

Lab 4

Par Yoan PIDERI, Floriane THOCQUENNE et Raphaël CASIMIR

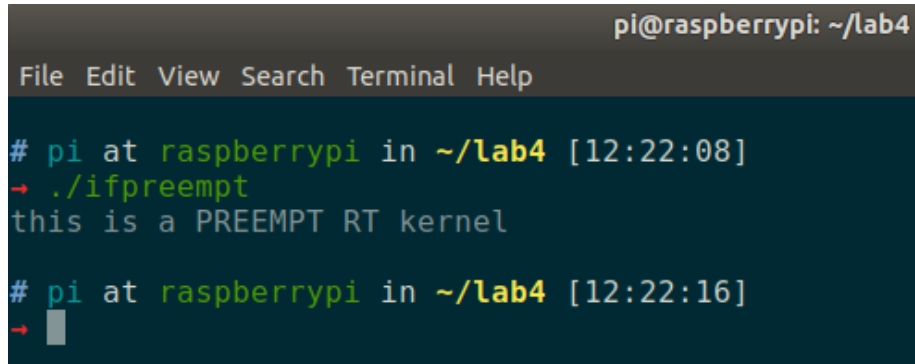
Le but de ce lab est de compiler le noyau linux d'une raspberry pi avec une configuration d'ordonnanceur adaptée pour une utilisation temps-réel.

Cross-compilation et installation d'un noyau temps-réel

1. La première étape est de télécharger la version du noyau qui tourne déjà sur la raspberry pi, pour éviter toute incompatibilité. Pour cela nous avons cloné le repo github du noyau de la raspberry puis nous sommes retourné au commit correspondant à notre noyau en utilisant l'outil "get_hash.sh" fourni ainsi que la commande `git checkout <numero_du_commit>`. De cette manière nous sommes certains d'avoir exactement la même version que celle compilée sur la raspberry.
2. Ensuite nous avons téléchargé et appliqué le patch PREEMPT-RT correspondant à notre version de noyau. La commande "patch" est un outil pratique permettant de ne télécharger que la différence entre notre version des fichiers source et la version que nous voulons atteindre.
3. Nous avons ensuite utilisé modprobe et zcat sur la Pi pour générer un fichier .config qui contient la configuration noyau actuelle. Cette configuration a été importée sur la machine de compilation, puis modifiée en utilisant `menuconfig` pour sélectionner la nouvelle configuration d'ordonnanceur activable grâce au patch.
4. Pour pouvoir faire la cross-compilation, c'est à dire compiler le noyau de la Pi sur un ordinateur avec une architecture différente, il faut exporter les variables `ARCH=arm` et `CROSS_COMPILE=<emplacement_du_fichier_tools_fourni>/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-`. Cela précise au compilateur l'architecture cible et les outils de cross-compilation à utiliser.
5. En utilisant `make zImage -j<nombre de threads à utiliser>` puis `make modules -j<X>`, `make dtbs` et enfin `make modules_install` on génère une version compilée du noyau, puis des modules, puis le "blob d'arborescence matérielle", qui permet entre autres au système de gérer de manière fiable les accès concurrents aux ressources matérielles.
6. Ensuite nous générons une image du noyau en format img que nous plaçons dans notre arborescence de travail.

```
./scripts/mkknling ./arch/arm/boot/zImage $INSTALL_MOD_PATH/boot/kernel7.img
```

7. Ensuite nous avons copié les device tree et overlays générés précédemment dans le fichier boot crée à la racine de notre architecture de travail puis avons compressé et transféré le tout sur la raspberry.
8. Finalement, nous avons pu fusionner l'arborescence générée avec celle de la raspberry. Après un redémarrage, on peut en effet voir que le noyau est en mode préemptible.



```
pi@raspberrypi: ~/lab4
File Edit View Search Terminal Help

# pi at raspberrypi in ~/lab4 [12:22:08]
→ ./ifpreempt
this is a PREEMPT RT kernel

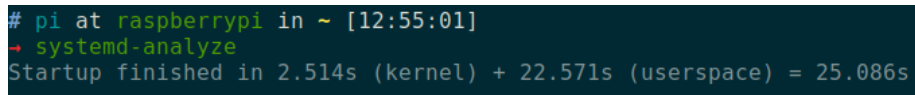
# pi at raspberrypi in ~/lab4 [12:22:16]
→
```

Figure 1: La sortie de l'outil fourni "ifpreempt"

9. La commande chrt permet de lancer un process avec une politique d'ordonnancement spécifique. Par exemple `sudo chrt -f -p 20 23705` lance le processus 23705 avec la politique `SCHED_FIFO` et la priorité 20.

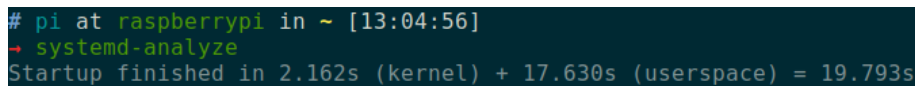
Optimisation du temps de démarrage

En utilisant une adresse ip statique (modifier `/etc/networks/interfaces`), on obtient un temps total de démarrage réduit d'environ 25%.



```
# pi at raspberrypi in ~ [12:55:01]
→ systemd-analyze
Startup finished in 2.514s (kernel) + 22.571s (userspace) = 25.086s
```

Figure 2: Avant, utilisation de DHCP



```
# pi at raspberrypi in ~ [13:04:56]
→ systemd-analyze
Startup finished in 2.162s (kernel) + 17.630s (userspace) = 19.793s
```

Figure 3: Après, utilisation d'une adresse statique

Création d'un module kernel

Voici le module kernel simple que nous avons écrit :

```
// Début module
#include <linux/module.h>    /* Needed by all modules */
#include <linux/kernel.h>    /* Needed for KERN_INFO */

int init_module(void)
{
    printk(KERN_EMERG "Hello world\n");

    return 0; // If it doesn't return 0, the module failed to start
}

void cleanup_module(void)
{
    printk(KERN_EMERG "Goodbye world\n");
}

MODULE_LICENSE("MIT");
MODULE_DESCRIPTION("Gooblox generator. Generates Goobloxes of the finest quality.");
MODULE_AUTHOR("Jarjar Binks");
// Fin module
```

Note: les fonctions `init_module` et `cleanup_module` peuvent en réalité être renommées, à condition de préciser leur nom dans les fonctions suivantes :

```
module_init(init_function);
module_exit(cleanup_function);
```

et d'inclure ces indications compilateur après la définition desdites fonctions, dans le fichier `.c`, en dehors de toute autre fonction.

On peut remarquer l'utilisation de la priorité `KERN_EMERG` pour les messages. Il s'agit de la priorité de messages la plus forte, et c'est la seule qui a permis d'afficher un message dans le terminal lors du chargement / déchargement du module. Cependant les messages de priorité inférieure s'ajoutaient bien au fichier `/var/log/messages`.

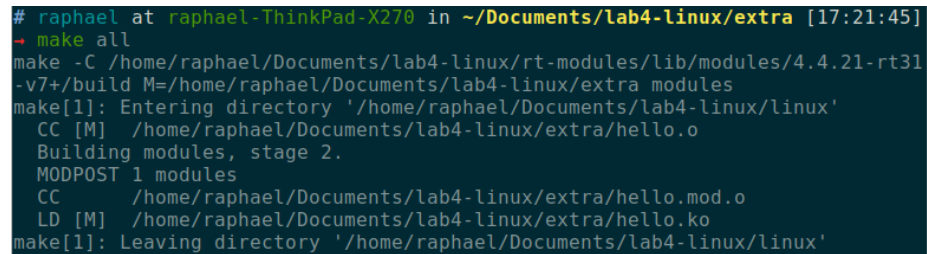
Voici le fichier makefile associé au module :

```
# Début makefile
obj-m += hello.o

all:
    make -C $(INSTALL_MOD_PATH)/lib/modules/4.4.21-rt31-v7+/build M=$(PWD) modules

clean:
    make -C $(INSTALL_MOD_PATH)/lib/modules/4.4.21-rt31-v7+/build M=$(PWD) clean
# Fin makefile
```

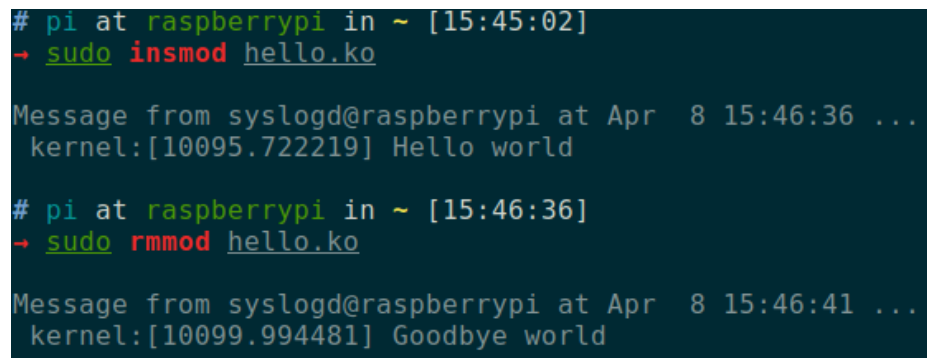
Après compilation avec `make all` on obtient entre autres un fichier `.ko` :



```
# raphael at raphael-ThinkPad-X270 in ~/Documents/lab4-linux/extra [17:21:45]
→ make all
make -C /home/raphael/Documents/lab4-linux/rt-modules/lib/modules/4.4.21-rt31-v7+/build M=/home/raphael/Documents/lab4-linux/extra modules
make[1]: Entering directory '/home/raphael/Documents/lab4-linux/linux'
CC [M] /home/raphael/Documents/lab4-linux/extra/hello.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/raphael/Documents/lab4-linux/extra/hello.mod.o
LD [M] /home/raphael/Documents/lab4-linux/extra/hello.ko
make[1]: Leaving directory '/home/raphael/Documents/lab4-linux/linux'
```

Figure 4: Compilation du module “hello”

La sortie texte du module se fait bien comme prévu.



```
# pi at raspberrypi in ~ [15:45:02]
→ sudo insmod hello.ko

Message from syslogd@raspberrypi at Apr  8 15:46:36 ...
kernel:[10095.722219] Hello world

# pi at raspberrypi in ~ [15:46:36]
→ sudo rmmod hello.ko

Message from syslogd@raspberrypi at Apr  8 15:46:41 ...
kernel:[10099.994481] Goodbye world
```

Figure 5: Sortie après chargement et déchargement du nouveau module

Remerciements

Le site The Linux Documentation Project nous a été d’une aide précieuse quand à la rédaction et compilation de notre module.