HEIG-VD | REDS Institut Reconfigurable and Embedded Digital Systems

Systèmes d'exploitation (SYE)

Profs Daniel Rossier, Fiorenzo Gamba, Marina Zapater
Assistants : David Truan, Lucas Elisei, Mattia Gallacchi

Algorithmes de remplacement de page

lab07 (20.12.2021)

Objectifs de laboratoire

Ce laboratoire porte sur les différents algorithmes de remplacement de page dans la mémoire physique lorsque nécessaire.
Le but ici est d'implémenter l'algorithme LRU (*Least Recently Used*) ainsi qu'une version simplifiée de *WSClock*.

Echéance

- Le laboratoire sera rendu au plus tard la veille du prochain laboratoire (**durée : 4 semaines**)

Validation du laboratoire

Selon la méthode énoncée en début de semestre, **chaque fin de session** doit aboutir à l'exécution d'un script qui transmettra le travail fourni. Chaque laboratoire s'accompagne d'un script de rendu appelé *rendu.sh* qu'il est **nécessaire** d'appeler à la fin de chaque session :

```
reds@reds2021:~/sye/sye21_lab07$ ./rendu.sh session
```

Cette commande va générer un dossier *rendu* qui contiendra les différences avec le dépôt de base. Lorsque le laboratoire est terminé, il suffit **d'exécuter le script** en lui spécifiant **fin** comme paramètre :

```
reds@reds2021:~/sye/sye21_lab07$ ./rendu.sh fin
```

Cette commande va générer un dernier fichier de différences, puis archiver le dossier *rendu*. Il est demandé de **déposer cette archive** (*rendu.tar.gz*) **dans Moodle à la fin du laboratoire.**

Etape 1 - Mise à jour du dépôt (environnement)

La commande suivante permet de récupérer une mise à jour du dépôt pour la réalisation de ce laboratoire.

```
reds@reds2021:~/sye$ retrieve_lab sye21 lab07
```

Ceci va créer un dossier « **sye_lab07** » qui contiendra les fichiers nécessaires au laboratoire.

Etape 2 – Implémentation de l'algorithme LRU

Cette étape consiste à implémenter l'algorithme LRU « classique ». La configuration de la mémoire est la suivante : **3 pages physiques** et **16 pages virtuelles.**

Pour cela, le fichier « ***usr/src/memreplace.c*** » déclare une table de page gérant 16 pages virtuelles (variable globale « ***page_table*** ») et initialise au démarrage du programme la table des pages avec les pages 0, 1 et 2 en RAM et la 3 en « *swap* ». La fonction « *main()* » implémente déjà une base pour l'étape 2 et 3 ; pour cette étape, il s'agit de considérer le code lorsque la condition LRU est « vraie ».

a) Créer un tableau intégrant la notion de compteur pour chaque page virtuelle (max 16 pages).

b) Ecrire le contenu de la fonction « ***incCompteur()*** ». Elle doit permettre d'incrémenter un **compteur global** à chaque accès à une page et de mettre à jour le compteur de la page accédée avec la valeur courante du compteur global (adapter l'argument de la fonction en conséquence).

c) Etudier et éditer la fonction « ***main()*** » afin d'appeler la fonction « *incCompteur()* » lors de l'accès à une page par le processus (dans notre situation, un seul processus est présent, c'est le programme lui-même).

d) Enfin, écrire le contenu de la fonction « ***replaceLRU()*** » qui recherche la page avec la plus petite valeur de compteur (page la plus ancienne)

   ⇒ Mettre le bit « *valid* » à 0 et le bit « *swap* » à 1 de la page qui est déchargée.

e) Tester et valider en exécutant la commande « ***memreplace LRU*** **»** ; un ensemble de 7 pages simulées sont accédées par le processus selon la séquence suivante : **5 2 2 6 1 3 1 4**

```
so3% memreplace LRU
RAM : [0] [1] [2]
SWAP : [3]
Enter the page to be accessed:
```

   ⇒ Le résultat final attendu est :

```
RAM : [1] [3] [4]
SWAP : [0] [2] [5] [6]
```

f) Simuler le comportement sur papier de la version « classique » de l'algorithme LRU et comparez avec l'affichage obtenu pour valider l'implémentation.

Etape 3 – Evolution vers l'algorithme WSClock

Dans l'étape précédente, la page déchargée est la première page présente en RAM dont le bit **R** de la PTE est à zéro. Cette étape propose d'affiner la sélection de la page en appliquant la fenêtre du *working set* issue de l'algorithme WSClock avec les estampilles temporelles (*timestamps*) des pages. La configuration de la mémoire est la suivante : 3 pages physiques et 16 pages virtuelles.

Cette étape est réalisée dans la partie consacrée à l'algorithme WSC dans la fonction « ***main()*** ».

a) Créer un compteur représentant le **TVC** (Temps Virtuel Courant), que nous simplifierons en l'incrémentant à chaque accès à une page**.**

b) Créer un tableau intégrant la notion de **TDU** (Temps Dernière Utilisation) des pages lorsqu'elles sont référencées (max 16).

c) Ecrire la fonction « ***updateTDU()*** » qui a pour but de mettre à jour les **TDU** des pages à **TVC** lorsque le bit R vaut 1. Adapter la partie *WSC de la* fonction « ***main()*** » en conséquence lors de l'appel à une page pour que la fonction soit appelée à chaque fois.

d) Editer la fonction « ***main()*** » afin de modifier les propriétés suivantes de la page lors de son appel par le processus :

- Bit de référence défini à 1
- Bit de validité défini à 1

e) Ecrire le contenu de la fonction « ***replaceWSC()*** », qui définit la page à décharger dans le cas où la mémoire physique est pleine et que la page recherchée n'est pas présente, en considérant l'approche suivante :

   ⇒ Appliquer une seconde chance si R vaut 1, en passant R à 0. Si R vaut 0, tester si la page est dans l'ensemble de travail avec la fenêtre d'observation *delta =2*. Si la page est hors de l'ensemble de travail, décharger la page.

   ⇒ Si aucune page ne remplit les conditions pour être déchargée, supprimer la première page en RAM (**Ne pas oublier de** passer le bit « *valid* » à 0 et le bit « *swap* » à 1.

f) Tester et valider en exécutant « ***memreplace WSC*** ».

   ⇒ Le même résultat est attendu pour la séquence précisée au point f) de l'étape précédente.