Mémoire vive : pagination et segmentation

Licence MIAGE — Université Lille 1
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Semestre 6 — 2012-2013

Organisation de la mémoire physique

Sur une architecture 32 bits, la mémoire physique peut être considérée comme une suite d'octets. À chaque octet est associé un numéro de 0 à $2^{32} - 1$.

Remarquez que l'on peut adresser environ 4 gigaoctets de mémoire ce qui est plus que n'en possèdent la plupart des machines actuelles. Une partie de cette mémoire est virtuelle (cf. la suite).

La gestion de la mémoire vive présente des analogies avec la gestion de la mémoire persistante.

Répartition de la mémoire entre système et processus

La mémoire principale se divise en deux parties :

- l'une est destinée au système d'exploitation et contient
  - une pile d'interruption système,
  - la table des processus,
  - le code du système (noyau, etc.),

  Cette partie est toujours résidente en mémoire ;
- l'autre est destinée aux processus en cours d'exécution

Image d'un processus en mémoire
Pile d'exécution (pass. param.)
tas (zone malloc)
données non initialisées (à 0)
données initialisées
code utilisateur

un autre processus
Pile d'exécution (pass. param.)
tas (zone malloc)
données non initialisées (à 0)
données initialisées
code utilisateur

Adressage monoprogrammation et description des contraintes

Dans ce modèle, la mémoire du processus n'est pas fragmentée.

Si la mémoire ne contient qu'un (ou un nombre restreint) de processus, ils seront le plus souvent dans l'attente d'une entrée-sortie. Donc le processeur sera inactif.

Ce problème peut se traiter en installant un système de file d'attente et en permutant les processus actifs (cf. cours sur l'ordonnancement).

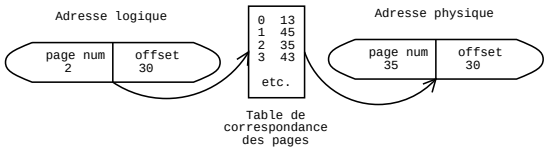
Ceci implique de partitionner la mémoire (car sinon, il faudrait charger les processus en mémoire depuis le disque).

Même en faisant des partitions de tailles variables, on tombe sur les problèmes suivants :

- comment s'arranger pour que le processus reçoive exactement la taille nécessaire sans gaspillage ?
- comment un programme peut-il adresser de la mémoire dans ce cadre ?

Pagination : une abstraction au niveau matériel

La mémoire est partitionnée en *pages* de taille fixe et petite.
Chaque page possède un numéro physique et comme pour un FS, l'OS maintient une liste des pages libres.
Un processus est divisé en pages ayant un numéro logique et une table des pages le constituant est maintenu par l'OS.
l'adresse d'un octet est défini par sa page et son offset.
On peut ainsi convertir une adresse relative au processus en une adresse absolue pour l'OS :



Pagination : conséquence sur la programmation

Cette abstraction est invisible à l'utilisateur mais elle a des conséquences sur la gestion d'un vaste espace mémoire.

Supposons que la taille d'une page soit de 128 sizeof(int) et qu'un tableau soit stocké ligne par ligne :

tab[0][0]	tab[0][1]	...	tab[0][127]	1 page
⋮	⋮	...	⋮	d'autres
tab[127][0]	tab[127][1]	...	tab[127][127]	la dernière

Les deux codes suivants ne sont pas équivalents en termes de gestion mémoire (le premier peut nécessiter des swaps).

```
int tab[128][128] ;
for(int j = 0 ; j<128 ; j++)
    for(int i = 0 ; i<128 ; i++)
        tab[i][j] = 0 ;
    /* parcours les pages */
```

```
int tab[128][128] ;
for(int i = 0 ; i<128 ; i++)
  for(int j = 0 ; j<128 ; j++)
    tab[i][j] = 0 ;
  /* reste dans une page */
```

Segmentation : une abstraction sémantique

La segmentation permet au programmeur de voir un processus en mémoire sous la forme de plusieurs espaces d'adressages : les segments.

1. chaques composantes du processus (pile, code, tas, données, etc.) peut avoir son propre segment ;
2. on peut mettre des droits sur l'accès aux segments ;
3. la taille des segments est variable sous l'action de l'OS (tas, pile) ;
4. un segment peut être partagé par plusieurs processus.

Voir codage du segment ci-dessous.

Mémoire vive : pagination et segmentation
Pagination
Segmentation
Mémoire paginée segmentée
Mémoire virtuelle

Mémoire paginée segmentée

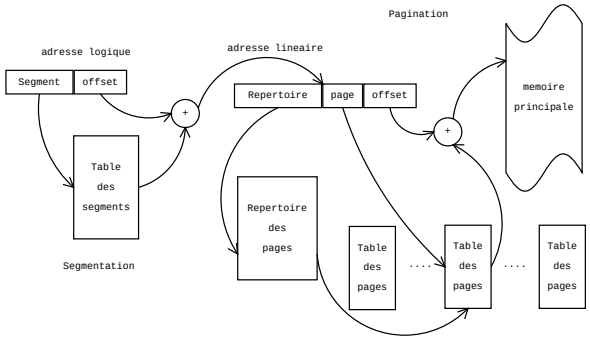
Pour accéder à un octet dans un segment, on utilise une *adresse logique*. Cette adresse est composée du numéro du segment et d'un offset.

Citons comme exemple d'adresse logique, l'adresse de la prochaîne instructions à exécuter dans un processus :

- le registre EIP (Instruction Pointer) contient l'offset de la prochaîne instruction à exécuter ;
- Le registre CS (Code Segment) contient le numéro du segment mémoire dans lequel sont stocké les instructions assembleur du code à exécuter.

Un segment s'étend sur plusieurs pages et le matériel propose un mécanisme de traduction d'adresse relative en adresse physique.

Mémoire vive : pagination et segmentation
Pagination
Segmentation
Mémoire paginée segmentée
Mémoire virtuelle



Mémoire vive : pagination et segmentation
Pagination
Segmentation
Mémoire paginée segmentée
Mémoire virtuelle

Mémoire virtuelle

On a souvent besoin de plus de mémoire vive que n'en disposent nos machines. L'OS met à disposition de la mémoire virtuelle par le biais d'une partie du FS qui est utilisée pour stocker des données en attente de réintégrer la mémoire (le FS dévolu à cette tâche est le swap).

Cette mémoire virtuelle permet aussi un mécanisme de protection : chaque processus à son propre espace mémoire qui peut être protégé par des droits.

Le principal défaut de cette astuce est que l'accès à un disque est lent au point que l'on peut avoir un *écroulement* : un processus s'écroule lorsqu'il passe plus de temps à paginer qu'à s'exécuter.

Mémoire vive : pagination et segmentation
Pagination
Segmentation
Mémoire paginée segmentée
Mémoire virtuelle

Pagination à la demande

Dans ce mode, la page d'un processus n'est chargée en mémoire que lorsque c'est nécessaire.
FS présente des fichiers liés à la mémoire

- **/dev/kmem** mémoire virtuelle vue comme un périphérique
- **/dev/mem** mémoire physique vue comme un périphérique
- **/dev/drun** mémoire de pagination

- **swap** : une partie du disque contenant une mémoire virtuelle

et une commande externe du shell vmstat donnant des statistiques sur — entre autre — la mémoire virtuelle :

```
procs ---------memory--------- --swap-- ---io-- --system-- --cpu--
 r  b   swpd   free   buff  cache   si so   bi   bo   in   cs us sy id wa
 0  0      0 60456 31576 224656    0  0    6    3  169  250  2  0 98  0
 0  0      0 60444 31576 224656    0  0    0    0  143  135  0  0 100 0
 0  0      0 60428 31592 224656    0  0    0   40  159  235  0  0 100 0
```