HE-IG 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

Ordonnancement et gestion mémoire

lab05 (22.11.2021)

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_lab05$ ./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_lab05$ ./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 lab05
```

Ceci va créer un dossier « **sye_lab05** » qui contiendra les fichiers nécessaires au laboratoire.

Etape 2 – Ajout de l'appel système « *renice()* » et gestion des priorités

Nous avons modifié *SO3* pour qu'il utilise une politique d'ordonnancement par priorité statique. L'implémentation de cet algorithme est disponible dans le fichier « *so3/kernel/schedule.c* ».

Lorsqu'un processus est créé, la priorité par défaut (**10**) lui est assignée. La commande du shell « ***renice*** » doit permettre de changer à la volée cette priorité. Selon la convention vue en cours, plus la valeur numérique de la priorité est haute, plus le processus est prioritaire.

L'appel système « ***renice()*** » a été rajouté, mais sans implémentation. L'application « ***time_loop*** » sera utilisée dans les différents tests.

⇒ Il est utile de retenir que le **shell** a le PID **1**.

a) Compléter l'implémentation de **« *renice()*** » dans le fichier « *so3/kernel/process.c »* (noyau) afin de pouvoir changer la priorité des processus.
b) Lancer l'application « ***time_loop*** » en arrière-plan (avec le symbole &) depuis le *shell*.

⇒ Que se passe-t-il et pourquoi ? (Répondre dans le fichier « ***rapport_lab05.md*** »)

c) Changer la priorité du *shell* pour que le processus « ***time_loop*** » deviennent plus prioritaire.

⇒ Que se passe-t-il et pourquoi ? (Répondre dans le fichier « ***rapport_lab05.md*** »)

Vous pouvez continuer à explorer les différents scénarios créés par l'ordonnancement par priorité statique en lançant par exemple deux processus « ***time_loop*** » et en jouant avec leur priorité respective.

Etape 3 – Exploration de la gestion mémoire dans le noyau de SO3

Cette étape consiste en une exploration du code de la gestion mémoire, le but étant de localiser les différentes parties du code du noyau SO3 qui sont responsables de l'allocation mémoire et de la gestion de la MMU notamment.

Au préalable, il faut réactiver l'ordonnancement utilisant le *Round Robin* en exécutant les commandes suivantes :

```
reds@reds2021:~/sye/sye21_lab05$ cd so3
reds@reds2021:~/sye/sye21_lab05/so3$ make vexpress_full_defconfig
reds@reds2021:~/sye/sye21_lab05/so3$ cd ..
reds@reds2021:~/sye/sye21_lab05$ make
```

a) Où se trouve le code gérant l'allocation dynamique dans le noyau (gestion du tas) ?
b) Quel est l'algorithme de gestion mémoire utilisé dans ce contexte ?
c) Où se trouve le code de la gestion de la MMU ?
d) Compléter l'implémentation de la fonction « ***do_translate()*** » dans le fichier « so3/arch/arm32/mmu.c » afin qu'elle retourne l'adresse physique correspondante à l'adresse virtuelle passée en paramètre.
e) Compléter l'application « ***memory »*** (fichier *memory.c*). On vous demande d'allouer dynamiquement une plage mémoire de 4 KiB, de faire appel à l'appel système « ***sys_translate()*** » implémenté précédemment, et enfin d'afficher les adresses virtuelle et physique de la plage mémoire. N'oubliez pas de clore proprement les ressources utilisées.
f) Exécuter plusieurs fois l'application « ***memory*** ».

⇒ Que constate-on au niveau des adresses physique et virtuelle ? Pourquoi ? (Répondre dans le fichier « ***rapport_lab05.md*** »)