Travail pratique #3 - IFT-2245 - Rapport

Ahmed Medhat (x) Youssef Zaki (20052467)

April 27, 2017

1 Objectifs

Ce TP vise a nous familiarisé avec la programmation système dans un système d'exploitation de style POSIX.

1. Parfaire notre connaissance de Python et POSIX.
2. Lire, trouver et comprendre cette donnée.
3. Compléter le code fourni pour implémenter en langage C un programme qui simule un gestionnaire de mémoire virtuelle par pagination (paging). 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 un TLB (Translation Look- aside Buffer) et une table de pages (page table).
4. Savoir comment écrire un rapport en LaTeX.

2 Description de l'implantation

A fin de créer notre programme, on a du implémenter les fonctions vmm.c, pm.c, pt.c et tlb.c y compris l'implémentation de l'algorithme de remplacement du TLB "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. Finalement, a fin de tester l'efficaité de notre programme, on a du fournir deux fichiers command1.in et command2.in qui testent la performance du TLB et l'algorithme de remplacement des pages consécutivement.

I. Paging & Paging Demand

Pour implémenter le schème du paging, on a du utilisé un TLB qui fonctionne comme une mémoire cache initialement vide dans lequel on stock les frames qu'on vient d'accéder. Puis, le page table qui fonctionne comme un convertisseur des adressess logique à physique, nous 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 du TLB. 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 entré, on associe ces trois attributs avec lesquels on vérifie l'existence d'une page ainsi sa frame dans la mémoire physique a 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 a travers la fonction pt_lookup.

II. Algorithme & Page replacement

Pour gérer le remplacement des pages dans le TLB, on utilisé l'algorithme "Least Recently Used" qui détecte la page qui demeurait longtemps inutilisé avec le counter tlb_lru_counter pour qu'elle soit remplacé avec la nouvelle page récemment appelé en appellant la fonction tlb_add_entry.

Dans l'autre coté, on a utilisé l'algorithme "Second Chance" pour choisir la page victime qui sera remplacé lorsqu'il y a un page fault en appellant la fonction pt_set_entry.

III: Traitement de Page Fault

Dans le cas d'un TLB-miss, on appelle la fonction tlb_lookup qui vérifie si la page voulue existe déjà dans notre table de pages. Si c'est le cas, on obtient directement le frame correspondant dans la mémoire physique. Sinon, on vérifie si la page se trouve dans le page table en appellant la fonction pt_lookup. Dans le pire cas, un Page Fault est produit en appellant la fonction pageFault et on procède a lire une page dans un fichier BACKING_STORE.txt et la stocker dans un frame disponible dans la mémoire physique initialement vide, toujours en mettant à jour notre TLB et la table de pages.

IV: Bugs & Difficultés rencontrées

V:Tests