Travail pratique #1 - (IFT2245)

Waël ABOU ALI (p20034365)

April 30, 2019

1 Niveau personnel

Finalement, dernier tp pour cette session!C'était un plaisir pour moi de travailler sur ces tps malgrè que j'ai jamais trouver quelqu'un pour m'aider dans ces projets. C'était très stressant de début puisque j'avais à combattre le temps pour terminer soit avant de commencer les examens ou après la fin. Ce tp je le considère comme un "wrapping" pour tous qu'on a étudié dans cette session. Il travaille sur la synchronisation entre les différents niveaux dans un système d'exploitation. L'objectif principal de ce tp était de

1. Nous familiariser avec le fonctionnement du mémoire virtuelle et les algorithmes du remplacement des pages mémoire dans un système d'exploitation de style POSIX.

2. soldifier et perfectionner nos connaisances en POSIX, C et LATEX.

3. Compléter le code fourni en langage C.
   Ce code représente un programme qui simule **un gestionnaire de mémoire virtuelle par pagination(paging)**.

2 Difficultés

1. La difficulté majeure que j'ai confronté dans ce tp était de choisir le meilleur algorithme pour les pages TLB.

2. la synchronisation entre les différents fichiers était un peu mélangeant au début puisqu'il y en a plusieurs fichiers à compléter dont chacun à besoin de savoir c'est quoi les recommendations à respecter et les données à fournir pour rassembler les pièces ensembles.

3. Dernière chose était le choix des tests en command1 et 2.

3 Surprises

1. La plus grande et bonne surprise que j'ai vu dans ce tp était de comment c'était bien apprécié le très grand effort que vous nous avez demandé dans les deux premiers tps puisque ça m'a aidé personellement dans le troisième pour le terminer un peu plus rapide et plus efficace comme même.

2. Je pense que ça était la seule chose j'ai qui m'a étonné dans ce tp. Ce n'est pas un tp très difficile et je pense que la manque de temps était bien considérée avant de le publier.

4 Les choix

Les choix étaient de choisir entre les plusieurs algorithmes qui sont didés au TLB et la recherche des données à partir de la connexion entre mémoire physique, virtuelle, CPU et système d'exploitation.
Pour moi, j'étais mélée entre les algorithmes qui vont traiter les données s'il y en a un **"page fault"**.
En général, mes choix étaient :

1. L'algorithme de remplacement de la table des pages **"Second Chance"** avec un bit de référence pour vérifier si la page dans **"Page Table"** a été accédée ou pas ainsi que la gestion de l'état **"dirty"** des pages.

2. Deuxième choix était de remplacer les pages dans la TLB avec l'algorithme LRU.

5 Fonctionnement et démarches en bref

1. À fin de créer notre programme, on a du implémenter des fonctions dans les fichiers vmm.c, pm.c, pt.c et tlb.c y compris l'implémentation de l'algorithme de remplacement de la table des pages **"Second Chance"** avec un bit de référence pour vérifier si la page dans **"Page Table"** a été accédée ou pas ainsi que la gestion de l'état **"dirty"** des pages.

2. Le programme devra lire et exécuter une liste de commandes sur des adresses logiques. Pour y arriver il devra traduire chacune des adresses logiques à son adresse physique correspondante en utilisant une TLB **(Translation Look-aside Buffer)** et une table de pages(page table).

3. Pour tester l'efficacité de notre programme, on a du fournir deux fichiers command1.in et command2.in qui testent la performance de la TLB et l'algorithme de remplacement des pages consécutivement.

6 Les étapes

I. Paging & Paging Demand

Pour implémenter le schèma du paging, une TLB et une table de pages qui fonctionnent comme 2 niveaux de mémoire cache initialement vide dans lesquels on résèrve les frames qu'on vient d'accéder sous forme des adresses logiques. Leur rôle est **convertir des adressess logique à physique** en permettant d'obtenir la frame correspondante dans la mémoire physique au cas où la page demandée n'était pas cachée ou bien a été supprimée de la TLB ou de la table des pages. Pour la structure de la TLB, on a utilisé :-

1. Page Number.

2. Frame Number correspondant.

3. Un bit de référence **"Read Only"**.

Or, pour chaque entrée, on associe ces trois attributs, avec lesquels, on vérifie l'existence d'une page ainsi sa frame dans la mémoire physique à travers la fonction **tlb_lookup**.
Pour la structure de le page table, on a utilisé:-

1. Frame Number.

2. Bit **"dirty"** readonly.

3. Boolean **"valid"**.

Or, pour chaque entré, on associe ces trois attributs, avec lesquels, on vérifie l'existence d'une page dans le page table ainsi sa frame dans la mémoire physique à travers la fonction **pt_lookup**.

II. Algorithme & Page replacement

Pour gérer le remplacement des pages dans la TLB, on utilisé l'algorithme **"FIFO (first in first out)"**.
Dans l'autre sens, on a utilisé l'algorithme **"Second Chance"** dans la table des pages.

6.1 Stratégies de remplacement

6.1.1 FIFO

Pour l'algorithme FIFO(first-in, first-out), la plus vieille page dans la TLB est celle qui est remplacée par la nouvelle qui prendra sa place.
Un compteur **fifo_count** a été rajouté à la TLB qui est incrémenté à chaque fois qu'une nouvelle page est inséré dans la TLB et chaque entrée de la TLB a son numéro fifo qui **permet de savoir sa position dans la file (plus le numéro est petit plus cette entrée est ancienne)**.

Avantages

- Simple et facile à implémenter.

- Si on suppose qu'une fois une page utilisée, ses chances d'être réutilisée dans le futur sont faible, c'est un bon algorithme à utiliser.

Inconvenients

- Concernant le 2ième point, ce n'est pas le cas en pratique.

- Même si on utilise fréquemment une page, elle sera quand même remplacé dû à sa ancienneté.

- D'une part, il se peut que la page soit un module d'initialisation qui a été utilisé il y a longtemps et dont nous n'avons plus besoin, mais d'autre part, cette page pourrait contenir une variable couramment utilisée qui a été initialisée tôt dans le programme.
  C'est une situation dont l'algorithme ne tient pas compte. Toute-fois, l'exécution se passe correctement, elle est simplement ralentie lorsque nous remplaçons une page utilisée, car une faute est signalée presque immédiatement et la page est récupérée ensuite.

- Souffre de l'**anomalie de Belady**.

Dans ce cas, la taille d'une TLB sera trés petite(8 entrées au total) et on n'a pas besoin à implémenter un algorithme plus complexe pour la gérer. L'algorithme détecte la page qui demeurait longtemps dans la TLB si jamais la TLB est pleine avec le counter **tlb_fifo_count**, pour qu'elle soit remplacée avec la nouvelle page récemment appelée, en appellant la fonction **tlb_add_entry**.

6.1.2 Second Chance(CLOCK)

Pour l'algorithme **"Second chance"**, c'est un algorithme un peu plus complexe mais plus efficace que les autres algorithmes de remplacement. Il consiste à avoir un bit **"referenced"** qui représente la dernière page référencée qui est mis à 0 si la page n'est pas récemment référencée et à 1 si l'inverse.
Cet algorithme dégénère à FIFO si tous les bits sont tous à 0 (ou à 1).

Avantages

- Plus efficace que les autres algorithmes.

- Dans le cas de sous utilisation ou sur utilisation de la table de pages où l'algorithme devient FIFO qui est bon et efficace à utiliser dans ce cas.

- Évite les problèmes des autres algorithmes comparables comme LRU.

- Ne souffre pas de l'**anomalie de Belady**.

Inconvenients

- Plus complexe à implémenter.

Si on a besoin d'un algorithme plus complexe et efficace pour le changement de pages, alors l'algorithme de **"Second chance"** est très efficace à implémenter. **Pour choisir la page victime** qui sera remplacée lorsqu'il y a un **"page fault"** en appellant la fonction **pt_set_entry**.

III: Traitement de Page Fault

On appelle la fonction **tlb_lookup** pour vérifier si la page demandée est déjà dans la TLB. Si oui c'est un hit. Sinon, c'est un miss et on appelle la fonction **pt_lookup** pour chercher la page dans la table des pages. Si trouvée, alors on la télécharge dans la TLB. Sinon, c'est un **"page fault"**. On appelle la fonction **pageFault** pour charger la page demandée du fichier **backing_store**. Si, le frame associé à la page demandée contient des données, on utilise l'algorithme de **"Second chance"** pour choisir la victime à enlever. Ensuite, on appele la fonction **pm_backup_frame** pour sauvegarder les données dans le frame victime dans **backing_store**. Puis, on met le numéro du frame comme **"invalide"** avec la fonction **pt_set_entry** dans la table des pages. Puis, on télécharge le nouveau frame du **backing_store** et on met à jour la table des pages ainsi que la TLB.