Module d'approfondissement SEEE

Systèmes d'exploitation et environnements d'exécution embarqués

Prof. Daniel Rossier / Magali Fröhlich

Virtualisation embarquée

Labo reptar4 *(v2017.1.5)*

Ce laboratoire permet de découvrir un environnement de virtualisation embarquée dans l'environnement émulé et d'y déployer les composants développés précédemment. Le *framework* de virtualisation *EmbeddedXEN* - utilisé dans le cadre de ce laboratoire - est le fruit de plusieurs projets de recherche au sein de l'institut REDS depuis 2007. *EmbeddedXEN* permet de déployer deux OS invités (appelés *Dom0* et *DomU*) sur différentes plates-formes équipées d'un microcontrôleur de la famille ARM.

Ce laboratoire est à réaliser dans l'environnement émulé uniquement.

Consignes de laboratoire

- Ce laboratoire est à faire en binôme.
- Un rapport détaillé doit être élaboré par groupe et sera évalué.

Prologue **- Mise à jour du *workspace***

a) Dans le dossier *drivers*, ouvrez le fichier *reptar_sp6_buttons.c*
b) Réactivez les lignes qui avaient été commentées au début du laboratoire 3, en remplaçant `#if 0` par `#if 1`, comme suit :

```
#if 1

#include <xen-guest/io/kbdif.h>
#include <xen-guest/console.h>

extern int omapfb_xen_switch_domain(domid_t dom);

#endif /* 1 */
```

c) Pour ce laboratoire, il est nécessaire d'installer le composant suivant :

```
$ sudo apt-get install elfutils
```

d) Depuis votre *workspace*, exécutez le script *./buildex* . Cette étape prend du temps.
e) Nous avons besoin d'une image contenant les *rootfs* des deux OS invités. Nous vous avons envoyé un e-mail contenant l'adresse à laquelle le fichier peut être téléchargé.
f) Placez le fichier *reptar4-sdcard.img* dans le répertoire *filesystem* de votre *workspace*.
g) Dans votre *workspace*, vous trouverez l'image du noyau Linux que nous avons utilisé lors des laboratoires précédents ; elle s'appelle *uImage*. Renommez-la en *uImage.old* :

```
$ mv uImage uImage.old
```

h) Créez un lien symbolique vers la nouvelle image qui a été générée à l'étape 0 et qui comporte

l'hyperviseur *EmbeddedXEN* et les deux OS invités :

```
$ cd seee_student
$ ln -sf embeddedxen/uImage.embeddedxen.reptar uImage
```

Etape no. 1 – Test de l'environnement *EmbeddedXEN*

Tous les fichiers relatifs à l'environnement de virtualisation *EmbeddedXEN*, à l'exception du système de fichiers (*rootfs*) préparé précédemment, se trouvent dans le répertoire *embeddedxen* situé dans votre *workspace*.

a) Le paramètre *ex* peut être utilisé avec le script *deploy* pour le déploiement des fichiers dans le *rootfs* utilisé avec EmbeddedXEN, comme suit :

```
$ ./deploy ex
```

Déployez les fichiers dans le nouveau *rootfs*.

b) Démarrez le système en lançant le script *./stex* .
c) Vous pouvez observer le démarrage de l'hyperviseur, suivi du premier OS invité (*Dom0*), puis du second OS invité (*DomU*). Les messages seront "mélangés" par la suite car les deux OS accèderont simultanément à l'UART de la console. La commutation de la console entre *Dom0* et *DomU*, puis entre *DomU* et la console de l'hyperviseur, puis entre cette dernière et *Dom0* de nouveau, s'effectue à l'aide de **trois combinaisons *Ctrl-A*** successives (3 × *Ctrl-A*). Connectez-vous dans les deux domaines (login et mot de passe : *root*).

- On remarquera la présence du script *qtrun* dans le répertoire *home*, dans les deux domaines. Ce script lance l'application *QTextEdit*.
- L'écran LCD de la plate-forme *Reptar* est visible dans la fenêtre graphique QEMU standard. Cette fenêtre sera partagée entre les deux OS invités.

Etape no. 2 – Déploiement du *driver sp6* dans *Dom0*

Dans cette étape, vous allez déployer votre *driver sp6* dans *Dom0*. Pour cela, il faut d'abord le recompiler pour le noyau correspondant à ce domaine.

a) Modifiez le *Makefile* dans le répertoire *drivers* afin que la compilation du module s'effectue **avec le noyau *linux-3.0-reptar-dom0*** présent dans *EmbeddedXEN*, et non plus avec le noyau *linux-3.0-reptar*.
b) Compilez votre module.
c) Déployez votre module dans le *rootfs* de *Dom0* avec *./deploy ex* .
d) Démarrez QEMU.
e) Dans *Dom0*, restez dans le répertoire */root*, chargez le module puis lancez l'application *./qtrun* :

```
# cd
# insmod /sp6.ko
# ./qtrun
```

(Un appel à la commande *cd* sans paramètre retourne dans le répertoire *home* de l'utilisateur. Le répertoire *home* de *root* est */root* .)

Etape no. 3 – Interactions entre *Dom0* et *DomU* sur l'interface *input* para-virtualisée

Dom0 peut accéder directement aux périphériques et les piloter, mais *DomU* ne le peut pas. Ainsi, nous avons besoin de créer une interface virtuelle dans *DomU* pour lui permettre d'accéder aux ressources du périphérique souhaité, en assurant une bonne coordination des accès. Les drivers sont découpés en deux parties : le *backend* dans *Dom0* et le *frontend* dans *DomU*.

Lors de cet exercice, nous allons permettre à l'application Qt dans *DomU* de récupérer les événements *input* (*input events*) provenant de la FPGA.

a) Identifiez les différents blocs qui interviennent lors de la propagation de l'*input event* depuis la FPGA vers *DomU*. Etablissez un schéma faisant intervenir différents blocs : le *backend input*, le driver FPGA développé par vos soins, le *frontend input*, les *input subsystems* de *Dom0* et *DomU*… (Vous trouverez le *backend* dans *linux-3.0-reptar-dom0/drivers/input/xen-kbdback* et le *frontend* dans *linux-3.4.6-domU/drivers/input/misc/xen-kbdfront.c*).

b) Modifiez votre *driver* afin que celui-ci s'enregistre auprès du *backend input* à l'aide de la fonction ***xenvkbd_inputdev_register( )*** (quelques indications sont fournies ci-après).

Par défaut, le *focus* du *framebuffer* est attribué à *Dom0*. Adaptez le code de votre *driver* afin que l'appui sur le bouton n°8 (en haut à droite) place le *focus* sur *DomU* et l'appui sur le bouton n°6 (en haut à gauche) place le *focus* sur *Dom0*.

Effectuez différents tests sur l'application Qt dans *Dom0* et *DomU*.

Indications

- L'enregistrement auprès du *backend input* s'effectue à l'aide de la fonction ***xenkbd_inputdev_register( )***.

  ➢ Le type de périphérique est indiqué avec un *enum* :
  ```
  enum inputdev_type { KEYBOARD, MOUSE, TOUCHSCREEN, GPIOKEY }
  ```

  ➢ La signature de la fonction est la suivante :
  ```
  void xenvkbd_inputdev_register(struct input_dev *dev, enum inputdev_type t)
  ```

- La fonction `omapfb_xen_switch_domain(<domID>)` permet d'attribuer le *focus* du *framebuffer* à *Dom0* (`domID = 0`) ou à *DomU* (`domID = 1`).

- Le bouton n°8 est associé à la constante *KEY_BACKSPACE* et le bouton n°6 à la constante *KEY_ESC*.

→ Vous devez maintenant être en mesure d'expliquer le cheminement de l'événement *input* depuis l'appui sur un bouton, jusqu'au traitement applicatif.

Etape no. 4 – Para-virtualisation de l'interface LED

Nous allons maintenant nous intéresser à l'interface LED. Au terme de cet exercice, nous aurons écrit une application dans *DomU*, qui interceptera les evénements *input* des boutons et réagira en allumant une LED.

a) Tout d'abord, il est nécessaire de fournir une interface virtuelle LED dans *DomU*. Ouvrez le fichier *linux-3.4.6-domU/drivers/leds/xen-ledfront.c* . Complétez le fichier pour créer les entrées nécessaires dans */sys/class/leds* et pour que les requêtes d'allumage et d'extinction des LEDs soient propagées vers *Dom0*.
b) Ouvrez le fichier *linux-3.0-reptar-dom0/drivers/leds/xen-ledback.c* . Complétez le fichier pour que les requêtes d'allumage et d'extinction des LEDs provenant de *DomU* soient traitées et effectuent l'action désirée.

→ Vous devez maintenant être en mesure d'expliquer le cheminement depuis l'écriture de la valeur dans l'entrée *sysfs* d'une LED dans *DomU*, jusqu'à la mise à jour du registre FPGA dédié aux LEDs par *Dom0*.

c) Ecrivez une application qui doit respecter les critères suivants :

- Un appui sur le bouton n°2 (gauche) allume la LED 2, un appui sur le bouton n°5 (centre) allume la LED 1 et un appui sur le bouton n°4 (droite) allume la LED 0. Un appui sur le bouton n°3 (bas) éteint les LEDs.
- L'application fait du *polling* sur */dev/input/event0* pour détecter l'appui sur un bouton. Elle doit décoder la valeur de la touche pour identifier le bouton.
- L'application utilise les entrées *sysfs* liées au sous-système LED.

Testez sur l'émulateur.

Indications

- Pour créer les différentes entrées dans */sys/class/leds* , inspirez-vous de ce qui a été fait dans le driver *sp6* dans *Dom0*.
- Dans le *frontend*, la fonction ***send_led_request( )*** propage une commande depuis *DomU* vers *Dom0* :
  - Le premier paramètre (`id`) est un numéro qui vous permet d'identifier la LED (nous vous laissons choisir les numéros à votre guise).
  - Le second paramètre (`brightness`) est la valeur d'intensité, gérée par le sous-système LED, qu'il faut transmettre à *Dom0*.
- Dans le *backend*, la fonction ***receive_led_request( )*** est appelée lors de l'arrivée d'une commande depuis *DomU*. Ses paramètres correspondent à ceux de ***send_led_request( )*** . Elle est appelée dans un contexte d'interruption.