on peut refaire une partition.
- Gros avantage, chaque processus est alloué exactement la taille de mémoire requise.
- **Attention** Problème de la fragmentation externe. A cause du changement de partition, on peut changer les tailles

Stratégies d'allocation

- Stratégie du premier qui convient (First Fit): Partitions libres triée par ordre des adresses croissantes. Donc recherche commence par la partition libre de plus basse adresse, jusqu'a la rencontre de la première partition dont la taille est au moins égale à celle du process en attente.
- Stratégie du meilleur qui convient (Best Fit): Table des partitions triée par tailles croissantes. On donne la plus petite partition possible au processus en attente.
- Stratégie du pire qui convient (Worst Fit): On alloue à tout les processus la partition la plus grande.

Pour des exemples, voir la diapo 22/23/24

Problème de famine: 2 processus en cours d'éxecution sont dans une partition, empeche un troisième processus de rentrer car il est trop volumineux pour l'espace restant. Un quatrième processus de taille plus petite que le 3 rentre dans la partition => le 3 sera bloqué indéfiniment (pour peu qu'on ai invasion de processus + petits).

Problème de fragmentation interne Programme de taille M, partition de taille N, Si N>M, alors partie non occupée => fragmentation interne

Problème de fragmentation externe Si on stocke des processus dans l'operating system, mais qu'il reste des octets de mémoire libre. Ex => Si il reste 64 kO, mais qu'aucun processus a une taille <= a 64 kO. *voir diapo 25 pour explications détaillées Attention* Ces fragments sont externes aux partitions! Pas interne.

Partitions contiguës Siamoises (Buddy system)

- mémoire allouée en puissances de 2
- compromis entre partitions tailles fixes et tailles variables.
- Quand la mémoire doit être attribuée à un processus, ce dernier reçoit une unité de mémoire dont la taille est la plus petite puissance de 2 supérieure à la taille du processus..
- *voir fin diapo 26 pour compléter*

Ex: Au départ on a notre mémoire de taille 1M O. On divise successivement /2 jusqu'a remplir avec la plus petite puissance de 2 sup à la taille du process. Quand les programmes se terminent, on re-fusionne!

Ré-Allocation (translation d'adresse) et protection

- Ou commence exactement un programme dans la mémoire? Il faut aussi empêcher un processus d'accéder à l'espace d'adressage d'un autre processus.
- Quand un prog est lié, il faut savoir ou il démarre dans la mémoire. **Protection** Déconseillé d''autoriser des processus à lire ou) écrire dans la mémoire appartenant à d'autres utilisateurs (et au S.E) => problèmem de **protection**. **Registres de base et de limite**
- On compare les adresses avec la valeur du registre de limite pour etre sur qu'elles ne référenceront pas une adresse hors de la partition courante. *diapo 32*
- On protège aussi les registres de base et de limite pour empêcher les programmes utilisateur de les modifier. *voir figure diapo 33*

Technique du Va et Vient (swap)

Supposons qu'on a un processus de la même taille que la mémoire. Il ne peut pas y avoir d'autres processus actifs. Il faut donc conserver les process supplémentaires sur un disque, puis les charger pour qu'ils s'exécutent dynamiquement.

Fragmentation et compactage

Méthode: Compactage de la mémoire (regroupement des zones libres, couteux, car on change les adresses des partitions). Il faut donc un mécanisme de réallocation dynamique des processus.

voir diapo 39

Pagination: on découpe l'espace adressable(espace virtuel) en zones de taille fixe appelée **pages**. La mémoire réelle est aussi découpée en cases(frame), ayant la taille d'une page pour que chaque page puisse être implantée dans n'importe quelle case de mémorie réelle. On crée alors une *table des pages* qui contient:

- Indicateur de *présence* : case allouée à cette page.
- *Numéro de case* allouée à la page
- Indicateurs de *protection*: opérations autorisées sur cette page par le processus
- *Page modifiée*
- *Page accédée voir diapo p 41*

Pagination à un niveau (une seule table des pages)

- Adresse divisée en 2 : num de page *p* + déplacement à l'intérieur de la page *d*
- Taille de la page(et donc case) est une puissance de 2 *diapo42*

Traduction adresse virtuelle en adresse physique.

On part de (p,d) => on traduit en (f,d). On va trouver dans la tables des pages la référence p. On accède donc à la bonne page, qui contient le numéro de la case f. On va pouvoir accéder à la bonne case mémoire. Le décalage ne change pas.

Le problème qui se pose avec la Pagination à un niveau => Grosse taille de la table des pages. Il faut donc l'alléger =>Pagination à 2 niveaux.

Pagination à 2 niveaux

Adresse virtuelle cette fois contient:

- numéro de page maître
- numéro de page secondaire

- décalage (*d*)

Segmentation

- Analogue à la pagination, mais taille d'un segment est variable.
- Peut refléter une vision logique du programme (segmentation en code /programme etc..) *diapo 53*

```
CREATE TABLE
int c = 2;
```