Système d'exploitation – TP3

Vincent Antaki et Guillaume Poirier-Morency

Pour compiler, un `Makefile` est fournit, alors il suffit de lancer la commande:

```
make
```

Pour alterner d'un algorithme de mise en cache à un autre, il suffit de changer la directive `#define <algorithme>` dans le fichier `common.h`:

```
#define LRU
#define FIFO
#define LFU
```

La séquence d'action qui a été implantée dans l'exécution d'une commande est la suivante:

1. vérifier si la page se trouve dans le TLB
2. sinon, vérifier si la page est encore chargée en mémoire
3. sinon, demander la page à la mémoire physique
4. sinon, swapper un frame qui n'est pas dans le TLB avec la page désirée
5. ajouter la page au TLB

Pour l'allocation initiales des frames, un compte est gardé pour mémoriser combien de frames ont été alloués. Une fois tous les frames alloués, la stratégie qui suit consitue à faire du *swapping* en fournissant un frame que l'on souhaite utiliser pour charger la page demandée. Le *write back* n'a pas été implanté, car aucune opération d'écriture n'est permise dans le cadre du travail.

Trois algorithmes (FIFO, LRU et LFU) ont été implantés pour le TLB et peuvent être conditionellement inclus à l'aide d'une définition de pré-processeur.

Stratégie de remplacement pour le TLB

Pour la TLB, nous avons utilisé LFU comme algorithme de remplacement car, des trois algorithmes implantés, c'est celui qui se comportant le mieux avec l'exemple "addresse.txt".

Chaque algorithme de remplacement à ses cas où il

- LFU fonctionne bien lorsqu'un nombre limité de frames ont un nombre d'accès beaucoup plus élevé que les autres et que ceux-ci sont appelés...
- LRU fonctionne bien lorsque les accès à une frame particulière sont rapprochés.
- FIFO fonctionne bien lorsque les accès à une frame particulière sont très rapprochée, voir quasi-consécutive.

On peut observer assez clairement la différence au niveau du *TLB hit rate* entre les trois algorithmes.

Algorithme	Page-Fault rate	TLB hit rate
LFU	0.018%	0.842%
LRU	0.018%	0.587%
FIFO	0.018%	0.587%

Page fault ratio

Avec le premier fichier exemple où 18 pages sont chargés, le *page fault ratio* est le même pour tous les algorithmes utilisés, car le nombre de pages à charger est inférieur à la taille du TLB. Toute les pages se trouvent dans la cache, alors le *TLB miss* reste constant.

Dans le cadre du deuxième exemple, un plus grand nombre de page est demandé de manière aléatoire et on peut observer un nombre très élevé de *page fault*. Les algorithmes de cache sont conçu pour exploiter la localité temporelle ou spatiale, ce qui ne se retrouve pas dans une distribution aléatiore.

Si le nombre de page à charger est inférieur au nombre de place en mémoire physique, le *page fault ratio* sera toujours "le nombre de page à charger"/" le nombre total de requete".

Pour "adresse.txt", il est de 18/1000 car seul 18 page sont utilisé dans l'exemple

Pour changer l'algorithme de remplacement, il changer la ligne "#define LFU" dans le fichier "common.h" par "#define FIFO" ou par "#define LRU". Il est important de n'avoir qu'un seul de ces define de présent sinon on se retrouve avec plus qu'une fonction addentry pour le TLB.

Avec 256 frames et 18 entrées dans le TLB, on obtient les résultats suivants:

Algorithme	Page fault ratio	TLB hit rate
FIFO	0.36%	6.22%
LFU	0.36%	6.28%
LRU	0.36%	5.94%

Gestion d'une petite mémoire physique

Notre implémentation utilise les définitions du fichier `common.h`, alors il suffit de les changer et de recompiler. L'implémentation fait du *swapping* dans le cas ou tous les frames sont utilisés et qu'une page doit être chargée en mémoire. La première page qui ne se trouve pas dans le TLB est alors remplacé par la page désirée.

Les même comparaisons d'algorithmes avec une 128 frames et 256 pages donnent les résultats suivants:

Algorithme	Page fault ratio	TLB hit rate
FIFO	49.96%	6.23%
LFU	49.87%	6.21%
LRU	49.87%	6.20%