TP3 MI11 Linux Xenomai

Semestre printemps 2022

Guillaume Sanahuja – guillaume.sanahuja@hds.utc.fr

Table des matières

Préambule

Pour ce TP, vous devez réutiliser la machine virtuelle de la séance précédente.

Un compte rendu de TP au format PDF est à rendre sur le moodle de l'UV. Il vous est demandé d'y écrire les réponses aux questions figurant dans les différents exercices. Également, vous pouvez compléter le compte rendu avec toute information que vous jugerez utile, par exemple les manipulations que vous avez réalisées pendant la séance, les résultats que vous avez observés, etc.

Travail préalable à la séance de TP

Relire les cours sur Linux embarqué et sur Linux temps réel disponibles sur le moodle MI11. Regarder la documentation de Xenomai :
https://xenomai.org/documentation/xenomai-3/html/xeno3prm/index.html
Et lire notamment la documentation sur l'API alchemy, que vous utiliserez pour ce TP :
https://xenomai.org/documentation/xenomai-3/html/xeno3prm/group__alchemy.html
Ainsi que la documentatin sur l'API RTDM :
https://xenomai.org/documentation/xenomai-3/html/xeno3prm/group__rtdm__profiles.html

Le noyau linux généré au TP1 Xenomai ne possède pas toutes les fonctionnalités nécessaires à ce TP. Vous devez donc récupérer sur le moodle de l'UV un nouveau noyau à copier dans */tftpboot*.

Exercice 1 : Communication Xenomai/Linux

Cet exercice va vous permettre d'échanger des messages entre des tâches temps réel et non temps réel. Vous utiliserez pour cela les *Message pipe services* de l'API Alchemy de Xenomai (*rt_pipe_create/delete/read/write*), et les fonctions de gestion de fichiers coté Linux (*open/close/read/write*). Ce système permet aux tâches temps réel d'accéder à des ressources non temps réel (affichage, réseau, périphériques, etc).

Ecrivez un programme créant un *pipe* Xenomai et une tâche temps réel. Ajoutez une écriture périodique dans le *pipe* à la tâche temps réel. Le *main* (non temps réel) de son coté doit se charger de la lecture. Envoyez par exemple le message *hello world* de Xenomai vers Linux.

Question 1.1 : Donnez le code du programme et le résultat.

Le *pipe* étant bi-directionnel, ajoutez une tâche temps réel lisant le *pipe* et ajoutez de l'écriture dans le *main*.

Question 1.2 : Donnez le code du programme et le résultat.

Exercice 2 : GPIOs RTDM

Nous allons dans cet exercice créer un driver RTDM pilotant les leds (via les GPIOs), ainsi qu'une application utilisant ce driver.

1. Préparation

Afin de pouvoir compiler le module RTDM, le noyau doit également avoir été configuré et préparé. En effet le module est dépendant du noyau. Pour cela effectuez les opérations suivantes :

- application du patch Xenomai aux sources du noyau

```
cd /opt/mi11/linux-raspberrypi
patch -p1 < /opt/mi11/meta-joypinote/recipes-kernel/linux/files/pre-rpi4-4.19.86-xenomai3-simplerobot.patch
cd /opt/mi11/
wget https://xenomai.org/downloads/xenomai/stable/xenomai-3.1.tar.bz2
tar -xf xenomai-3.1.tar.bz2
./xenomai-3.1/scripts/prepare-kernel.sh --arch=arm --linux=/opt/mi11/linux-raspberrypi/ --ipipe=/opt/mi11/meta-joypinote/recipes-kernel/linux/files/ipipe-core-4.19.82-arm-6-mod-4.49.86.patch.ipipe
```

- préparation des sources du noyau

```
cd /opt/mi11/linux-raspberrypi
. /opt/poky/3.1.16/cortexa7thf-neon-vfpv4/environment-setup-cortexa7t2hf-neon-vfpv4-poky-linux-gnueabi
make clean && make mrproper
```

make joypinote-xenomai_defconfig
make modules_prepare

Récupérez sur le moodle de l'UV l'archive rtdm_gpio.tar.bz2 contenant les fichiers nécessaires au module RTDM, qui seront à compléter.

Lisez le squelette *rtdm_gpio.c* du driver pour comprendre son fonctionnement actuel. Le module à réaliser est très simple et permettra 3 actions commandées par IOCTL : configuration du GPIO en sortie, mise à 0 du GPIO et mise à 1 du GPIO. La fonction *rtdm_printk* permet d'afficher des messages dans les logs du noyau, visibles avec la commande *dmesg*.

Placez vous dans le dossier des source du module et cross compilez-le avec la commande *make*.

Le module compilé s'appelle *rtdm_gpio.ko*. Sur la cible, son chargement s'effectue avec la commande :
insmod rtdm_gpio.ko
et le déchargement :
rmmod rtdm_gpio

Question 2.1 : Donnez le résultat des chargements et déchargements du module (logs noyau et apparition du driver RTDM). Où se trouve le périphérique créé par le module ?

Ecrivez ensuite un programme réalisant l'ouverture et la fermeture du périphérique RTDM (fonctions *open* et *close*) dans une tâche temps réel. Afin de lier le programme avec RTDM, il faut utiliser un *Makefile* spécial, disponible sur le moodle de l'UV (*Makefile RTDM userspace*).

Question 2.2 : Donnez le code du programme et les résultats obtenus.

2. Registres

Tout étant en place, le but est maintenant d'écrire la partie relative aux GPIOs dans le driver. Pour cela il faut s'aider de la documentation *BCM2711 ARM Peripherals* disponible sur le moodle de l'UV.

Commencez par lire la section *1.2. Address map*. L'adressage que vous utiliserez est le *Low Peripheral*. Lisez ensuite le chapitre 5 *General Purpose I/O*. Pour simplifer votre module, seuls les GPIOs 0 à 9 seront pilotables.

Question 2.3 : Quelle est l'adresse de base du registre GPIO que vous devez utiliser?

Question 2.4 : Donnez les opérations à faire sur les registres pour activer le GPIO i ($0 \leq i \leq 9$) en sortie, sans altérer la configuration des autres GPIOs.

Question 2.5 : Donnez les opérations à faire sur les registres pour mettre le GPIO i ($0 \leq i \leq 9$) à 0 et à 1.

Un registre est exprimé en fonction d'une base (REGISTRE_BASE) et d'un offset (OFFSET) ; pour lire la valeur d'un registre dans le driver RTDM il faut effectuer les opérations suivantes :

unsigned long base=(unsigned long)ioremap(REGISTRE_BASE, 4);
*int val=readl((void *)base + OFFSET);*

et pour l'écriture il faut faire les opérations suivantes :

unsigned long base=(unsigned long)ioremap(REGISTRE_BASE, 4);
*writel(val,(void *)base + OFFSET);*

Complétez alors les fonctions *rtgpio_direction_output* et *rtgpio_set_value* du driver.

Question 2.6 : Donnez le code du driver.

3. Clignotement

Complétez maintenant le programme utilisant le périphérique afin de configurer un GPIO (5 ou 6) en sortie, et de faire clignoter la led associée. Pour cela il faut utiliser la fonction *ioctl* :
ioctl(int fd, int request, int gpio_pin)
Les différentes *request* sont définies dans le fichier *rtdm_gpio.h* du driver, que vous devrez inclure.
Attention, les leds sont câblées en logique inverse (*active low*) : la led est allumée lorsque le GPIO associé est à 0 et éteinte lorsque le GPIO est à 1.

Question 2.7 : Donnez le code du programme.

Nous allons améliorer un peu le programme qui fait clignoter les leds. Transformez votre code pour que le clignotement se fasse en fonction d'une fréquence donnée, et d'un rapport cyclique donné. Le rapport cyclique est le ratio entre le temps allumé et la période. Un rapport cyclique de 0 indique une led toujours éteinte, un rapport de 1 indique une led toujours allumée.

Question 2.8 : Donnez le code du programme. Quelle fonctionnalité avez-vous programmée ? Quelle fréquence faut-il au minimum ?

Enfin, utilisez une RT_PIPE pour envoyer depuis le *main* une consigne de fréquence et de rapport cyclique pour une led donnée. Utilisez par exemple *scanf* pour que l'utilisateur puisse changer ces paramètres à volonté.

Question 2.9 : Donnez le code du programme.

4. Robustesse

Afin de tester la robustesse de Xenomai, nous allons forcer un crash du système avec la commande :
echo c > /proc/sysrq-trigger **(à effectuer sur la cible et non sur la VM!!!)**

Lancez donc dans un premier terminal votre programme de clignotement, et dans un second terminal forcez le crash système.

Question 2.10 : Que constatez-vous ? Quels sont les éléments fonctionnant encore et pourquoi ? Quels sont les éléments ne fonctionnant plus et pourquoi ?