Travail pratique #3 - IFT-2245

Boumediene Boukharouba 20032279
Alexandre Deneault 20044305

April 13, 2017

1 Description du travail

Le troisième travail pratique consistait à implanter une simulation d'un algorithme de gestion de mémoire virtuelle. Notre programme devait être capable d'adresser un espace virtuel de 2^{16} octets. Il doit faire la traduction de l'adresse logique vers l'adresse physique en utilisant un TLB et une table des pages. Lorsque la page demandée ne se trouve pas en mémoire, il libère l'espace mémoire dans lequel il va charger la nouvelle page en évinçant la page qui s'y trouve, la sauvegardant dans le fichier lorsque nécessaire. Le programme effectue ensuite les opérations démandées (lecture et écriture). Lorsque le programme termine, il sauvegarde les pages qui ont été modifiées sur le disque/fichier.

2 Recherche

2.1 TLB et table des pages

Afin de réaliser le travail, nous avons revus les notions sur le TLB et la table des pages dans les notes de cours. Nous avons étudiés plus en détails la structure de chacun de ces composantes afin de pouvoir les traiter correctement. Nous avons aussi étendu nos recherches à la conversion des adresses logiques en adresses physiques. Revoir les notes du cours nous a semblé suffisant pour accomplir le travail.

2.2 Algorithme de remplacement

Nous avons aussi revus les différents algorithmes de remplacement de page. Nous avons finalement décider d'utiliser un algorithme FIFO pour le TLB et un mélange de FIFO et LRU de style "Clock" pour la table des pages. Ces deux algorithmes nous semblaient être adapté à la tâche et un bon compromis pour l'efficacité.

3 Implémentation

3.1 Mémoire physique

Nous avons débuté le travail par l'implémentation du transfert des caractères du fichier vers la mémoire physique et de la mémoire physique vers le fichier, ainsi que la lecture et l'écriture des caractères en mémoire. Nous avons aussi ajouté une structure pour pouvoir enregistrer les pages qui se trouvent en mémoire physique sur le fichier lorsque le programme se termine.

3.2 Table des pages

Il n'y avait rien de compliqué à faire pour la table des pages. Nous avions déjà étudié et bien compris la structure de la table. Nous avons simplement complété les fonctions présentes en sauvegardant les données passées en paramètre aux bons endroits dans le tableau représentant les pages ou en retournant les informations demandées.

3.3 TLB

Nous avons ensuite complété la section sur le TLB. Nous parcourons le tableau des entrées du TLB pour voir si la page recherché s'y trouve. Nous retournons la "frame" correspondante si nous trouvons la page, sauf si la page est marqué en lecture seulement. Pour l'ajout, nous vérifions d'abord si la "frame" à ajouter fait partit des "frames" qui sont marqués par le TLB afin d'attraper les modifications de l'état de "lecture seulement" et les pages qui en ont évincées une autre qui se trouvait dans le TLB. Si ce n'est pas le cas nous évinçons l'entrée la plus vieille du TLB pour pouvoir y placer la nouvelle page.

3.4 VMM

Lorsque le vmm reçoit une commande, il commence par traduire l'adresse logique en adresse physique. Les 8 premiers bits représentent l'offset et les 8 derniers, le numéro de page. L'adresse physique de la page est d'abord recherché dans le TLB. Si le TLB ne donne pas de résultat, il regardons dans la table des pages. Si la table des pages retourne le numéro de "frame", vmm ajoute l'entrée dans le TLB et continue la conversion de l'adresse, sinon la procédure de "page fault" est enclenchée. Lors d'un page fault, vmm choisit d'abord dans quel "frame" il va placer la nouvelle page à l'aide de l'algorithme "clock". Si nécessaire, il va enregistrer la page qui s'y trouve dans le fichier et marquer l'entrée de la table des pages invalide avant d'y installer la nouvelle page. Vmm va ensuite ajouter l'entrée dans la table des pages et dans le TLB. Vmm va ensuite terminé la conversion de l'adresse logique en adresse physique et appeler la fonction appropriée (read ou write).

3.5 Tests

Nous avons décidé d'écrire un programme en python pour générer les fichiers de test. Pour le premier test, nous avons choisit 32 pages. Nous avons ensuite écrit le programme de façon à choisir aléatoirement parmis 12 page à la fois. Il choisit aussi aléatoirement le nombre d'accès consécutifs qu'il fait sur cette page (entre 1 et 5). Le programme choisit aussi aléatoirement l'index du caractère dans la page et s'il effectue une lecture ou une écriture. Lorsqu'il écrit il écrit le caractère 'a'. Nous avons décidé d'utiliser cette technique pour générer le test puisqu'il aurait été facile de contrôler la performance du TLB en choisissant quand faire des accès séquentiels et quand accéder à une autre page. Nous avons pensé que guider des accès aléatoire représenterait mieux l'utilisation de quelqu'un qui ne connait pas l'algorithme utilisé.

Pour le deuxième test, nous avons testé le remplacement des pages en commençant par réécrire toute la mémoire. La première page est rempli de 'a', la deuxième est rempli de 'b', etc. Ensuite, le fichier de test accède une fois en lecture à chacune des pages. Finalement, il accède une fois en lecture puis une fois en écriture à toute les pages, y écrivant le caractère de la page suivante à la position 100 ('b' pour la première page, 'c' pour la deuxième page, etc.). De cette manière, il est facile de vérifier l'état du fichier après le test.

4 Conclusion

En conclusion, ce travail était intéressant. Les consignes et le code fournit, la structure du travail en général, étaient beaucoup mieux que pour le deuxième travail pratique. Nous n'avons pas eu à faire de grosse recherche et la matière nécessaire à la réalisation du travail avait déjà été vue en classe. Il nous a permis d'approfondir nos connaissances de la mémoire virtuelle. C'était un bon travail.