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

Signaux et sockets réseau (IPC)

lab04 (08.11.2021)

Objectifs de laboratoire

Ce laboratoire permet d'exercer l'utilisation des *sockets* permettant la communication via le réseau. Il exerce aussi l'utilisation de signaux et de leur handlers associés.

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_lab04$ ./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_lab04$ ./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 lab04
```

Ceci va créer un dossier « **sye_lab04** » qui contiendra les fichiers nécessaires au laboratoire.

Etape 2 – Mise en place des signaux

Cette étape propose de prendre en main le traitement et l'utilisation des signaux. Une mise en place de « *handler* » ainsi que des tests seront réalisés. Ces manipulations sont à effectuer dans le fichier **socket_client.c.**

a) Ajouter des handlers des signaux suivant dans le programme du client :

- Les signaux **SIGUSR2** et **SIGUSR1** doivent utiliser le même handler
- **SIGINT** utilise son propre handler

b) Tester les handlers des signaux en affichant le numéro du signal dans les handlers.

Pour tester le fonctionnement, lancer l'application **socket_client** en *background* en ajoutant le symbole **&** à la fin de la commande

```
so3 % socket_client &
```

La commande ***kill*** permet d'envoyer un signal à un processus via son PID. Par exemple, pour envoyer le signal **SIGURS2** au processus ayant le PID **2** :

```
so3 % kill -USR2 2
```

Remplacer -USR2 par le suffixe du signal à envoyer (-USR1, -INT, …). Dans so3, le PID d'une application lancée depuis le shell est affiché entre crochet lorsque celle-ci est exécutée.

Etape 3 – Mise en place d'un serveur TCP

Cette étape propose d'implémenter un client-serveur TCP simple. Le client pourra envoyer différents types de messages, en réaction à des signaux provenant du shell. Le serveur devra répondre, si nécessaire, à ces messages. Pour cela, deux instances de QEMU seront à lancer en parallèle via le script **st.**

Les fichiers à modifier sont :

- **usr/src/socket_server.c :** Fichier du serveur.
- **usr/src/socket_client.c :** Fichier du client
- **usr/src/socket_sye.h :** Header commun (pas forcément besoin de le modifier, mais possible). Des « #define » sont disponibles pour votre utilisation.

a) Ajouter le support de base pour établir une connexion TCP entre le client et le serveur. L'adresse du serveur et son port seront passés en paramètres à l'application **socket_client.** Le port du serveur **doit être fixé** à **5000.**

b) Ajouter un moyen de réagir aux signaux et d'envoyer un type de message par signal :

- **SIGUSR1** envoie les chaînes de charactères « Bonjour » et « Aurevoir » en alternance. Le serveur doit répondre : « Bonjour/Aurevoir client <IP_DU_CLIENT> »

- **SIGUSR2** envoie la chaîne de caractère « Compteur ». Le serveur doit répondre avec une valeur de compteur qu'il tiendra à jour et qu'il incrémente à chaque fois qu'il reçoit cette chaîne. Le format de la chaine est libre, tant que la valeur du compteur y est visible

- **SIGINT** envoie la chaîne de charactères « Quitter » et termine son exécution. Le serveur ne renvoie rien et arrête son exécution**. Dans les deux cas, les applications doivent nettoyer leurs ressources !**

Dans tous les cas, le serveur doit afficher la chaîne reçue, avant l'envoi de sa réponse et le client doit afficher la réponse, s'il y en a une.

Informations utiles :

⇒ Nous vous proposons de gérer l'interaction handlers/boucle-principale via une variable « de routage » globale. Celle-ci sera modifiée dans les handlers et l'envoi/réception des données sera fait dans la boucle principale. Le choix du message à envoyer se fait en lisant la variable de routage, dans une boucle.
⇒ Après le démarrage de *so3*, il faut **attendre quelques secondes** pour acquérir une adresse IP avant de lancer une application. Celle-ci est affichée lorsqu'elle est disponible.
⇒ Voici un exemple de la commande à lancer pour le client, en assumant un serveur à l'adresse **192.168.53.12** (va changer) au port **5000** (ne changera pas) :

```
so3 % socket_client 192.168.53.12 5000 &
```

Comme dit précédemment, le symbole **&** permet de lancer une application en *background*.
Dans notre cas, nous utilisons cela pour pouvoir envoyer des signaux à **socket_client** depuis le shell.

Voici un exemple de l'interaction client-serveur qui est demandée :

Côté serveur :

```
so3 % socket_server
Waiting for clients...
Client connected!
The client said: Bonjour
The client said: Aurevoir
The client said: Bonjour
The client said: Compteur
The client said: Compteur
The client said: Compteur
The client said: Quitter
The client asked for a disconnection, now quitting...
```

Côté client:

```
so3 % socket_client 192.168.53.118 5000 &
[2]
so3% Connecting to server at 192.168.53.118:5000... SUCCESS
kill -USR1 2
Handler USR1
so3% Bonjour client 192.168.53.31
kill -USR1 2
so3% Handler USR1
Aurevoir client 192.168.53.31
kill -USR1 2
so3% Handler USR1
Bonjour client 192.168.53.31
kill -USR2 2
so3% Handler USR2
Valeur compteur 1
kill -USR2 2
so3% Handler USR2
Valeur compteur 2
kill -USR2 2
so3% Handler USR2
Valeur compteur 3
kill -INT 2
so3% Handler INT
[2] Terminated
```

⇒ En jaune : Les commandes exécutées depuis le shell.
⇒ En vert : Les messages affichés dans le handlers des signaux.
⇒ En rouge : Les réponses provenant du serveur.
⇒ Sans surbrillance : Messages d'information de l'app client ou du shell.

⇒ On remarque que l'on envoie au PID 2, car c'est celui-ci qui a été affecté au processus du client, visible entre crochet après la commande.