Travail pratique #3 - Rapport - IFT-2245

Vincent G. Beauregard && Philippe Caron

April 27, 2017

Rapport

Lecture et compréhension de la matière

Aucune difficulté importante ne nous à particulièrement causé des problème à l'exception de la matière concernant les dirty-bits et le read-only. La documentation est clair du point de vue théorique mais son application pratique nous semblais quelque peu nébuleuse.

Lecture et compréhension du code fournis

Sur une vue d'ensemble, les sections du code nécessitant une modification de notre part était bien indiquées et les noms de fonctions très concrets à la matière, ce qui nous permettait de faire le pont entre la théorie et le code facilement.

Compléter le code fourni.

Le code fournis étant clair, nous étions bien guidé pour remplir les fonctions adéquatement. Le code fonctionne simplement de la façon suivante : sachant qu'il s'agisse d'une écriture ou d'une lecture, l'accès à une page devra toujours passer par une adresse logique de forme $pageNumber + offset$, une recherche via le TLB s'en suit. en cas de hit, la lecture de la page ce fait directement via la mémoire physique. Autrement, le programme se dirige vers la pageTable. En l'absence de page-fault, le TLB est mis à jour en FIFO et la lecture du page s'exécute. S'il y a présence de page-fault, si la table des pages est pleine, une page est alors évincé,selon un algorithme second chance, et la nouvelle page est téléchargée et mise en mémoire physique .

1. **Algorithme - Page Fault**

   Nous avons implémenté l'algorithme de remplacement des pages de façon à ce que pour chaque nombre d'accès prédéfini (dans RATE) le programme va placer à *true* le bit de référence de la page accédée. Le programme n'évince alors que les vielles pages en les parcourant de manière cyclique à chaque page-fault. À tous les passages, si ce bit est a *true*, il est réinitialisé à *false*, autrement la page sera évincé. Ainsi un fonctionnement LRU second-chance est mis de l'avant pour gérer les page-faults en utilisant un bit de référence dans la page table. Cette algorithme tends à être très semblable à FIFO si aucune bit de référence page n'a la chance d'être initialisé a *true* ou si au contraire, ils sont tous initialisé à *true*.

2. **Algorithme - TLB-miss**

   Nous avons implémenté l'algorithme FIFO tout à fait simple pour gérer les entrés du TLB. À chaque TLB-miss, l'ensemble des valeurs sont déplacer d'une case dans le TLB, la dernière est évincer et la nouvelle entré est positionné à la première case.

Test-Méthodologie

1. **command1.in - Efficacité du TLB**

   Notre implémentation du TLB étant en FCFS, le test vérifie les 8 dernière entrés différentes n'occasionne pas de TLB-miss.

   Nous occasionnons les entrée : //TODO et recevons tel qu'un algorithme FCFS le devrait //TODO TLB-miss

2. **command2.in - Efficacité table des pages**

   Afin de tester l'efficacité de notre algorthme, il nous a fallu simuler un ensemble identique suffisament souvent pour changé le bit de référence à *true* pour ensuite exploiter cette donnée.

   Nous occasionnons les entrée : //TODO et recevons tel qu'un algorithme FCFS le devrait //TODO page-fault

Conclusion

Au final, le programme fourni répond au mandat spécifié par la donnée du travail à l'exception de la gestion du COW. La modification du TLB ce fait en fifo lors de TLB-miss, et l'algorithme de remplacement de page se chargea de modifier les pages en mémoire physique avec la méthode second chance.