Systèmes d'Exploitation

Gestion de la mémoire

Didier Verna

didier@lrde.epita.fr
http://www.lrde.epita.fr/~didier

Table des matières

Généralités

Allocation contigüe

Monoprogrammation

Multiprogrammation

Pagination

Segmentation

1 Généralités

2 Allocation contigüe
- Monoprogrammation
- Multiprogrammation

3 Pagination

4 Segmentation

Niveaux d'adressage

- **Adresse symbolique** : manipulée au niveau du programme
- **Adresse logique** : générée par le CPU
- **Adresse physique** : emplacement mémoire réel
- **MMU** (Memory Management Unit) : dispositif de liaison d'adresse

Liaison d'adresse

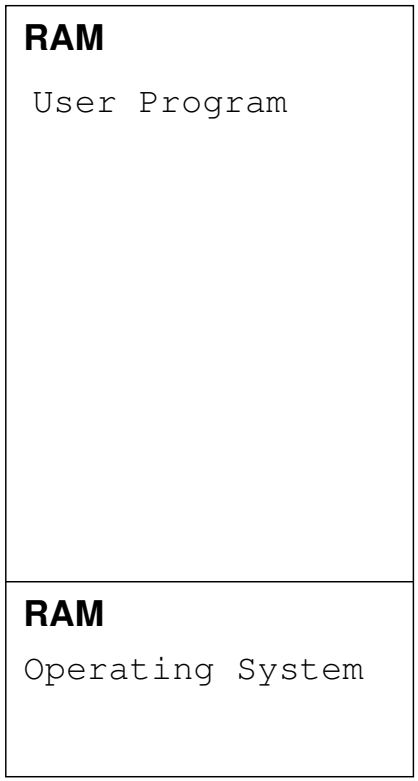
- **Problème**
  - Adresses symboliques $\Longrightarrow$ adresses physiques
  - À quel moment fabriquer une adresse physique ?
- **Solutions**
  - **Compilation :** « Code absolu »
    Adresse de chargement connue.
    Exemple : `command.com`.
  - **Chargement :** « Code translatable »
    Adressage relatif.
  - **Exécution :** « Code dynamique »
    L'emplacement du programme peut varier dans le temps. Nécessite un matériel spécial.

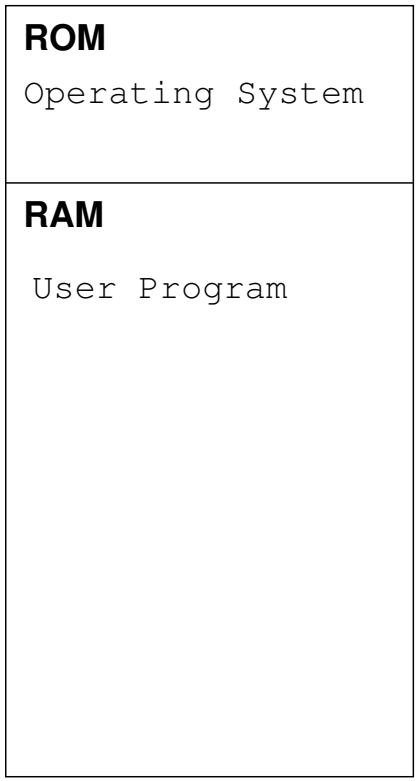
Techniques de haut niveau

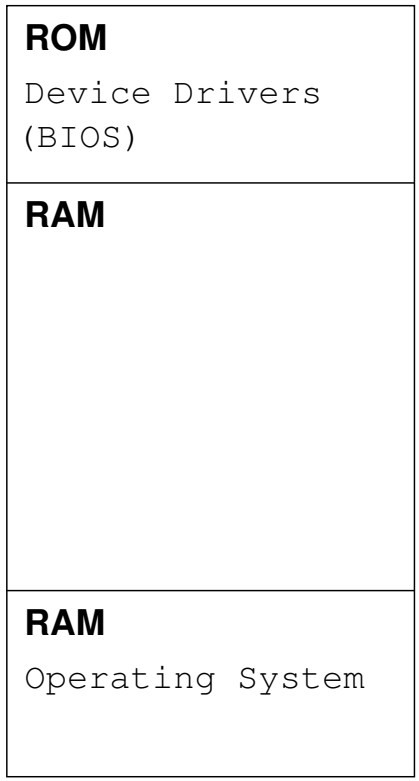
- **Recouvrements :** (overlays) charger un processus par tranches de code indépendantes. Niveau utilisateur.
- **Chargement dynamique :** charger le code nécessaire uniquement quand on en a besoin. Niveau utilisateur.
- **Édition de liens dynamique :** bibliothèques partagées.
- **Swapping** : déplacement de processus entre mémoire et mémoire auxiliaire (disques). Coûteux en commutation.

Monoprogrammation

Mainframes, Palm et systèmes embarqués, MS-DOS





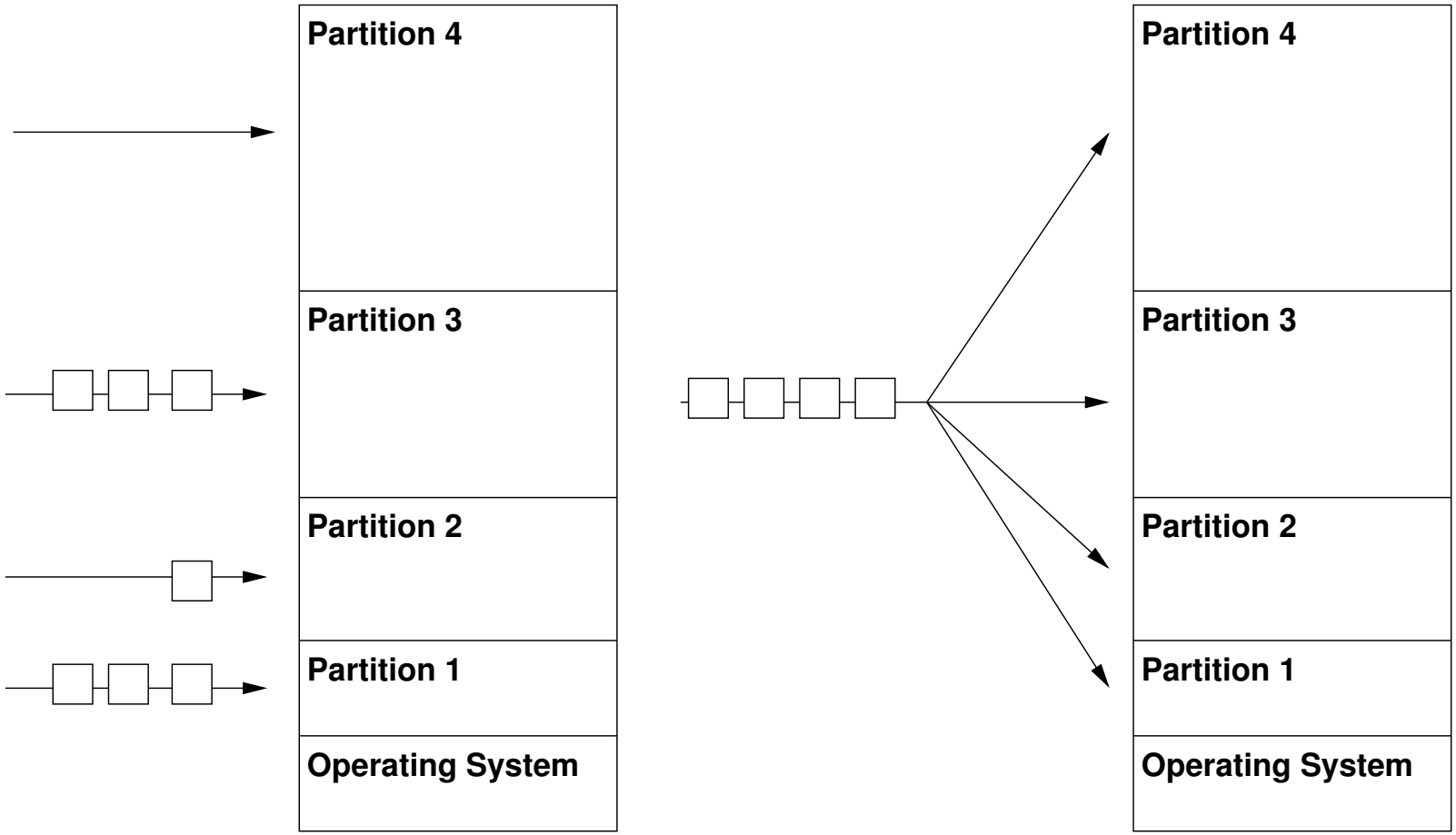


0xFF..

0x00

Multiprogrammation & partitions fixes

« OS/MFT » sur OS/360

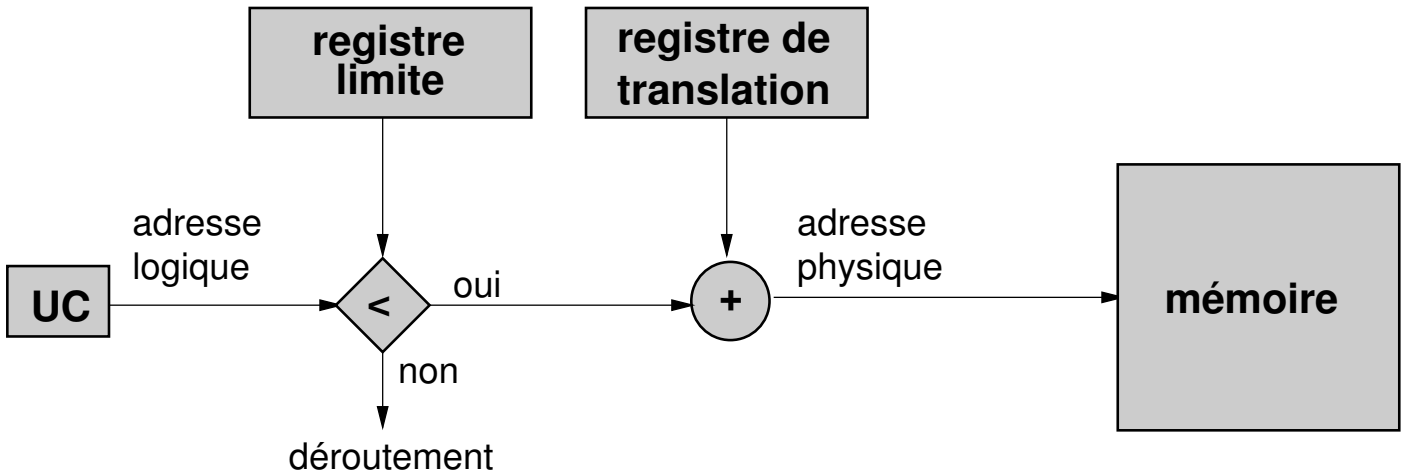


Localisation et protection

- **Modification du code :** (OS/MFT, édition de liens) nécessite de connaître les mots à modifier. Ne résoud pas le problème de protection.

$\Longrightarrow$ Blocs de 2kB protégés par un code de 4 bits (PSW).

- **Par MMU :** (CDC 6600, Intel 8088)



$\Longrightarrow$ Commutation de contexte plus coûteuse.

Multiprogrammation & partitions dynamiques

- **Principe**
  - Une partition mémoire par processus
  - Allocation / libération de la mémoire en fonction de l'ordonnancement
  - Idem pour le swapping
- **Implémentations**
  - **Bitmap :** 1 bit par zone mémoire (kB) indiquant si la zone est libre ou occupée. Attention à la taille des zones
    - Avantages : simple, taille du bitmap connue
    - Inconvénients : lent
  - **Listes chaînées :** zone libre / occupée, adresse de début et longueur. Tri par adresse, taille, listes distinctes de processus, trous. Listes doublement chaînées.

Politiques d'allocations

- **First-Fit :** premier trou suffisant. Rapide.
- **Next-Fit :** idem, mais recherche à partir de l'emplacement précédent. Un peu moins bon.
- **Best-Fit :** trou le plus petit possible. Moins performant.
- **Worst-Fit :** trou le plus grand. Bof.
- **Quick-Fit :** maintient de listes par tailles fréquentes. Rapide pour la recherche, lent pour la désallocation.

Remarque : attention aux politiques de tri des listes.

Fragmentation

Apparition de zones inoccupées dans la mémoire

- **Types**
  - **Fragmentation externe** : espace suffisant pour l'allocation d'un nouveau processus, mais non contigu
  - **Fragmentation interne** : allocation volontaire de zones inoccupées pour diminuer le travail de gestion de la mémoire
- **Compactage**
  - Défragmentation de la mémoire par translation des processus (code dynamique)
  - Stratégies de compactage difficiles à trouver, lenteur

Pagination

Allocation de zones de mémoire non contigües pour un même processus

- **Cadres de page** : mémoire physique découpée en zones de taille fixe
- **Adresse logique** : numéro de page + déplacement dans la page
- **Table de pages** : liaison entre numéro de page et cadre de page (une table par processus)
- **Taille des pages** : puissance de 2

MMU pour la pagination

Systèmes d'Exploitation

Didier Verna
EPITA

Généralités

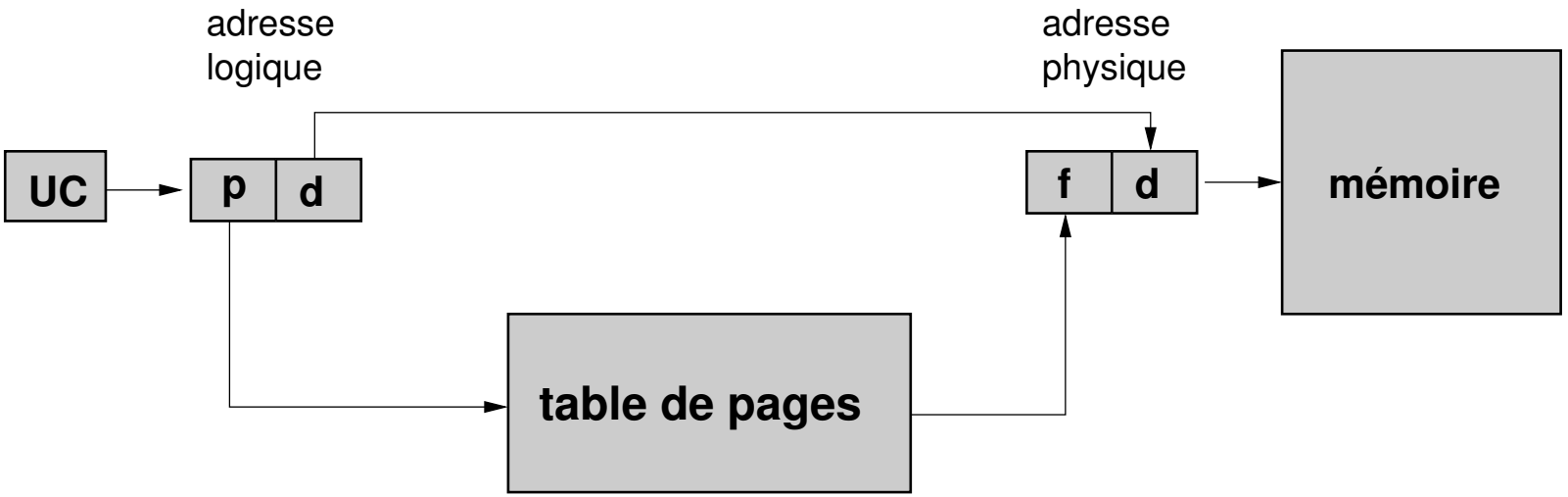
Allocation contigüe

Monoprogrammation

Multiprogrammation

Pagination

Segmentation



Caractéristiques

- Pas de fragmentation externe, mais fragmentation interne
- Petites pages $\Longrightarrow$ moins de fragmentation
- Grandes pages $\Longrightarrow$ commutation moins coûteuse
- Impossible par définition d'accéder à une page interdite
- Mécanismes de contrôle : pages (in)valides, en lecture, écriture, exécution *etc.*
- Implémentation du partage de la mémoire plus facile
- Taille moyenne des pages : 2 – 4 Ko

Support matériel
La pagination doit être rapide (ns)

- **Registres dédiés :** possible seulement pour des petites tables (ex. DEC PDP-11 : adresse 16 bits, pages de 8ko $\Longrightarrow$ 8 entrées).
- **Mémoire principale :** un unique registre « PTBR » (Page Table Base Register). Surcharge de commutation de contexte faible, mais temps d'accès double.
- **Registres associatifs :** « TLB » (Translation Look-Aside Buffers). Fonctionne comme un cache de la table de pages (ex. Motorola 68030 : TLB à 22 entrées, i486 : 32 entrées, taux de présence de 98%).
  - **Gestion logicielle des TLB :** par le système au lieu du MMU (Architectures RISC : Sparc, MIPS, Alpha). Acceptable pour des TLB assez grands (64 entrées). Simplifie le MMU.

Pagination à plusieurs niveaux

Exemple (courant) : adressage sur 32 bits + pages de 4 Ko. Une table de page contient alors 1 million d'entrées, soit 4 Mo de mémoire ! !

- **À éviter**
  - allouer autant d'espace en contigu
  - charger l'intégralité des tables de page en mémoire
- **Pagination de la table de page**
  - **SPARC** : pagination 32 bits à trois niveaux
  - **Motorola 68030** : pagination à quatre niveaux
- **Problème** : trois ou quatre indirections à chaque fois que l'on doit récupérer un octet.

Table de pages inversée

Autre solution au problème de la taille des tables de pages

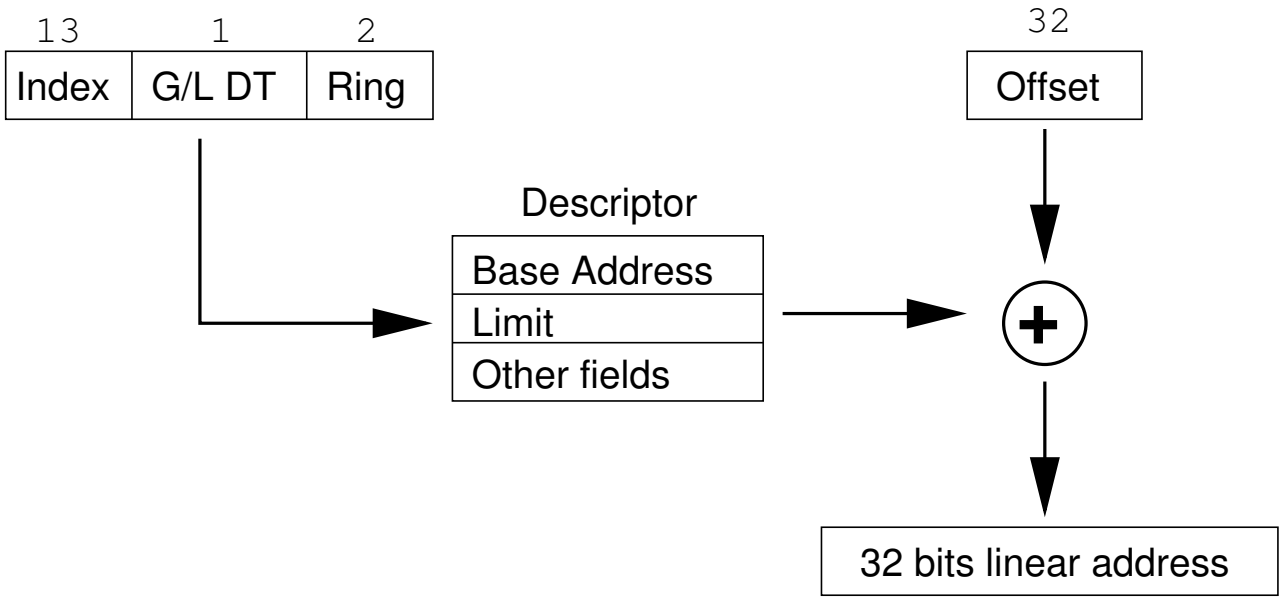
- **Principe**
  - Une unique table de pages pour tout le système, indexée par les cadres de page
  - Chaque entrée contient un numéro de processus et un numéro de page virtuelle
- **Problèmes**
  - Liaison d'adresse plus compliquée (parcours de la table). $\Longrightarrow$ TLB, tables de hash par adresse virtuelle.
  - Partage de pages plus difficile à implémenter au niveau système.
- **Utilisation actuelle** : IBM, HP, de plus en plus d'architectures 64 bits.

Segmentation

- **Segmentation *vs.* Pagination**
  - **Pagination :** obtenir un espace d'adresse *linéaire* aussi grand que souhaité. $\Longrightarrow$ Dimension 1
  - **Segmentation :** obtenir *plusieurs* espaces d'adressage distincts (segments). $\Longrightarrow$ Dimension 2 (ex. compilateur + pile, source, table de symboles, AST*etc.*)
- **Principe**
  - Segment = unité *logique* (niveau utilisateur)
  - Taille variable, mais indépendante des autres segments
  - Facilité de partage (ex. bibliothèques partagées)
  - Facilité de protection (ex. lecture, écriture. exécution)
- **Implémentation**
  - **Table de segments :** idée analogue à la pagination
  - **Adresse logique** : nom du segment + déplacement
  - Segmentation et pagination combinée

Segmentation et Pagination
Exemple : Intel Pentium

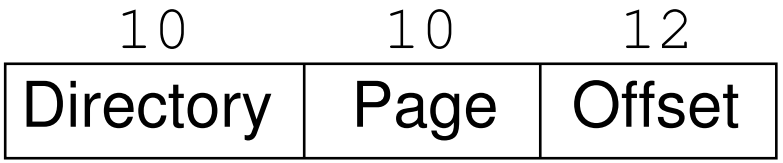
- 16K segments de 4 Go par processus, dont :
  - 8K segments privés (une table de descripteurs locale « LDT » par processus)
  - 8K segments partageables (une table de descripteurs globale « GDT)
- Sélecteur de segment (16 bits) + offset (32 bits) $\Longrightarrow$ « Adresse linéaire » (32 bits)



Intel Pentium (suite)

Pagination :

- Pagination à 2 niveaux.
- Pages de 4Ko.



Protection :

- 4 anneaux différents.
- Progression en profondeur par portes / guichets.