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

Mécanismes de pagination

lab06 (06.12.2021)

Objectifs de laboratoire

Ce laboratoire porte sur les notions d'ordonnancement, en particulier la gestion de la priorité au niveau des processus. Il aborde également différents aspects relatifs à la gestion mémoire.

Echéance

- Le laboratoire sera rendu au plus tard la veille du prochain laboratoire (**durée : 2 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_lab06$ ./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_lab06$ ./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 lab06
```

Ceci va créer un dossier « **sye_lab06** » qui contiendra les fichiers nécessaires au laboratoire.

Etape 2 – Implémentation d'un algorithme de traduction d'adresse

Cette étape consiste à implémenter le mécanisme de traduction d'adresse opéré par la MMU durant les opérations de lecture/écriture à une adresse virtuelle.

Pour cela, le fichier « ***usr/src/memsim.c*** » déclare une mémoire fictive (variable globale « ***main_mem*** ») de 64 KB organisée en **256 pages** de **256 bytes.** Dans le cadre de ce laboratoire, l'espace d'adressage virtuel sera de même taille que l'espace d'adressage physique. L'adressage est sur **16 bits** et la pagination est à un niveau.

Les prototypes des fonctions à implémenter sont disponibles dans le code. L'initialisation de la table des pages (variable **« *page_table*** ») est déjà implémentée.

⇒ La table des pages est initialisée de telle sorte à ce que toutes les pages virtuelles soient mappées dans l'espace mémoire physique (il y a donc déjà un mappage existant).
⇒ L'architecture de l'adresse virtuelle est de type **(8 | 8).**
⇒ Les PTEs sont sur 16 bits, bien qu'il ne soit pas utile de disposer d'autant de bits pour encoder le numéro de page physique.
⇒ Examiner comment les bits sont utilisés à ce stade.

a) Implémenter la conversion d'une addresse physique en pte dans la fonction « **virt_to_pte()** »
b) Implémenter l'écriture d'un byte en mémoire dans la fonction « ***store_byte()*** ».
c) Implémenter la lecture d'un byte en mémoire dans la fonction « ***get_byte()*** ».
d) Une fonction permettant de printer une page mémoire « **print_page()** » est fournie et peut être utilisé pour débugger.
e) Tester et valider en executant memsim. La fonction « **test_mem()** » sera exécuté pour vérifier le remplissage de la mémoire. Pour que le programme exécute cette fonction mettez à «1» les « #define » nécessaires.

Etape 3 – Ajout de bits d'attribut dans les PTEs

Cette étape consiste à augmenter les PTEs en rajoutant des bits d'attribut.

a) Ajouter un moyen de spécifier si la page est en *« **READ/WRITE/EXECUTE*** » et adapter les fonctions d'accès en conséquence.
b) Configurer les pages **10 à 12** en lecture seule (*read only*) et valider le fonctionnement en testant l'écriture dans ces pages ; une telle écriture doit provoquer une erreur (afficher simplement un message dans ce cas).
c) Ajouter un bit d'attribut « ***VALID*** » qui permettra d'indiquer si la page est mappée ou non.
d) Modifier la fonction d'initialisation de telle sorte à ce que les pages **5 à 8** provoquent une faute de page (pas de mappage pour ces pages). Une faute de page conduira à l'affichage d'un message d'erreur.
e) Tester et valider en executant *memsim*. La fonction « **test_mem2()** » sera exécuté pour vérifier les attributs. Pour que le programme exécute cette fonction, il faut mettre à «1» les « #define » nécessaire

Etape 4 – Changement de mémoire

Dans cette étape, on souhaite changer le nombre de pages et leur taille. La mémoire aura 64 pages avec une taille de 1kB. Ces modifications auront bien entendu une incidence sur l'architecture de l'adresse virtuelle/physique.

⇒ Adapter l'ensemble du code afin de supporter cette nouvelle configuration.
⇒ Tester et valider le fonctionnement avec différentes lectures/écritures comme au point d) de l'étape 2 et mettant les « #define » à 0.