

CONSTRUCTION DE SYSTÈMES  
EMBARQUÉS SOUS LINUX  
**Rapport de laboratoire**  
Master HES-SO

Émilie GSPONER, Grégory EMERY

31 octobre 2015  
version 1.0

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Environnement Linux Embarqué</b>	<b>3</b>
2.1	Objectifs . . . . .	3
2.2	Informations pratiques . . . . .	3
2.3	Gravure de la carte SD . . . . .	3
2.4	Mise en place de l'infrastructure . . . . .	3
2.4.1	Configuration de la machine virtuelle . . . . .	4
2.4.2	Accès ssh sans mot de passe . . . . .	6
2.4.3	Création de l'espace de travail . . . . .	7
2.5	Debugging de l'application . . . . .	8
2.6	Test des différents exemples proposés . . . . .	12
2.6.1	Fibonacci . . . . .	12
2.6.2	Tracing . . . . .	13
2.6.3	Core dumps . . . . .	13
2.6.4	Backtrace . . . . .	14
2.6.5	System calls . . . . .	14
2.6.6	Memory leaks . . . . .	15
2.7	Mise en production de l'ODROID-XU3 . . . . .	15
2.8	Réponse aux questions . . . . .	16
<b>3</b>	<b>Travaux pratiques 1</b>	<b>17</b>
3.1	Gestion de la mémoire, bibliothèques et fonctions utiles . . . . .	17
3.1.1	Exercice 4 . . . . .	17
3.1.2	Exercice 5 . . . . .	17
3.2	Accès aux entrées/sorties . . . . .	17
3.2.1	Exercice 6 . . . . .	17
3.3	Gestion des interruptions . . . . .	19
3.3.1	Exercice 9 . . . . .	19

## 1 Introduction

Ce rapport présente les résultats obtenus tout au long des travaux pratiques fournis durant le cours de CSEL1, construction de systèmes embarqués sous Linux. Le document est structuré en sections, représentant les séries d'exercices données, en sous-sections présentant les thèmes proposés pour les travaux et en sous-sous-sections pour les réponses à chacune des questions posées dans le document.

Ce cours est effectué avec la cible Odroid XU3<sup>1</sup> et U-Boot<sup>2</sup> dans le cadre du cours de Master HES-SO en systèmes embarqués, orientation TIN et TIC.

1. Lien : [http://www.hardkernel.com/main/products/prdt\\_info.php?g\\_code=G140448267127](http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127)

2. Lien : <http://www.denx.de/wiki/U-Boot>

## 2 Environnement Linux Embarqué

### 2.1 Objectifs

Ce travail pratique vise les objectifs suivants :

1. Mise en œuvre d'un système embarqué sous Linux
2. Mise en oeuvre de l'environnement de développement de systèmes embarqués sous Linux
3. Debugging d'applications sous Linux embarqué
4. Mise en production d'un système embarqué sous Linux

### 2.2 Informations pratiques

Nous avons choisi l'option de travailler directement avec la machine virtuelle fournie.

### 2.3 Gravure de la carte SD

Avant de pouvoir graver la carte, il faut trouver le nom du périphérique auquel elle est attachée. Il peut être obtenu avec la commande suivante :

```
1 $ mount
2 ...
3 /dev/sd2 on /media/lmi/usrfs type ext4 (rw,nosuid,nodev,uhelper=udisks2)
4 /dev/sd1 on /media/lmi/5a13f590-5792-413e-ba62-403debd56a5 type ext4 (rw,
   nosuid,nodev,uhelper=udisks2)
5 ...
```

La commande va lister tous les périphériques connectés, dans notre cas, la carte SD se nomme lmi et est liée à /dev/sdb2 et /dev/sdb1.

Un script a ensuite été écrit, regroupant les différentes commandes à exécuter pour la gravure de la carte.

Le contenu du script est visible dans les fichiers remis avec le rapport sous */EnvLinuxEmbarque/flasheMMC.sh*.

Le plus simple est de placer le script directement dans le workspace CSEL. Il s'exécute à l'aide de la commande suivante :

```
1 ./flasheMMC.sh
```

Il va aller détacher les volumes attachés à la carte SD, créer la table de partition et graver les différents firmwares et images.

Une fois la carte gravée et insérée dans la cible, le ventilateur se met à tourner si tout s'est bien passé. Si rien ne se passe, il faut également contrôler que le switch de l'Odroid est en position pour booter sur la carte SD.

### 2.4 Mise en place de l'infrastructure

La cible ODROID-XU3 doit avoir l'adresse IP 192.168.0.11 et la machine hôte l'IP 192.168.0.4.

### 2.4.1 Configuration de la machine virtuelle

Pour associer la carte réseau de l'ordinateur à la machine virtuelle, il faut suivre les étapes ci-dessous :

1. Éteindre la vm, aller dans *edit->Virtual Network Editor...->change settings*
2. Dans la fenêtre des réseaux, changer la configuration en bridged to : Contrôleur PCIe (propre à l'ordinateur), carte réseau de l'ordinateur

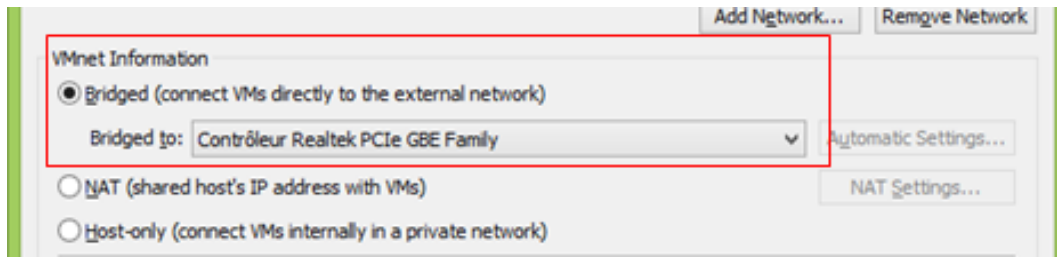


FIGURE 1 – Configuration de la carte réseau

Puis dans les settings de la VM, il faut aller changer le réseau pour utiliser celui que l'on vient de configurer.

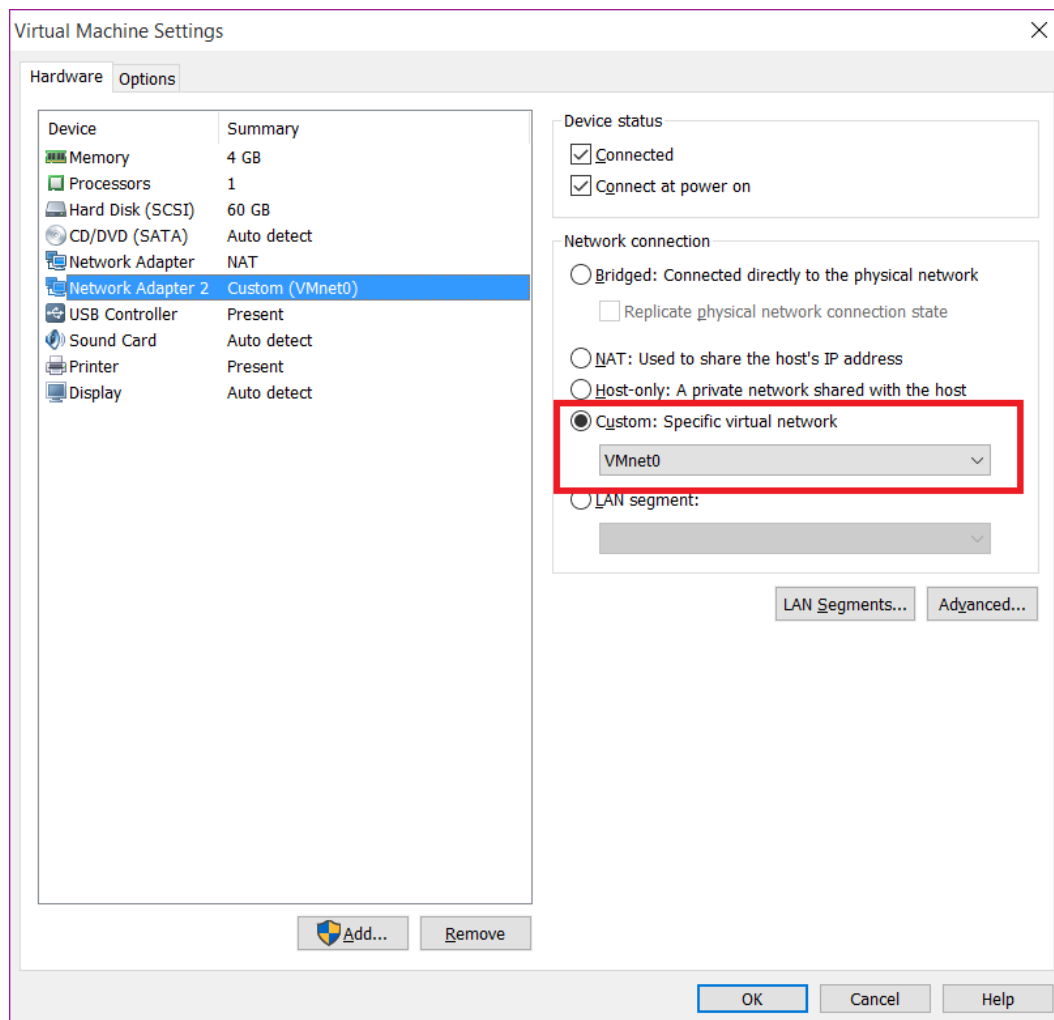


FIGURE 2 – Association de la carte réseau à la machine virtuelle

La dernière étape est d'activer le réseau dans la machine virtuelle (icône réseau -> enabling networking).

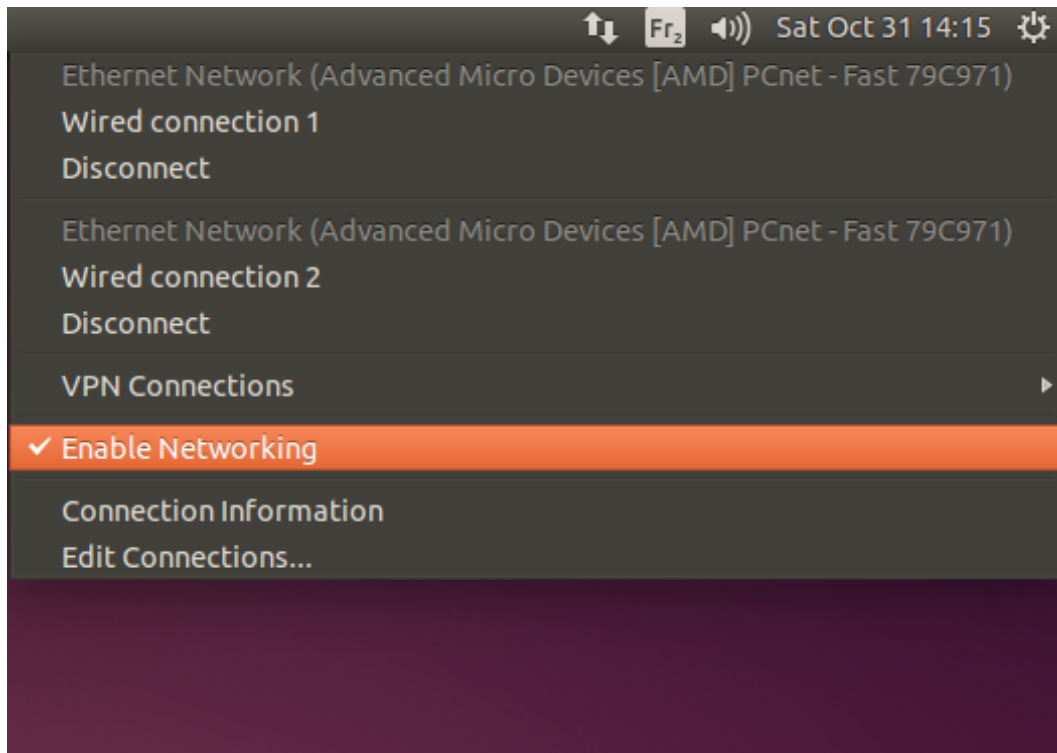


FIGURE 3 – Activation du réseau

### 2.4.2 Accès ssh sans mot de passe

Il faut aller modifier le fichier `/etc/ssh/sshd_config` sur la cible pour autoriser l'accès sans mot de passe. Pour accéder au fichier, il faut entrer sur la cible par la connexion série :

```
1 $ sudo minicom
2
3 Welcome to minicom 2.7
4
5 OPTIONS: I18n
6 Compiled on Jan  1 2014, 17:13:22.
7 Port /dev/ttyUSB0, 13:10:43
8
9 Press CTRL-A Z for help on special keys
10
11
12 Welcome to Hardkernel ODROID_XU3 board
13 odroidxu3 login: root
14 #
```

On peut ensuite rechercher le fichier et l'éditer avec vi ou vim pour modifier "PermitEmptyPassword yes". Une fois la configuration faite, il faut redémarrer la cible :

```
1 # reboot
```

Normalement, on devrait avoir accès à la cible en ssh en entrant la commande suivante dans la machine hôte :

```
1 lmi@cse11:~$ ssh root@192.168.0.11
2 #
```

Pour valider la connexion Ethernet/IP, on peut également arrêter la cible dans son U-boot en tapant la touche "carriage return" et faire un ping de la machine hôte :

```
1 lmi@cse11:~$ sudo minicom
2 ...
3 # reboot
4 ...
5 Press 'Enter' or 'Space' to stop autoboot:  0
6
7 ODROIDXU3> usb start
8 (Re)start USB...
9 USB:   Register 1313 NbrPorts 3
10 USB EHCI 1.00
11 scanning bus for devices... The request port(2) is not configured
12 The request port(2) is not configured
13 4 USB Device(s) found
14 scanning bus for storage devices... 0 Storage Device(s) found
15 scanning bus for ethernet devices... 1 Ethernet Device(s) found
16
17 ODROIDXU3> ping 192.168.0.4
18 Waiting for Ethernet connection... done.
19 Using sms0 device
20 host 192.168.0.4 is alive
21
22 ODROIDXU3> run nfsboot
```

Si la machine hôte répond au ping, tout a bien été configuré.

### 2.4.3 Création de l'espace de travail

Le but est de configurer le noyau Linux afin d'attacher un usrfs sous ext4 depuis la carte SD. En d'autres termes, partager un espace de travail entre la cible et l'hôte. Pour cela, il faut accéder à la cible via le port série ou par ssh et de taper les commandes indiquées dans la donnée.

```
1 # mkdir /usr/workspace
2 # vi /etc/fstab
3 # /etc/fstab: static file system information.
4 #
5 # <file system> <mount pt>      <type>    <options>          <dump> <pass>
6 /dev/root      /                ext2       rw,noauto           0       1
7 proc           /proc            proc       defaults             0       0
8 devpts        /dev/pts         devpts     defaults,gid=5,mode=620 0       0
9 tmpfs         /dev/shm         tmpfs      mode=0777            0       0
10 tmpfs         /tmp             tmpfs      mode=1777            0       0
11 sysfs         /sys             sysfs      defaults             0       0
```

```
12 192.168.0.4:/home/lmi/workspace /usr/workspace nfs      hard,intr,nolock
13 # mount -a
```

Pour contrôler que le répertoire est bien partagé avec la cible, on peut y entrer la commande `mount` et normalement on voit le répertoire partagé.

```
1 # mount
2 ...
3 192.168.0.4:/home/lmi/workspace on /usr/workspace type nfs (rw,relatime,vers=3,
   rsize=131072,wsiz=131072,namlen=255,hard,nolock,proto=tcp,timeo=600,
   retrans=2,sec=sys,mountaddr=192.168.0.4,mountvers=3,mountproto=tcp,
   local_lock=all,addr=192.168.0.4)
```

## 2.5 Debugging de l'application

Cette section présente les différentes étapes à réaliser pour configurer Eclipse pour qu'il utilise une connexion ssh entre l'hôte et la cible. Pour cela, il faut commencer par charger un projet dans Eclipse :

1. File -> Import... (si le projet existe déjà)
2. C/C++ -> Existing Code as Makefile Project (configure pour utiliser le Makefile du projet et non un de Eclipse)
3. Il suffit ensuite de définir le nom du projet et le path jusqu'au code source

Une fois le projet importé dans l'espace de travail, il faut configurer le debugger :



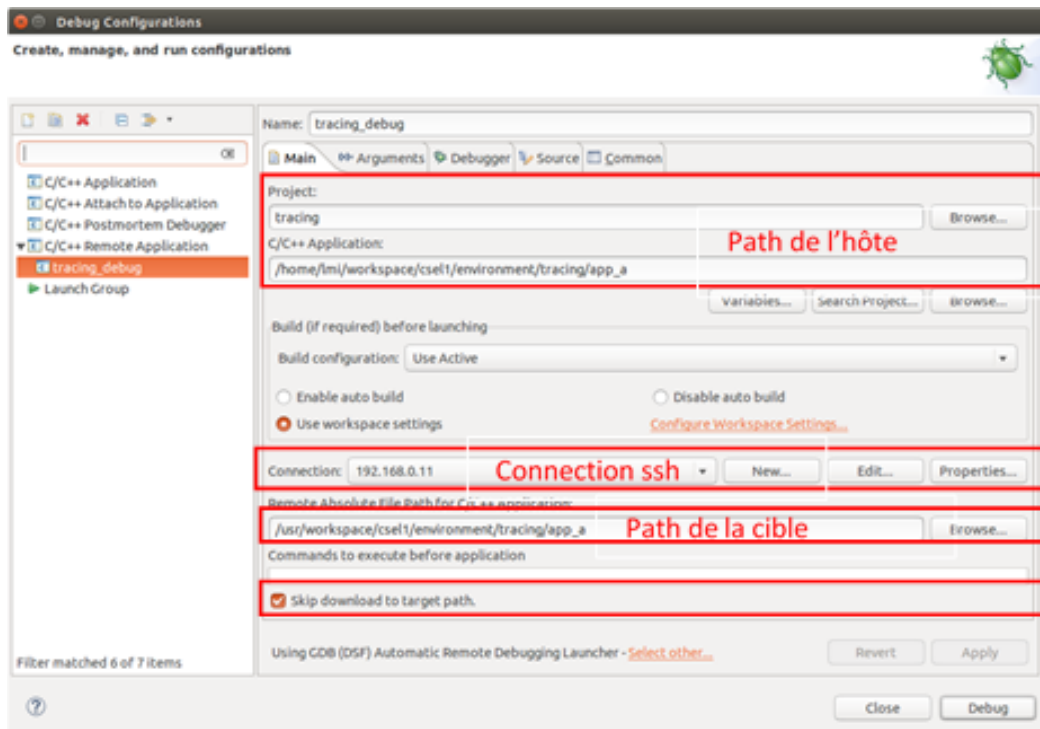


FIGURE 4 – Configuration du debugger

Pour pouvoir entrer le path de la cible, il faut impérativement que la connexion ssh soit configurée comme ci-dessous :

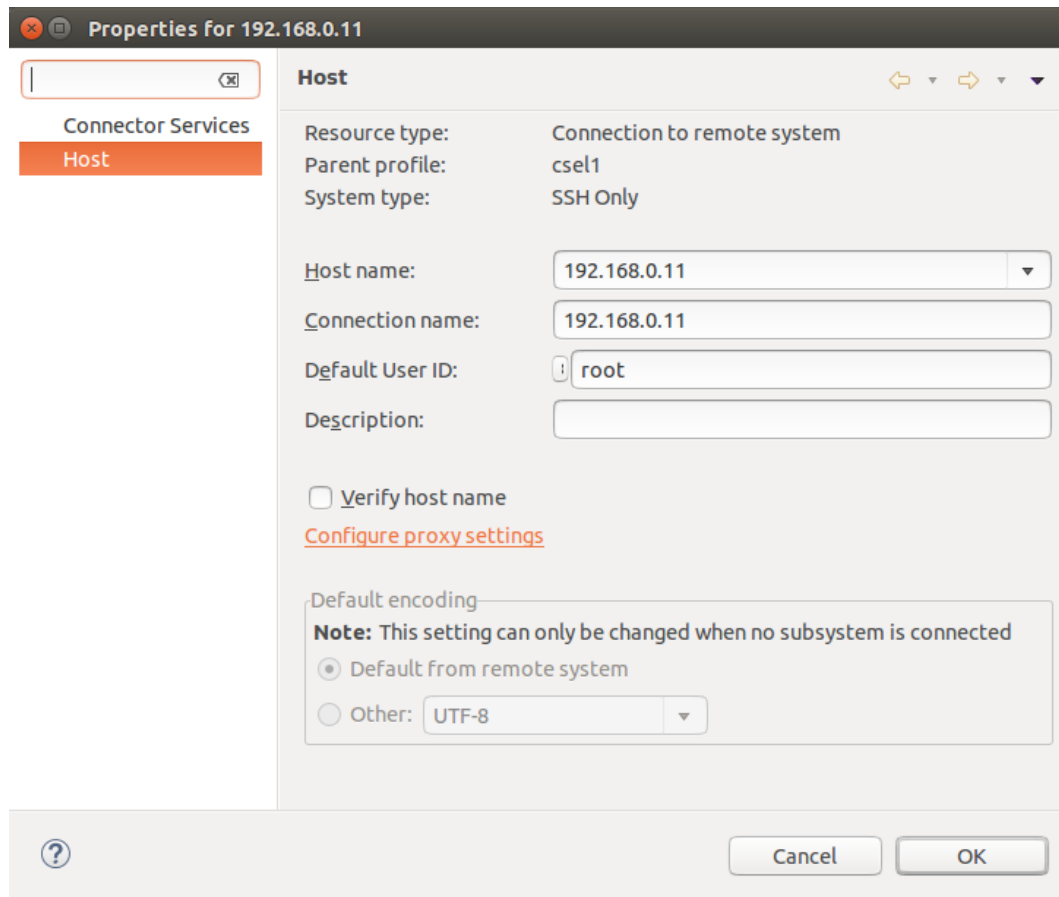


FIGURE 5 – Configuration de l'accès ssh

Il faut encore aller configurer le debugger gdb :

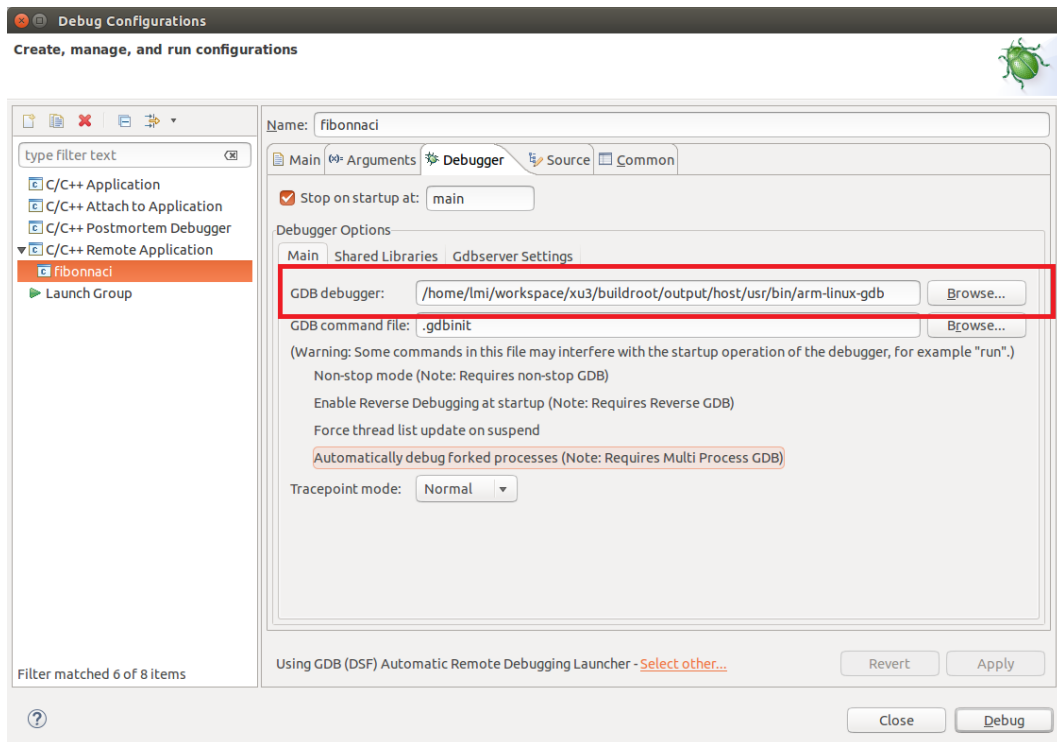


FIGURE 6 – Configuration du server gdb

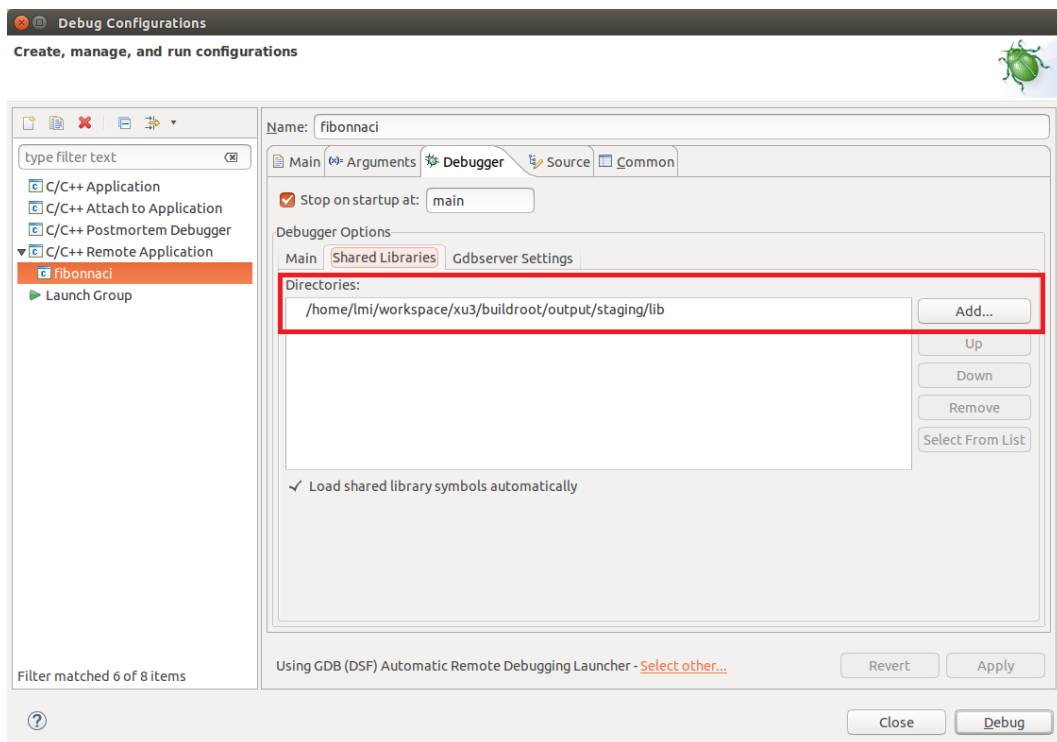


FIGURE 7 – Configuration des shared library

Avec cette configuration, on peut ensuite debugger pas à pas les exercices d'exemples. Voici un exemple avec fibonacci :

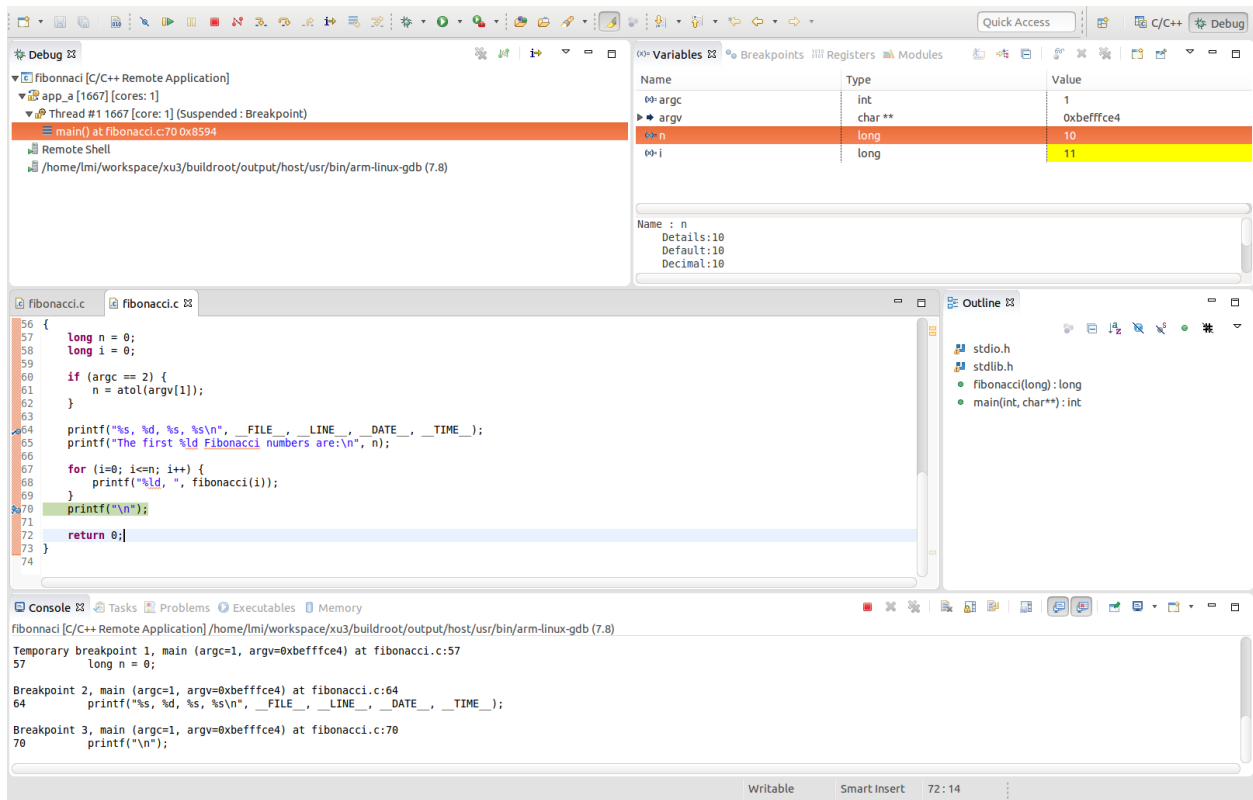


FIGURE 8 – Debug de l'exemple Fibonacci

## 2.6 Test des différents exemples proposés

Pour la suite du rapport, les symboles suivant sont définis :

1. \$ : commande sur la machine hôte
2. # : commande sur la cible
3. > : commande sur la cible en NFS

### 2.6.1 Fibonacci

```

1 lmi@cse11:~/workspace/cse11/environment/samples/fibonacci$ make clean all
2
3 lmi@cse11:~$ ssh root@192.168.0.11
4 # cd ../usr/workspace/cse11/environment/samples/fibonacci/
5 # ./app_a 10
6 fibonacci.c, 64, Oct 31 2015, 15:27:54
7 The first 10 Fibonacci numbers are:
8 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,

```

## 2.6.2 Tracing

Avec la trace active

```

1 lmi@cse11:~/workspace/cse11/environment/samples/tracing$ make DEBUG=1 clean all
2
3 # cd ../tracing/
4 # ./app_a 12
5 fibonacci.c, 70, Oct 31 2015, 15:35:42
6 The first 12 Fibonacci numbers are:
7 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,

```

Avec la trace inactive

```

1 lmi@cse11:~/workspace/cse11/environment/samples/tracing$ make clean all
2
3 # ./app_a 12
4 The first 12 Fibonacci numbers are:
5 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,

```

## 2.6.3 Core dumps

```

1 lmi@cse11:~/workspace/cse11/environment/samples/core_dumps$ make clean all
2
3 # cd ../core_dumps/
4 # ulimit -c unlimited
5 # ./app_a
6 Segmentation fault (core dumped)
7 # gdb app_a core
8 ...
9 Core was generated by './app_a'.
10 Program terminated with signal SIGSEGV, Segmentation fault.
11 #0 0x000083e0 in access_data () at core_dumps.c:31
12 31      *p=10;
13 (gdb) bt
14 #0 0x000083e0 in access_data () at core_dumps.c:31
15 #1 0x00008424 in call (n=0) at core_dumps.c:37
16 #2 0x00008420 in call (n=1) at core_dumps.c:36
17 #3 0x00008420 in call (n=2) at core_dumps.c:36
18 #4 0x00008420 in call (n=3) at core_dumps.c:36
19 #5 0x00008420 in call (n=4) at core_dumps.c:36
20 #6 0x00008420 in call (n=5) at core_dumps.c:36
21 #7 0x00008420 in call (n=6) at core_dumps.c:36
22 #8 0x00008420 in call (n=7) at core_dumps.c:36
23 #9 0x00008420 in call (n=8) at core_dumps.c:36
24 #10 0x00008420 in call (n=9) at core_dumps.c:36
25 #11 0x00008420 in call (n=10) at core_dumps.c:36
26 #12 0x00008448 in main () at core_dumps.c:43

```

### 2.6.4 Backtrace

Cet exemple s'effectue uniquement sur la machine hôte

```

1 lmi@cse11:~/workspace/cse11/environment/samples/core_dumps$ cd ../backtrace/
2 lmi@cse11:~/workspace/cse11/environment/samples/backtrace$ make clean all
3 ...
4 lmi@cse11:~/workspace/cse11/environment/samples/backtrace$ ./app_h
5 backtrace() returned 17 addresses
6 ./app_h[0x80484fc]
7 [0xb7727404]
8 ./app_h[0x804854a]
9 ./app_h[0x8048571]
10 ./app_h[0x804856c]
11 ./app_h[0x804856c]
12 ./app_h[0x804856c]
13 ./app_h[0x804856c]
14 ./app_h[0x804856c]
15 ./app_h[0x804856c]
16 ./app_h[0x804856c]
17 ./app_h[0x804856c]
18 ./app_h[0x804856c]
19 ./app_h[0x804856c]
20 ./app_h[0x804859c]
21 /lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3)[0xb757ca83]
22 ./app_h[0x8048401]
23 Segmentation fault (core dumped)
24 lmi@cse11:~/workspace/cse11/environment/samples/backtrace$ addr2line -e app_h 0
    x804859c
25 /home/lmi/workspace/cse11/environment/samples/backtrace/main.c:61

```

### 2.6.5 System calls

```

1 lmi@cse11:~/workspace/cse11/environment/samples/system_calls$ make clean all
2
3 # cd ../system_calls/
4 # ./app_a
5 current temperature: 46.00 degree Celcius
6
7 # strace ./app_a
8 execve("./app_a", ["./app_a"], [/* 21 vars */]) = 0
9 brk(0)                                = 0x11000
10 ...
11 +++ exited with 0 +++
12
13 # strace -e trace=mmap2 ./app_a
14 ...
15 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    x6ff9000
16 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    x6ffa000
17 current temperature: 47.00 degree Celcius
18 +++ exited with 0 +++

```

### 2.6.6 Memory leaks

```

1 lmi@cse11:~/workspace/cse11/environment/samples/memory_leaks$ make clean all
2
3 # cd ../memory_leaks/
4 # ./app_a
5 # valgrind --leak-check=full ./app_a
6 ==1714== Memcheck, a memory error detector
7 ==1714== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
8 ==1714== Using Valgrind-3.10.0 and LibVEX; rerun with -h for copyright info
9 ==1714== Command: ./app_a
10 ==1714==
11 ==1714==
12 ==1714== HEAP SUMMARY:
13 ==1714==      in use at exit: 31,880 bytes in 3,985 blocks
14 ==1714==    total heap usage: 4,000 allocs, 15 frees, 32,000 bytes allocated
15 ==1714==
16 ==1714== 31,880 (8 direct, 31,872 indirect) bytes in 1 blocks are definitely
17 ==1714==    lost in loss record 2 of 2
18 ==1714==    at 0x483535C: malloc (in /usr/lib/valgrind/vgpreload_memcheck-arm-
19 ==1714==    linux.so)
20 ==1714== LEAK SUMMARY:
21 ==1714==    definitely lost: 8 bytes in 1 blocks
22 ==1714==    indirectly lost: 31,872 bytes in 3,984 blocks
23 ==1714==    possibly lost: 0 bytes in 0 blocks
24 ==1714==    still reachable: 0 bytes in 0 blocks
25 ==1714==    suppressed: 0 bytes in 0 blocks
26 ==1714== For counts of detected and suppressed errors, rerun with: -v
27 ==1714== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

```

## 2.7 Mise en production de l'ODROID-XU3

```

1 lmi@cse11:~/workspace/cse11/environment/samples/daemon$ make clean all

```

Cette commande génère deux fichiers :

1. app\_a
2. S60appl

Le fichier S60appl a été légèrement modifié pour que le path vers app\_a corresponde :

```

1 #!/bin/sh
2 #
3 # Daemon application
4 #

```

```

5 case "$1" in
6     start)
7         /usr/workspace/csel1/environment/samples/daemon/app_a
8         ;;
9     stop)
10        killall app_a
11        ;;
12    restart|reload)
13        killall app_a
14        /usr/workspace/csel1/environment/samples/daemon/app_a
15        ;;
16    *)
17        echo $"Usage: $0 {start|stop|restart}"
18        exit 1
19 esac
20
21 echo "Daemon application launched"
22
23 exit $?

```

Il suffit ensuite de copier ce fichier dans */etc/init.d* et de faire un reboot. Au démarrage de la cible, le script */etc/init.d/rcs* va effectuer tous les fichiers S?? présent dans le répertoire */etc/init.d*, donc notre application également.

```

1 # cp /usr/workspace/csel1/environment/samples/daemon/S60appl /etc/init.d
2 #reboot
3 ...
4 Starting logging: OK
5 Starting mdev...
6 Initializing random number generator... done.
7 Starting network...
8 ip: RTNETLINK answers: File exists
9 Starting sshd: OK
10 Daemon application launched
11 [ 9.933833] [c4] pwm-samsung: tin parent at 66600000

```

## 2.8 Réponse aux questions

1. Comment faut-il procéder pour générer l'U-Boot ?

---

2. Comment peut-on ajouter et générer un package supplémentaire dans le Buildroot ?

---

3. Comment doit-on procéder pour modifier la configuration du noyau Linux ?

---

4. Comment faut-il faire pour générer son propre RootFS ?



---

5. Comment faut-il procéder pour utiliser la carte eMMC en lieu et place de la carte SD ?

---

## 3 Travaux pratiques 1

### 3.1 Gestion de la mémoire, bibliothèques et fonctions utiles

#### 3.1.1 Exercice 4

**Donnée :** Créer dynamiquement des éléments dans le noyau. Adapter un module noyau afin que l'on puisse lors de son installation spécifier un nombre d'éléments à créer ainsi qu'un texte initial à stocker dans les éléments précédemment alloués. Chaque élément contiendra également un numéro unique, Les éléments seront créés lors de l'installation du module et chaînés dans une liste. Ces éléments seront détruits lors de la désinstallation du module. Des messages d'information seront émis afin de permettre le debugging du module.

#### 3.1.2 Exercice 5

**Donnée :** Indiquer les différents allocateurs SLAB disponibles dans le noyau Linux pour la cible ORDOID-XU3

1. SLAB : "as cache friendly as possible, benchmark friendly"
2. SLOB : "as compact as possible"
3. SLUB : "Simple and instruction cost counts. Superior Debugging. Defragmentation. Execution time friendly"

Source : [https://www.google.ch/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0CDEQFjACahUKEwiqj6GfhKbIAhWLXBoKHxDUAow&url=http%3A%2F%2Fwww.cs.berkeley.edu%2F~kubitron%2Fcourses%2Fcs194-24-S14%2Fhand-outs%2Fbonwick\\_slab.pdf&usg=AFQjCNENx6NuNkg&sig2=ZdJ\\_jUWHIf01qFIiIkEyHA](https://www.google.ch/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0CDEQFjACahUKEwiqj6GfhKbIAhWLXBoKHxDUAow&url=http%3A%2F%2Fwww.cs.berkeley.edu%2F~kubitron%2Fcourses%2Fcs194-24-S14%2Fhand-outs%2Fbonwick_slab.pdf&usg=AFQjCNENx6NuNkg&sig2=ZdJ_jUWHIf01qFIiIkEyHA)

### 3.2 Accès aux entrées/sorties

#### 3.2.1 Exercice 6

À l'aide d'un module noyau, réserver la zone mémoire correspondante au registre du uP décrivant son identification. Adress de départ 0x1000'0000, taille de la zone 0x100. Valider cette réservation à l'aide de la commande

```
1 cat /proc/iomem
```

Adapter ce module afin d'afficher cet identifiant dans la console de débogage "dmesg". Ce dernier est composé des champs suivants :

1. Bit 31..12 : product id
2. Bit 11..8 : package id
3. Bit 7..4 : major revision
4. Bit 3..0 : minor revision

Voici le code :

```

1 #include <linux/module.h>
2 #include <linux/init.h>
3 #include <linux/kernel.h>
4 #include <linux/io.h>
5 #include <linux/ioport.h>
6
7 const unsigned long addr = 0x10000000;
8 const unsigned long mem_len = 0x100;
9
10 static int __init skeleton_init(void)
11 {
12     void* regs;
13     int memId;
14
15     pr_info("Linux module skeleton loaded\n");
16
17     // Allocate memory region
18     request_mem_region(addr, mem_len, "uP register");
19
20     pr_info("Memory allocated\n");
21
22     // Mapp memory with linux noyau
23     regs = ioremap(addr, mem_len);
24     if (regs == NULL) {
25         pr_info("Error while trying to map processor chipid register...\n");
26         return -EFAULT;
27     }
28
29     // Read memory id;
30     memId = ioread32(regs+0x00);
31     pr_info("uP register: Bit 31..12 : product id=0x%05x\n", (memId>>12)&0x7fff);
32     pr_info("uP register: Bit 11..8 : package id=0x%01x\n", (memId>>8)&0xf);
33     pr_info("uP register: Bit 7..4 : major revision=0x%01x\n", (memId>>4)&0xf);
34     pr_info("uP register: Bit 3..0 : minor revision=0x%01x\n", (memId)&0xf);
35
36     iounmap(regs);
37
38     return 0;
39 }
40
41 static void __exit skeleton_exit(void)
42 {
43     pr_info("Linux module skeleton unloaded\n");
44
45     // Free memory region
46     release_mem_region(addr, mem_len);
47

```

```

48 pr_info("Memory released\n");
49 }
50
51 module_init(skeleton_init);
52 module_exit(skeleton_exit);
53
54 MODULE_AUTHOR("Emilie Gsponer");
55 MODULE_DESCRIPTION("Module Skeleton");
56 MODULE_LICENSE("GPL");

```

code/skeleton\_serlex6.c

L'identifiant demandé par la donnée est en fait l'identifiant de la zone mémoire réservée. Il peut être obtenu grâce à la méthode *ioread*. Mais pour pouvoir lire cette zone, il faut la mapper dans la mémoire virtuelle du noyau avec la méthode *ioremap*, car le noyau n'a pas directement accès aux entrées/sorties.

### 3.3 Gestion des interruptions

#### 3.3.1 Exercice 9

Développement d'un petit module permettant de capturer les pressions exercées sur les sw-tiches de la carte d'extension par interruption. Afin de permettre le debugging du module, chaque capture affichera un petit message.

Informations fournies :

- Configurer la direction des GPIO en entrée :

```
1 gpio_request(EXYNOS5_GPX<gpio_nr>(<pin_nr>));
```

```
1 gpio_direction_input(EXYNOS5_GPX<gpio_nr>(<pin_nr>));
```

- Obtenir le vecteur d'interruption avec le service suivant :

```
1 gpio_to_irq(<io_nr>);
```

- Informations sur les switches de la carte d'extension

- sw1 - gpio\_nr=2, pin\_nr=5, io\_nr=29
- sw2 - gpio\_nr=2, pin\_nr=6, io\_nr=30
- sw3 - gpio\_nr=1, pin\_nr=6, io\_nr=22
- sw4 - gpio\_nr=1, pin\_nr=2, io\_nr=18

Voici le code. Les switches de 1 à 4 sont interceptés.

```

1 /*skeleton.c*/
2 #include <linux/module.h>
3 #include <linux/init.h>
4 #include <linux/kernel.h>
5 #include <linux/moduleparam.h>
6 #include <linux/interrupt.h>

```

```

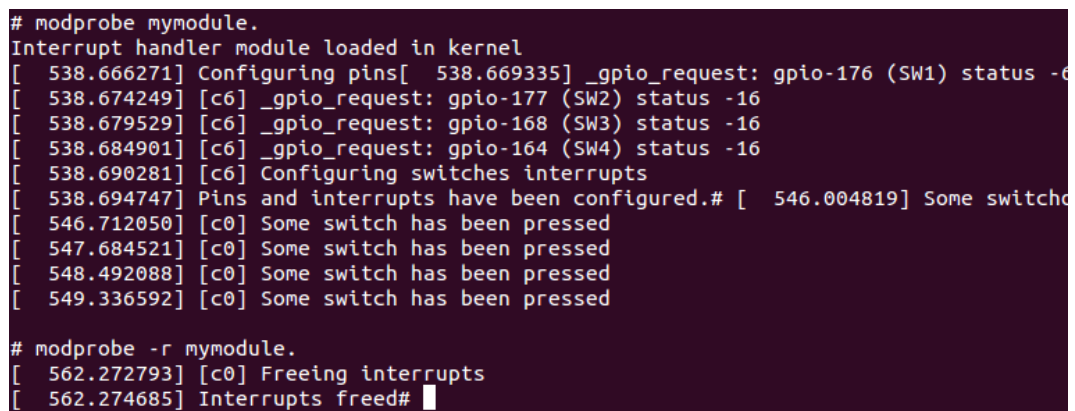
7 #include <linux/io.h>
8 #include <linux/gpio.h>
9
10 #define CHECK_RET(ret) if (ret != 0) return -1;
11
12 /*First argument, 50 length's string*/
13 static char* text="dummy help";
14 module_param(text, charp, 0);
15 /*Second argument, int */
16 static int element_num = 1;
17 module_param(element_num, int, 0);
18
19 static int Pin_nr = 4;
20 static int SW_pin_nr[4] = {5, 6, 6, 2};
21 static int SW_io_nr[4] = {29, 30, 22, 18};
22 static int SW_ID[4] = {100, 101, 102, 103};
23 static char * SW_name[4] = {"SW1", "SW2", "SW3", "SW4"};
24
25 irqreturn_t switch_irq_handler(int irq, void *dev_id){
26     pr_info("Some switch has been pressed\n");
27     return IRQ_HANDLED;
28 }
29
30 static int __init skeleton_init(void){
31     int ret;
32     int i;
33     ret = -1;
34     pr_info("Interrupt handler module loaded in kernel");
35     pr_info("Configuring pins");
36     for(i = 0; i < Pin_nr; i++) {
37         if(i < 2) {
38             ret = gpio_request(EXYNOS5_GPX2(SW_pin_nr[i]), SW_name[i]);
39             ret = gpio_direction_input(EXYNOS5_GPX2(SW_pin_nr[i]));
40         } else {
41             ret = gpio_request(EXYNOS5_GPX1(SW_pin_nr[i]), SW_name[i]);
42             ret = gpio_direction_input(EXYNOS5_GPX1(SW_pin_nr[i]));
43         }
44         CHECK_RET(ret);
45     }
46     pr_info("Configuring switches interrupts");
47     for(i = 0; i < Pin_nr; i++){
48         ret = request_irq(gpio_to_irq(SW_io_nr[i]),
49             switch_irq_handler,
50             IRQF_SHARED|IRQF_TRIGGER_RISING,
51             SW_name[i],
52             &SW_ID[i]);
53         CHECK_RET(ret);
54     }
55     pr_info("Pins and interrupts have been configured.");
56     return ret;
57 }
58
59 static void __exit skeleton_exit(void){
60     int i;
61     pr_info("Freeing interrupts");
62     for(i = 0; i < Pin_nr; i++)

```

```
63     free_irq(gpio_to_irq(SW_io_nr[i]), &SW_ID[i]);
64     pr_info("Interrupts freed");
65 }
66
67 module_init(skeleton_init);
68 module_exit(skeleton_exit);
69
70 MODULE_AUTHOR("Greg");
71 MODULE_DESCRIPTION("MISC");
72 MODULE_LICENSE("GPL");
```

code/skeleton\_serlex9.c

Et voici la preuve que tout fonctionne conformément, avec un message s'affichant pour chaque bouton pressé :



```
# modprobe mymodule.
Interrupt handler module loaded in kernel
[ 538.666271] Configuring pins[ 538.669335] _gpio_request: gpio-176 (SW1) status -6
[ 538.674249] [c6] _gpio_request: gpio-177 (SW2) status -16
[ 538.679529] [c6] _gpio_request: gpio-168 (SW3) status -16
[ 538.684901] [c6] _gpio_request: gpio-164 (SW4) status -16
[ 538.690281] [c6] Configuring switches interrupts
[ 538.694747] Pins and interrupts have been configured.# [ 546.004819] Some switch
[ 546.712050] [c0] Some switch has been pressed
[ 547.684521] [c0] Some switch has been pressed
[ 548.492088] [c0] Some switch has been pressed
[ 549.336592] [c0] Some switch has been pressed

# modprobe -r mymodule.
[ 562.272793] [c0] Freeing interrupts
[ 562.274685] Interrupts freed#
```

FIGURE 9 – Affichage du chargement du module, des pressions sur les boutons et de la suppression du module