Systèmes d'Exploitation

Mémoire Virtuelle

Didier Verna

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

Table des matières

Mémoire virtuelle

Inutile en général de faire figurer un processus en entier dans la mémoire

- Plus de limitation par la taille de la mémoire physique
- Plus de processus en même temps dans la mémoire
- Moins besoin de swapping, donc meilleure exécution

$\Longrightarrow$ Mémoire **virtuelle** aussi grande que souhaité

- Pagination à la demande
- Segmentation à la demande (ex. IBM OS/2)
  Plus difficile car segments de longueur variable

Pagination à la demande

- **Principe**
  - N'avoir en mémoire que les pages vraiment utilisées par les processus
  - Technique analogue au swapping, mais par page et non par processus
- **Matériel requis** idem que sans mémoire virtuelle (table de pages et mémoire auxiliaire). Utilisation du bit *valid / invalid* pour signaler si la page est présente ou non dans la mémoire.
- **Vocabulaire**
  - **Pagination à la demande pure** : ne jamais charger une page avant qu'elle soit requise
  - **Défaut de page** : requête d'accès à une page non chargée

Problèmes de performance

- Temps d'accès mémoire : $\mu = 10 - 200ns$
- Temps de traitement d'un défaut de page : $\delta = 25ms$
- Probabilité d'un défaut de page : p

$\Longrightarrow$ Temps réel d'accès mémoire : $\tau = (1-p)\mu + p\delta$

Pour une dégradation de moins de 10%, $p < 4.10^{-7}$, soit moins de 1 / 2 500 000 accès mémoire ! !

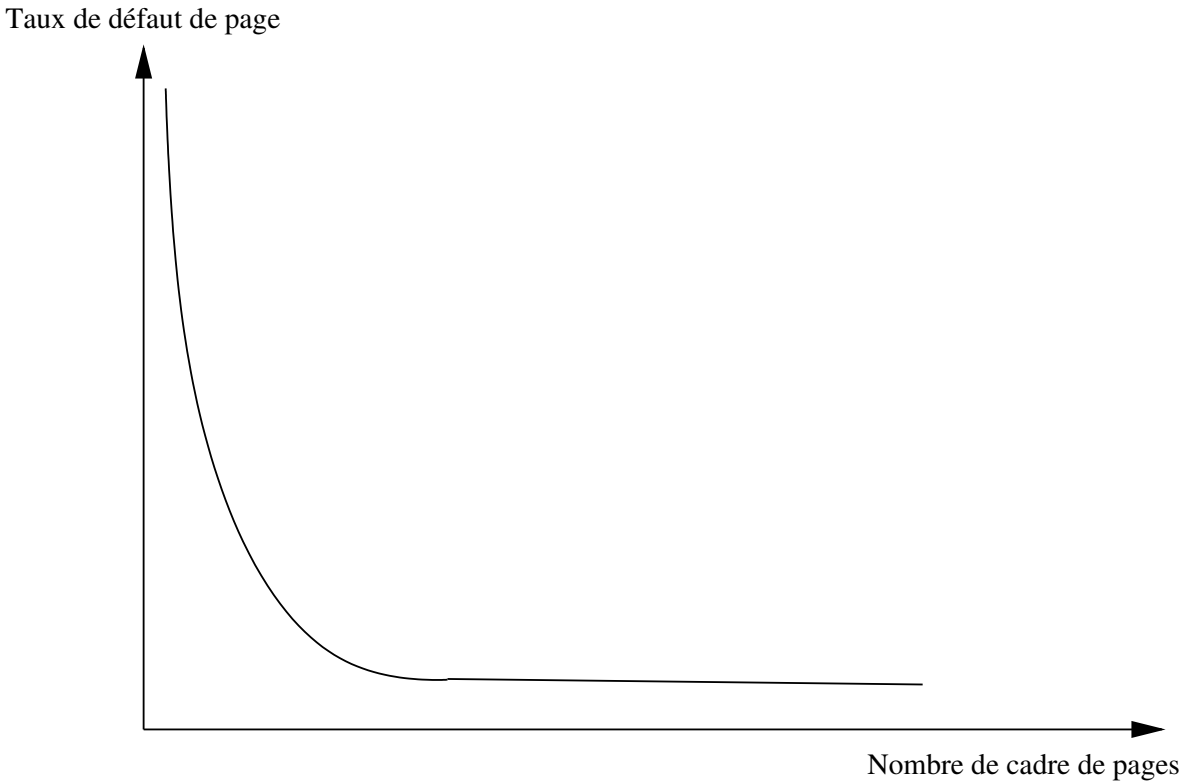
Remplacement de page

Que faire quand une page est manquante et que la mémoire est pleine ?

- Terminer le processus
- Swapper le processus
- Remplacer une page (inutilisée) par la page requise
  - Deux transferts de pages au lieu d'un en cas de page modifiée. Inacceptable en général.
  - Importance du choix des algorithmes d'allocation et de remplacement de pages.
- **Remarque :** Problématique très fréquente (caches, serveurs web *etc.*)

Algorithmes

- Minimiser le taux de défauts de page
- Modélisation par **chaîne de référence** : séquence d'adresses de pages générée par un processus



Algorithme Optimal

Remplacer la page qui mettra le plus de temps à être réutilisée

- Algorithme optimal pour un nombre donné de cadres de page
- Difficile (impossible ?) à implémenter : nécessite une connaissance du futur
- Particularité : pour une chaîne de référence S, le nombre de défauts de page est le même que pour S^{-1}

Remarque : algorithme utilisable en deux passes (enregistrement puis exécution). Utile pour comparer des algorithmes réels.

Algorithme FIFO

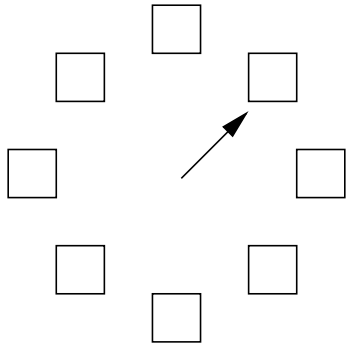
Remplacer la page la plus vieille

- Simple, mais pas très bon
- **Anomalie de Belady** : (1969) augmenter le nombre de cadres de page n'améliore pas forcément les performances du système

Algorithme de la seconde chance

Un bit de « référence » disponible

- **Description**
  - Algorithme FIFO avec examen du bit de référence
  - 0 $\Longrightarrow$ remplacer
  - 1 $\Longrightarrow$ donner une deuxième chance à la page (retour en queue de file et référence remise à zéro)
- **Remarque :** si tous les bits de référence sont à 1, dégénérescence en algorithme FIFO
- **Version « Clock »**
  - Implémentation alternative
  - Liste circulaire de cadres de page



Algorithme NRU

Un bit de référence et un bit de modification

- Le système réinitialise périodiquement le bit de référence
- (0, 0) : ni utilisée ni modifiée. Bon choix.
- (0, 1) : pas utilisée récemment mais modifiée. Nécessite une sauvegarde.
- (1, 0) : utilisée mais non modifiée. Probablement bientôt réutilisée, mais ne nécessite pas de sauvegarde.
- (1, 1) : utilisée et modifiée. Le pire des choix.

- Algorithme utilisé dans Mac OS (< X)
- Choix entre (2) et (3) délicat

Algorithme LRU

Remplacer la page qui n'a pas été utilisée depuis le plus longtemps

- **Description**
  - Algorithme optimal parmi ceux qui inspectent le passé plutôt que le futur
  - Même particularité que pour l'algorithme optimal
- **Algorithmes de pile :** (non victimes de l'anomalie de Belady) : si P_n est l'ensemble des pages présentes pour n cadres de page, alors $P_n \subset P_{n+1}$.

Implémentation du LRU

- **Implémentation par pile** : liste double avec pointeurs de tête et de queue. 6 pointeurs à mettre à jour maximum. Manipulation de la liste à chaque accès mémoire.
- **Implémentation par compteur :** (avec l'aide du matériel) compteur horloge associé à chaque entrée dans la table de pages. Recherche dans la table, mise à jour à la commutation, gestion des débordements d'horloge etc.
- **Implémentation matricielle :** matrice $N.N$ pour N cadres de page. Mise à 1 par ligne et à 0 par colonne. Ordonnancement des pages par inspection des lignes.

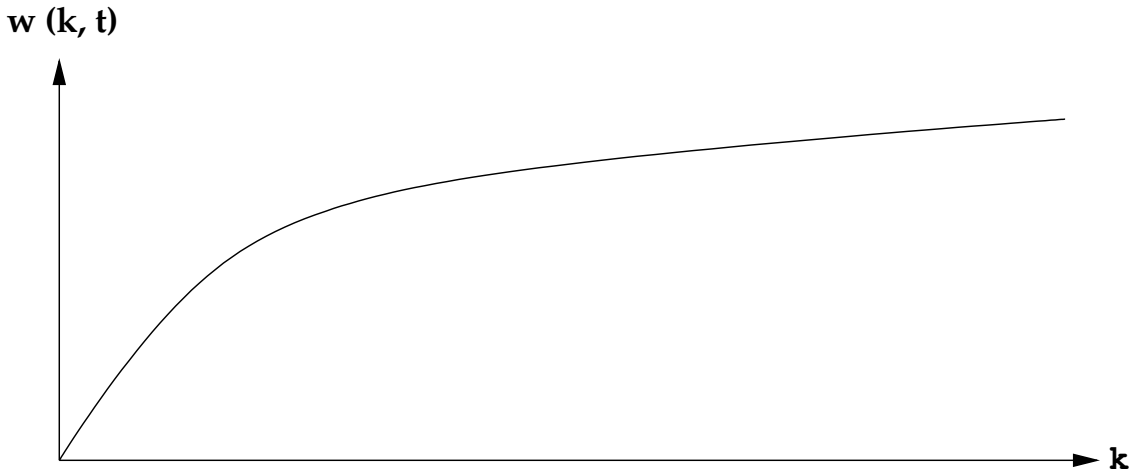
Approximation logicielle du LRU : NFU

Peu de systèmes supportent le matériel requis pour le LRU, mais le support du « bit de référence » est très répandu.

- **Description**
  - Un compteur logiciel par page.
  - Ajout du bit de référence au compteur à intervalle régulier.
- **Vieillissement**
  - Décalage des compteurs à droite avant addition.
  - Ajout du compteur à gauche plutôt qu'à droite.
- **Différences avec le LRU**
  - Granularité plus grossière (intervalles d'horloge plutôt que cycles CPU)
  - Historique moins précis

Algorithme de l'ensemble de travail (WS), 1970

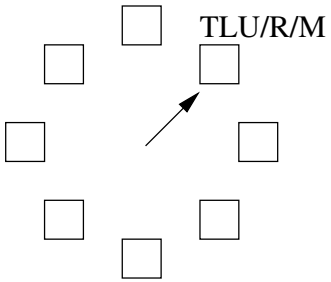
- **Phénomène de localisation :** l'ensemble des pages référencées dépend de « phases » dans l'exécution
- **Ensemble de travail :** $w(k,t)$ est l'ensemble des pages référencées à un instant t par les k derniers accès mémoire



Implémentation réelle impossible (trop coûteuse)

Implémentation approchée
Version temporelle plutôt que statistique

- **Description**
  - « Temps courant virtuel » : temps CPU utilisé jusqu'ici par processus
  - $w(t,k)$ devient l'ensemble des pages référencées à un instant t pendant les k dernières secondes de temps virtuel
- **Implémentation**
  - Maintien du « temps virtuel de dernière utilisation » pour chaque page
  - Procédure de sélection analogue à la seconde chance
- **Version « WSClock »** (1981)
  - Implémentation alternative
  - Liste circulaire de cadres de page



Algorithmes à bufferisation

- **Description**
  - Maintien d'un pool de cadres de page libres
  - Sélection d'une victime
  - Lecture de la page dans un cadre libre (à la place du remplacement)
- $\Longrightarrow$ Le processus peut donc redémarrer tout de suite
- **Ensuite**
  - Sauvegarde de la victime seulement quand le MMU est inactif
  - Pas de sauvegarde, donc récupération plus rapide
- **Remarque :** implémentation possible grâce à un « paging daemon »

Politiques de sélection des pages

- **Allocation globale :** considérer toutes les pages actuellement en mémoire. Difficulté de maîtriser son propre taux de défauts de page.
- **Allocation locale :** considérer seulement les pages du processus concerné. Nombre de cadres de page alloués fixe. Risque de ne pas profiter de pages non utilisées.

- Allocation globale plus performante en général (capacité de traitement)
- Mais certains algorithmes sont par définition locaux (WS, WSClock)

Répartition des cadres de page

- **Répartition fixe (ou initiale)**
  - **Allocation équitable** : répartition identique des cadres de page à tous les processus.
  - **Allocation proportionnelle** : nombre de cadres de page alloués en fonction de la taille du processus.
  - **Allocation mixte** : fonction de la taille du processus mais aussi de sa priorité.
  - **Prépagination** : éviter le surplus de défauts de page initial ou après swapping en ramenant plusieurs pages d'un coup (phénomènes de localisation).
- **Répartition dynamique**
  - Contrôler la fréquence des défauts de page. Allouer plus de pages aux processus très consommateurs, retirer des pages aux processus moins consommateurs.

Autres considérations

- **Contrôle de charge :** gérer l'« écroulement » (la somme des ensembles de travail dépasse la capacité mémoire). Swapping.
- **Pages partagées :** attention à leur éviction . . .
- **Interface avec la mémoire virtuelle :** donner à l'utilisateur un certain contrôle sur sa pagination. Implémentation efficace de l'envoi de message (passage de pages).
- **Structure des programmes** : organisation des structures de données et algorithmique sur celles-ci.

Redémarrage d'instructions

- Une instruction d'adressage complexe peut entraîner un défaut de page à plusieurs endroits (ex. Motorola 680x0 : `MOVE.L #6(A1), 2(A0)`).
- **Problème 1 :** Impossibilité fréquente de connaître l'adresse de début de l'instruction.
- **Problème 2 :** Auto-in(dé)crémentation de registres, avant ou après l'adressage.

$\Longrightarrow$ Maintient de l'adresse de début de l'instruction et des registres auto-in(dé)crémentés dans des registres cachés du CPU.

Verrouillage des E/S :

Problème : Éviction d'une page (algorithme global) pendant une attente de fin d'E/S. $\Longrightarrow$ Utilisation d'un bit « verrou » indiquant qu'une page ne doit pas être remplacée.

Autre utilisation du bit verrou : Éviter qu'une page ne soit chargée pour rien dans la mémoire (ex. remplacée immédiatement par un processus de plus haute priorité).

- **MacOS** (< X) : verrouillage systématique.
- **SunOS** : verrouillage possible, mais le système a le droit de ne pas verrouiller (ressources insuffisantes).
- **Solaris** : `root` peut demander à verrouiller des pages.