Série d'exercices #12

Solution

IFT-2245

9 avril 2019

12.1 Copy on Write

Qu'est-ce que le *copy-on-write* et dans quelles circonstances est-ce bénéfique ? Quel soutient matériel est nécessaire pour implanter cette fonction ?
En l'absence de soutient matériel pour le copy-on-write, quelle peut être une solution de rechange ?

12.1 Copy on Write

Qu'est-ce que le *copy-on-write* et dans quelles circonstances est-ce bénéfique ? Quel soutient matériel est nécessaire pour implanter cette fonction ?
En l'absence de soutient matériel pour le copy-on-write, quelle peut être une solution de rechange ?

Solution

Le *copy-on-write* est un mécanisme qui protège en écriture une zone mémoire. Il est utilisé dans le implantation du *fork* dans les système Linux et MacOSX, pour éviter les opérations de copie inutile. Il faut que le support d'un bit de *read*. Le noyaux peut garder trace des propriétés des pages et intercepter le *store* dans une page marquée comme *read-only* et la copier.

12.2 User level threads et page faults

Lorsqu'une page fault se produit, le processus qui tente d'accéder à la page doit bloquer en attendant la dite page. Supposons qu'il existe un processus contenant 5 user-level threads et que le mapping utilisé est many to one. Si un des threads entraîne une page fault en accédant la pile, est-ce que les autres user threads appartenant au processus devront attendre que la page fautive soit amenée en mémoire ?

12.2 User level threads et page faults

Lorsqu'une page fault se produit, le processus qui tente d'accéder à la page doit bloquer en attendant la dite page. Supposons qu'il existe un processus contenant 5 user-level threads et que le mapping utilisé est many to one. Si un des threads entraîne une page fault en accédant la pile, est-ce que les autres user threads appartenant au processus devront attendre que la page fautive soit amenée en mémoire ?

Solution

Oui, car il y a seulement un *kernel thread* pour les 5 *user threads*. C'est le *kernel thread* qui va bloquer suite à l'interruption et il sera bloqué tant que la *page fault* ne sera pas résolu. Ainsi, tout les *user threads* associés au *kernel thread* devront attendre.

12.3 Remplacement de pages

Considérez la suite de références de page suivante :

$$7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1$$

En supposant de la pagination à la demande avec 3 frames, combien de page faults y aura-t-il si on utilise les algorithmes de remplacement suivant :

1. Remplacement LRU (*Least Recently Used*)
2. Remplacement FIFO (*First In, First Out*)
3. Remplacement OPT (*Optimal*)

12.3 Remplacement de pages

Considérez la suite de références de page suivante :

$$7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1$$

En supposant de la pagination à la demande avec 3 frames, combien de page faults y aura-t-il si on utilise les algorithmes de remplacement suivant :

1. Remplacement LRU (*Least Recently Used*) 18
2. Remplacement FIFO (*First In, First Out*)
3. Remplacement OPT (*Optimal*)

12.3 Remplacement de pages

Considérez la suite de références de page suivante :

$$7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1$$

En supposant de la pagination à la demande avec 3 frames, combien de page faults y aura-t-il si on utilise les algorithmes de remplacement suivant :

1. Remplacement LRU (*Least Recently Used*) 18
2. Remplacement FIFO (*First In, First Out*) 17
3. Remplacement OPT (*Optimal*)

12.3 Remplacement de pages

Considérez la suite de références de page suivante :

$$7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1$$

En supposant de la pagination à la demande avec 3 frames, combien de page faults y aura-t-il si on utilise les algorithmes de remplacement suivant :

1. Remplacement LRU (*Least Recently Used*) 18
2. Remplacement FIFO (*First In, First Out*) 17
3. Remplacement OPT (*Optimal*) 13

12.4 Thrashing

Qu'est-ce que le *thrashing* ? Quelle est la cause du thrashing ? Comment est-il possible de le détecter ? Une fois détecté, comment est-il possible pour le système de régler le problème ?

12.4 Thrashing

Qu'est-ce que le *thrashing* ? Quelle est la cause du thrashing ? Comment est-il possible de le détecter ? Une fois détecté, comment est-il possible pour le système de régler le problème ?

Solution

Le thrashing se produit lorsqu'il y a une sous allocation des pages d'un processus par rapport au nombre minimum requis.Il en résulte une grande quantité de page fault. Le système peut détécter le thrashing en analysant l'utilisation du CPU comparativement au niveau de multi-programming. Il peut être réduit en réduisant le niveau de multi-programming.

12.5 Belady et LRU

Montrer formellement que l'algorithme de remplacement LRU n'est pas victime de l'anomalie de Belady.

12.5 Belady et LRU

Montrer formellement que l'algorithme de remplacement LRU n'est pas victime de l'anomalie de Belady.

Solution

L'anomalie de Belady apparait lorsque, pour un nombre supérieur de frames, l'algorithme de remplacement entraîne un plus grand nombre de page fault. Soit n le nombre de frames et $X := \{X_0, X_1, \ldots, X_k\}$ une suite ordonnée de références de page. L'anomalie de Belady se produit lorsque pour $j \in \mathbb{N}, n+1 < j \leq k, X_j$ entraîne un page fault lorsque le nombre de frames est $n+1$, mais pas lorsqu'il est de n. Lorsqu'on utilise l'algorithme LRU, et que X_j ne produit pas de page fault, ça veut dire qu'on accède a une des n dernières pages récemment utilisé. Or, si on augmente le nombre de frames et que X_j produit un page fault, c'est qu'on accède à une page qui n'a pas été récemment utilisée ($n+1$ pages la précéde). Or, on sait que X_j appartient au n dernières pages utilisée. Il est donc impossible d'avoir un page fault en augmentant le nombre de frames.

12.6 TLB et page faults

Supposons qu'un programme référence une adresse mémoire virtuelle. Décrire les scénarios pour lesquels ces événements peuvent se produire ? (expliquez si un scénario est impossible)

1. TLB miss et pas de page fault.
2. TLB miss et un page fault.
3. TLB hit et pas de page fault.
4. TLB hit et un page fault.

12.6 TLB et page faults

Solution

1. La page a été amenée en mémoire, mais enleve du TLB ;
2. La page a été enleve du TLB et n'est pas en mémoire ;
3. La page a été referencee recemment et donc se trouvedans le TLB et evidemment en mémoire ;
4. Le TLB est une cache dela table de page. Il est doncimpossible que la page soit dans le TLB mais pas en mémoire. De toutes façon, un TLB hit ne peux pas entrainer un page fault puisqu'on ne regardera jamais la table de page suite à un page fault.

12.7 Parcours de tableau

Soit le programme suivant :

```
int i, j ;
int data[128][128];
for (j = 0; j < 128; j++)
    for (i = 0; i < 128; i++)
        data[i][j] = 0;
```

Supposer que le système utilise LRU, des pages de 128 mots et un nombre de frames inférieur à 128. Expliquer en quoi cet exemple brise la transparence du système de pagination à la demande. Trouver un algorithme de remplacement simple qui donnerait de meilleurs résultats sur cet exemple.

12.7 Parcours de tableau

Solution

On a qu'une ligne du tableau est associée à une page. Or la boucle parcourt les rangées avant les colonnes. Il en résulte un accès constant à l'ensemble des pages plutôt que des accès concentré sur une page en particulier. Comme il y a moins de frame que lignes au tableau, il va y avoir un grand nombre de page fault (128×128). Le meilleur algorithme serait MRU (*Most Recently Used*).