FACULTÉ
DES SCIENCES
Unité de formation
et de recherche

UNIVERSITE
BRETAGNE
LOIRE

Système

Fabien GARREAU
sur la base du cours d'André Rossi

Université d'Angers

`fabien.garreau@univ-angers.fr`

L2 MPCIE, 2018–2019

Gestion de la mémoire centrale

Quand un programme est compilé, les adresses des données sont soit :

- Des adresses physiques : on sait exactement où elles se trouvent en mémoire centrale. Mais quand le processus est sauvé sur disque puis restauré en mémoire, il faut impérativement le réimplanter au même endroit en mémoire.
- Des adresses logiques : connaissant l'adresse de base du programme, l'adresse physique des structures de données s'obtient en additionnant l'adresse logique à l'adresse de base du programme. Le **swapping**, qui consiste à sauver le contexte d'un processus sur disque pour le restaurer plus tard, est facilité.

La multiprogrammation divise la mémoire centrale en partitions avec :

- MFT (Multiprogramming with a Fixed number of Tasks), toutes les partitions sont identiques (système hérité d'OS/360).
- MVT (Multiprogramming with a Variable number of Tasks), la taille et le nombre de partitions sont variables

Gestion de la mémoire centrale

Comme pour le disque, la mémoire centrale peut être fragmentée.

- La fragmentation interne se manifeste lorsque la partition est trop grande pour le processus qui l'occupe
- La fragmentation externe apparaît quand la partition disponible est trop petite pour recevoir un quelconque processus en attente.

Partage de la mémoire dans le cas de MVT
On dispose comme pour le disque de la liste des zones disponibles de la mémoire centrale. Deux stratégies :

1. Quand un processus arrive et demande de la mémoire, on lui en donne si on en trouve, sinon on le met en attente.
2. Quand un processus se termine, il libère sa place, on cherche dans la liste d'attente un processus implantable en mémoire.

Gestion de la mémoire centrale

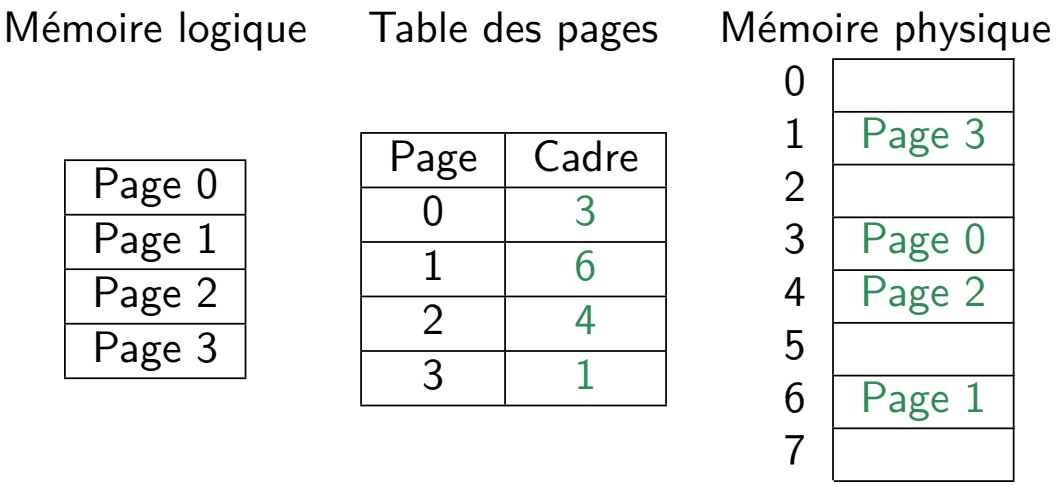
Il y a moins de fragmentation interne avec MVT qu'avec MFT, mais la fragmentation externe existe encore.

Deux options pour y remédier :

1. Le compactage (déplacer les processus en mémoire), mais cher et parfois impossible (processus non translatable)
2. La mémoire virtuelle : diviser un processus en plusieurs morceaux et les placer tour à tour en mémoire. Non seulement il n'y a plus de fragmentation, mais surtout on peut exécuter des programmes qui autrement ne tiendraient pas en mémoire centrale.

Pagination de la mémoire

On divise la mémoire physique en blocs de même taille, chaque bloc est un **cadre**, et la mémoire virtuelle en **pages**. Les cadres et les pages ont tous la même taille.
La **table des pages** a pour entrée les pages virtuelles, et indique dans quel cadre la page virtuelle est chargée.
Exemple : un cadre = 50 ko, donc un processus de 160 ko occupe 4 pages



Plusieurs processus peuvent ainsi partager des pages.

Pagination et segmentation de la mémoire

Il est naturel de découper un processus en segments de tailles différentes afin de distinguer le segment du code, celui des données, la pile, etc.
Chaque segment a un nom (souvent un numéro) et une longueur. Une adresse logique est un couple (numéro du segment, déplacement dans le segment).
En pratique, on peut combiner segmentation et pagination de la mémoire.
Comme il est rare qu'on ait besoin d'accéder à toutes les pages d'un processus en même temps, la **pagination à la demande** consiste à ne charger en mémoire que les pages dont on a besoin.
On observe un **défaut de page** lorsqu'on a besoin d'accéder à une page non chargée en mémoire. Dans ce cas :

1. On trouve un cadre libre
2. On localise la page sur le disque, et on la charge dans le cadre
3. On met à jour la table des pages
4. On reprend le processus à l'endroit du défaut de page

Pagination à la demande : optimalité et FIFO

Que faire si on ne peut trouver aucun cadre libre quand un défaut de page se produit ? Il faut libérer un cadre, et donc choisir une page victime (elle sera copiée sur le disque dur). Il convient de minimiser le nombre de défauts de page : il est souhaitable de maintenir en mémoire les pages les plus souvent utilisées. À cette fin, on présente quatre algorithmes :

- Algorithme optimal : la page victime est celle qui sera utilisée le plus tard (ou ne sera plus jamais utilisée). Mais en pratique il est difficile de prévoir les futurs besoins. Cet algorithme est utilisé sur un scénario pour évaluer la performance des autres algorithmes.
- FIFO : la victime est la page la plus ancienne. Cette stratégie est souvent trop simpliste : une page peut être ancienne mais très souvent utilisée.

Pagination à la demande : seconde chance et LRU

- Algorithme de la seconde chance. Utilisation d'un bit de référence R pour toutes les pages. R passe à 1 chaque fois que la page est référencée (=lue ou écrite). On applique FIFO, si le R de la page la plus ancienne est nul, c'est la victime. Sinon, son R passe à zéro et sa date de chargement = date courante. On recherche donc la page la plus âgée pour laquelle $R = 0$
  Remarque : si toutes les pages ont été référencées, comportement identique à FIFO.
- LRU (Least Recently Used). On calcule un entier codé en binaire d'autant plus grand que la page a été référencée récemment. Pour cela on construit M, une matrice $n \times n$ où n est le nombre de cadres. Initialement $M = 0$. Chaque fois que la page dont le cadre est i est référencée, la ligne i de M est remplie de 1, puis la colonne i de M est remplie de 0. Quand un défaut de page apparaît, la page contenue dans le cadre associé au plus petit entier codé en binaire calculé sur les lignes de M est la victime.

Pagination à la demande : illustration de LRU

Exemple : On a 3 cadres et on appelle les 5 pages $(P_2, P_3, P_7, P_3, P_4)$.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \quad \begin{matrix} - \\ - \\ - \end{matrix}$$

Situation initiale

$$\begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & 0 & 0 \\ \mathbf{0} & 0 & 0 \end{bmatrix} \quad \begin{matrix} \mathbf{3} \\ 0 \\ 0 \end{matrix} \quad \begin{matrix} P_2 \\ - \\ - \end{matrix}$$

Demande de P_2

$$\begin{bmatrix} 0 & \mathbf{0} & 1 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & \mathbf{0} & 0 \end{bmatrix} \quad \begin{matrix} 1 \\ \mathbf{5} \\ 0 \end{matrix} \quad \begin{matrix} P_2 \\ P_3 \\ - \end{matrix}$$

Demande de P_3

$$\begin{bmatrix} 0 & 0 & \mathbf{0} \\ 1 & 0 & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \end{bmatrix} \quad \begin{matrix} 0 \\ 4 \\ \mathbf{6} \end{matrix} \quad \begin{matrix} P_2 \\ P_3 \\ P_7 \end{matrix}$$

Demande de P_7

$$\begin{bmatrix} 0 & \mathbf{0} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 1 & \mathbf{0} & 0 \end{bmatrix} \quad \begin{matrix} 0 \\ \mathbf{5} \\ 4 \end{matrix} \quad \begin{matrix} P_2 \\ P_3 \\ P_7 \end{matrix}$$

Demande de P_3

$$\begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & 0 & 1 \\ \mathbf{0} & 0 & 0 \end{bmatrix} \quad \begin{matrix} \mathbf{3} \\ 1 \\ 0 \end{matrix} \quad \begin{matrix} P_4 \\ P_3 \\ P_7 \end{matrix}$$

Demande de P_4