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

Threads et tube (IPC)

lab03 (18.10.2021)

Objectifs de laboratoire

Ce laboratoire permet d'exercer la notion de *thread* ainsi que l'utilisation des tubes (*pipes*) anonymes.

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_lab03$ ./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_lab03$ ./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 lab03
```

Ceci va créer un dossier « **sye_lab03** » qui contiendra les fichiers nécessaires au laboratoire.

Etape 2 - Création et exécution de *threads*

Un processus peut contenir un ou plusieurs *threads*. Cet exercice permet d'exercer le démarrage et de gérer l'exécution de quelques threads à l'intérieur d'un processus. L'application qui servira de test est constituée du fichier « *usr/src/threads.c* ».

a) Compléter le fichier *threads.c* afin que celui-ci permette la création de N *threads* (N passé en paramètre) exécutant en parallèle une même fonction qui devra afficher la valeur d'un compteur allant de 0 à 100. Chaque ligne affichée devra comprendre également un préfix permettant l'identification du *thread* en cours d'exécution (l'identifiant devra être passé en paramètre à la création du *thread*.).
b) Limiter le nombre de threads à 15 au maximum

c) Modifier le code afin que l'exécution de chaque *thread* soit maintenant déterministe, c-à-d que chaque *thread* s'exécute l'un après l'autre.

⇒ On demande à ce que le programme puisse être lancé avec les deux variantes à l'aide d'un argument au démarrage du programme, comme suit :

```
so3% threads <N> p              # exécute l'application en version "parallèle"
so3% threads <N> s              # exécute l'application en version "séquentielle"
```

⇒ Si l'exécution en parallèle des threads n'est pas visible, augmenter le nombre d'itération à 500 ou 1'000 jusqu'à voir les compteurs itérer simultanément.

Etape 3 – Communication *client-serveur* au travers de tubes (*pipes*)

Un moyen simple pour communiquer entre processus est l'utilisation de tubes (*pipes*). Afin de mettre en pratique ce moyen de communication, cette étape propose de reprendre le jeu *tictactoe* développé au cours du précédent laboratoire et d'implémenter la communication entre le « *game manager* » et le « *player* » avec des pipes.

Le fichier « *tictactoe.h* » contient les prototypes des fonctions à implémenter pour permettre la communication entre les deux processus. Le fichier « ***tictactoe_ipc.c*** » doit être modifié pour que les fonctions qu'il contiennent aient le comportement suivant :

- **ipc_init()** : initialise les deux tubes contenus dans la structure *sye_ipc_t* (définie dans *tictactoe_priv.h*) et préserve les « *files descriptors* » de chaque tube dans cette même structure.
- **ipc_close()** : ferme les tubes.
- **(player) gm_get_cmd()** : lit la commande reçue dans le pipe correspondant.
- **(player) gm_send_cmd()** : écrit la commande dans le pipe correspondant.

- **ipc_player_argv1()** : retourne le *file descriptor* utilisé par *player* pour lire les commandes.
- **ipc_player_argv2()** : retourne le *file descriptor* utilisé par *player* pour écrire les commandes.
- **ipc_init_child()** : initialise les champs « *player_recv* » et « *player_send* » de la structure *sye_ipc_t* avec les *files descriptors* passés en argument.

⇒ **Attention !** Les fichiers « *tictactoe_gm.c* » et « *tictactoe_player.c* » ne doivent pas être modifiés. L'application finale devra avoir le même comportement que celle du laboratoire numéro 2.