

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

Émilie GSPONER, Grégory EMERY

5 novembre 2015
version 1.0

Table des matières

1	Introduction	3
2	Environnement Linux Embarqué	3
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	4
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
3	Programmation noyau : Module noyau	17
3.1	Module noyau	17
3.1.1	Exercice 1	17
3.1.2	Exercice 2	19
3.1.3	Exercice 3	20
3.2	Gestion de la mémoire, bibliothèques et fonctions utiles	21
3.2.1	Exercice 4	21
3.2.2	Exercice 5	21
3.3	Accès aux entrées/sorties	22
3.3.1	Exercice 6	22
3.4	Threads du noyau	23
3.4.1	Exercice 7	23
3.5	Mise en sommeil	23
3.5.1	Exercice 8	23
3.6	Gestion des interruptions	24
3.6.1	Exercice 9	24
4	Programmation noyau : Pilotes de périphériques	25
4.1	Pilotes orientés mémoire	25
4.1.1	Exercice 1	25
4.1.2	Exercice 2	26
4.2	Pilotes orientés caractères	27
4.2.1	Exercice 3	27
4.2.2	Exercice 4	27

4.2.3	Exercice 5	28
4.3	Opérations bloquantes	28
4.3.1	Exercice 6	28
4.4	sysfs	29
4.4.1	Exercice 7	29
4.5	ioctl (optionnel)	29
4.5.1	Exercice 8	29
4.6	procfs (optionnel)	29
4.6.1	Exercice 9	29
4.7	Gestionnaires de périphériques	29
4.7.1	Exercice 10	29

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 XU3¹ et U-Boot² dans le cadre du cours de Master HES-SO en systèmes embarqués, orientation TIN et TIC.

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. Lien : http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127
 2. Lien : <http://www.denx.de/wiki/U-Boot>

```
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,
5     nosuid,nodev,uhelper=udisks2)
6 ...
```

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.

Emplacement du script : */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

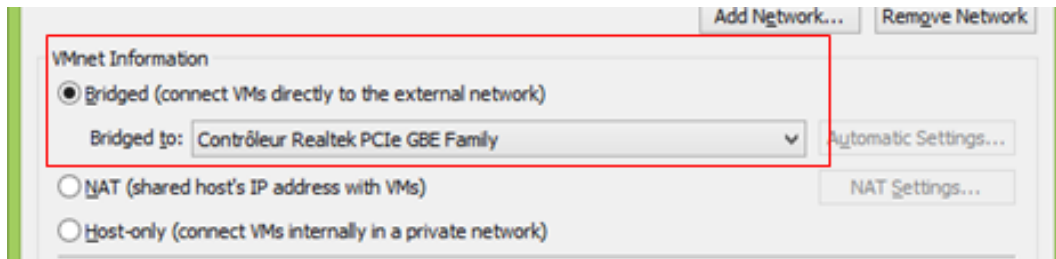


FIGURE 1 – Configuration de la carte réseau

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

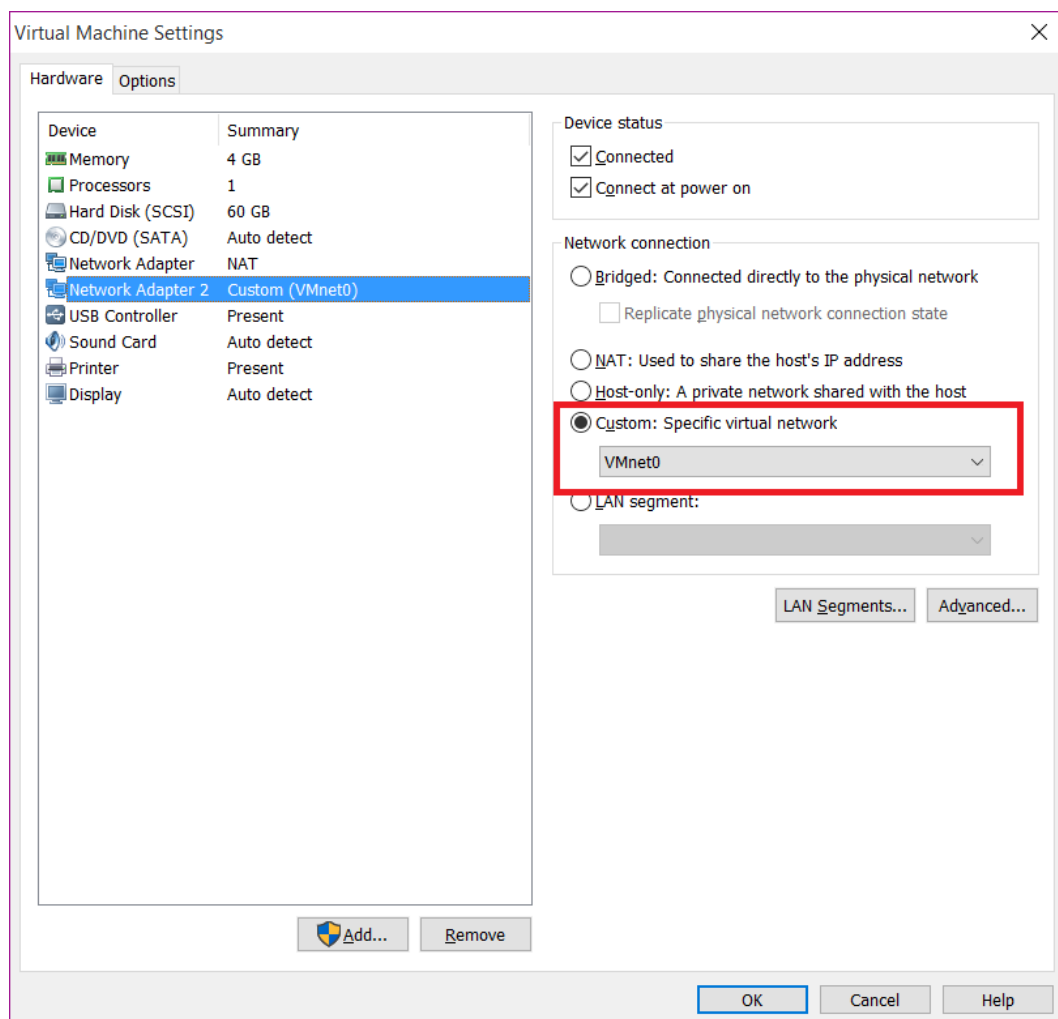


FIGURE 2 – Association de la carte réseau à la machine virtuelle

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

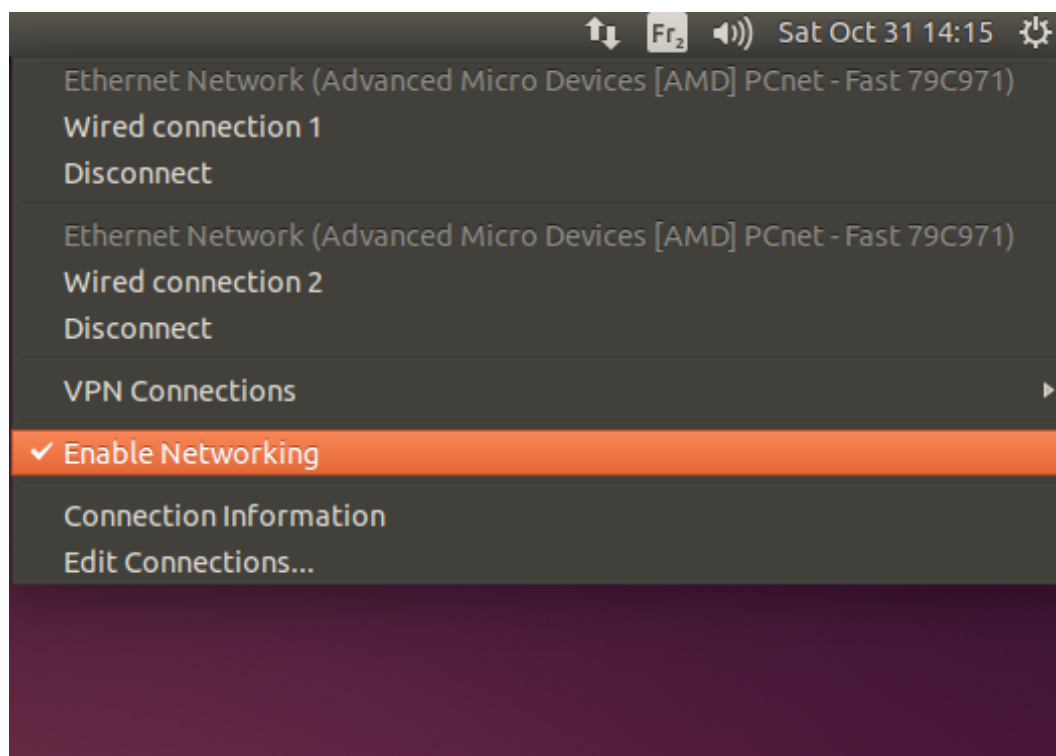


FIGURE 3 – Activation du réseau

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
14 # reboot
```

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 :



9

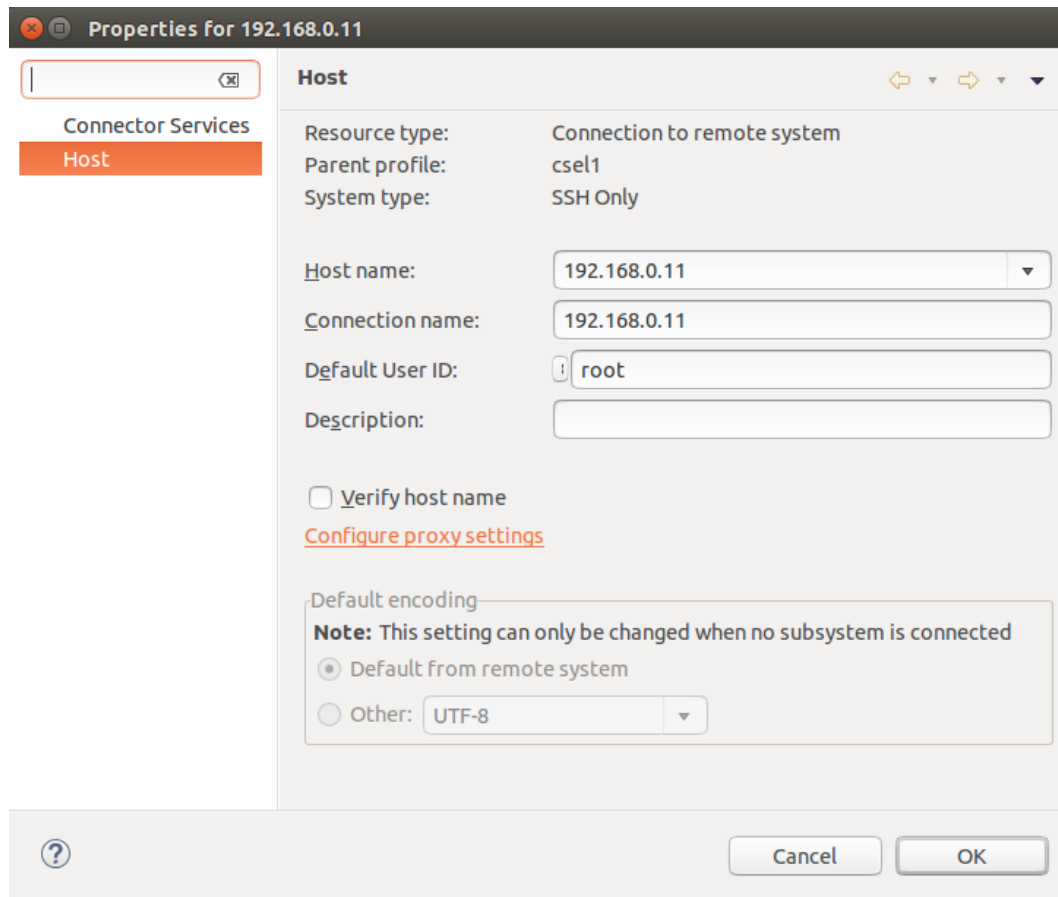


FIGURE 5 – Configuration de l'accès ssh

Il faut encore aller configurer le debugger gdb :

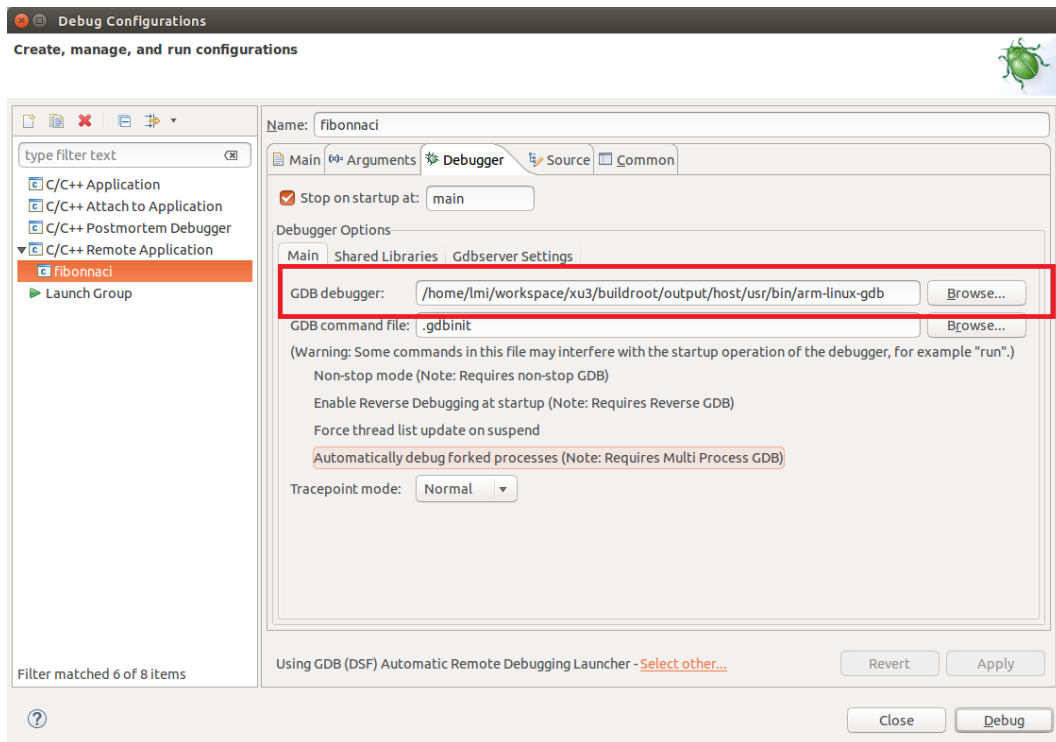


FIGURE 6 – Configuration du server gdb

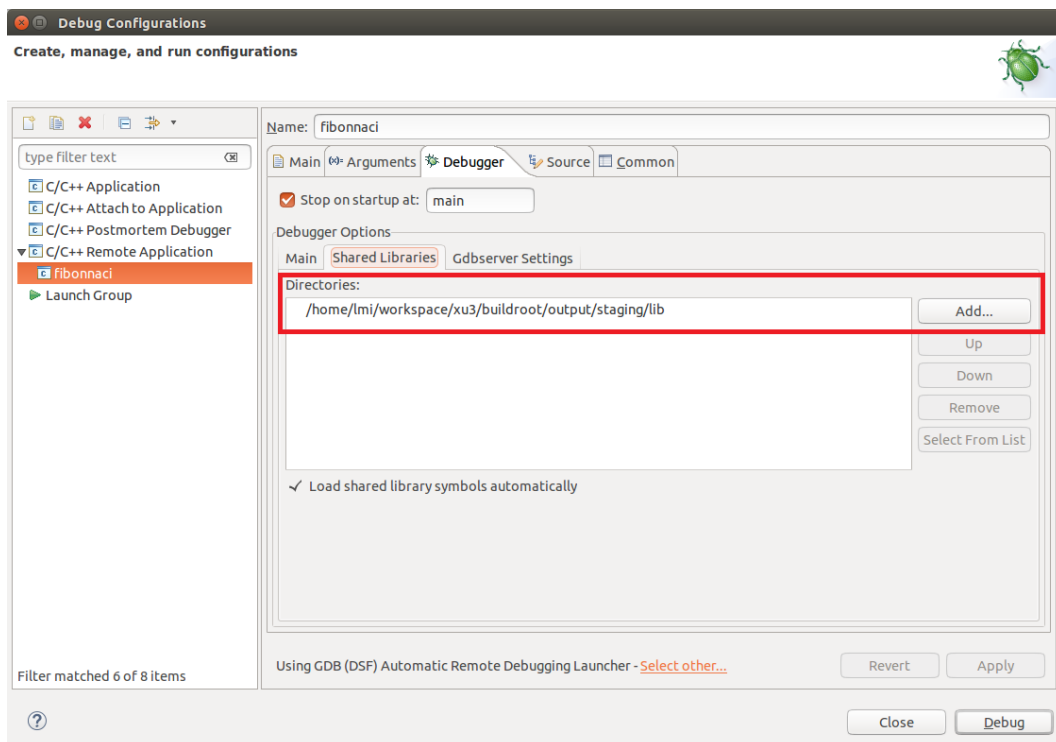


FIGURE 7 – Configuration des shared library

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

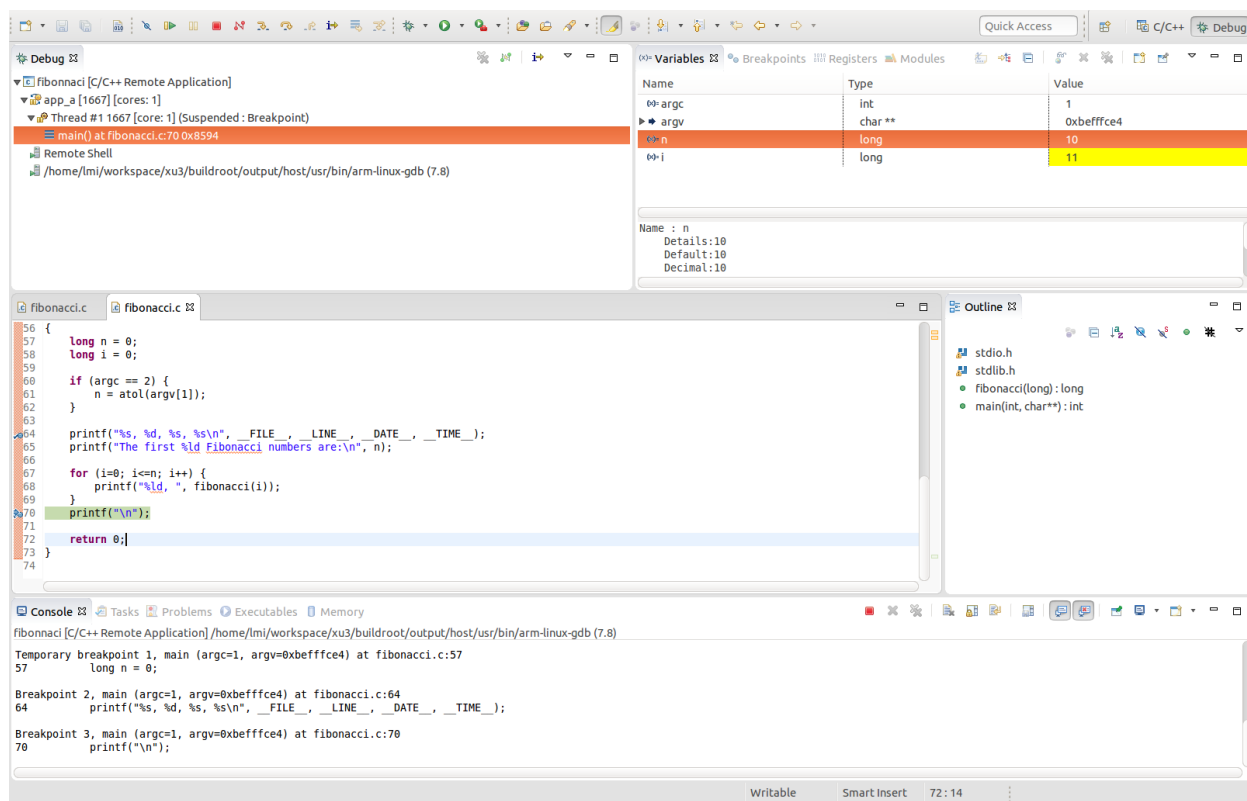


FIGURE 8 – Debug de l'exemple Fibonacci

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 arrêtée avant démarrage

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
    xb6ff9000
16 mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    xb6ffa000
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 Programmation noyau : Module noyau

3.1 Module noyau

3.1.1 Exercice 1

Donnée : Générer un module noyau "out of tree" pour la cible ODROID-XU3

Point a : Créer le squelette d'un module noyau et générer le en dehors des sources du noyau à l'aide d'un Makefile. Le module devra afficher un message lors de son enregistrement et lors de sa désinstallation.

Emplacement du code : */ModulesNoyau/exercice1-Module/pointA*

Le code a été écrit conformément aux exemples présentés dans le support de cours.

Exécution du code :

Le code doit être compilé sur la machine hôte à l'aide de la commande suivante :

```
1 $ make clean all
```

Point b : Tester sur la machine hôte la commande "modinfo" sur votre squelette de module et comparer les informations retournées avec celles du code source.

```
1 $ modinfo mymodule.ko
2 filename:          /home/lmi/workspace/csell/environment/module_noyau/exercice1 /
   pointa/mymodule.ko
3 license:           GPL
4 description:       Module Skeleton
5 author:            Emilie Gsponer
6 depends:
7 vermagic:          3.10.63 SMP preempt mod_unload ARmv7 p2v8
```

Ces informations correspondent à celles entrée dans le skeleton du module :

```
1 MODULE_AUTHOR("Emilie Gsponer");
2 MODULE_DESCRIPTION("Module Skeleton");
3 MODULE_LICENSE("GPL");
```

Point c : Installer le module (insmod), contrôler le log du noyau (dmesg).

Exécution du code : Le module doit être installé sur la cible à l'aide de la commande *insmod*. On peut observer que le module a bien été installé à l'aide de la commande *dmesg*.

```

1 # insmod mymodule.ko
2 # dmesg | tail -n 4
3 [    7.704536] [c5] Freeing unused kernel memory: 436K (c089c000 - c0909000)
4 [    7.877462] [c4] EXT4-fs (mmcblk0p1): re-mounted. Opts: errors=remount-ro,
    data=ordered
5 [    8.967876] [c7] pwm-samsung: tin parent at 66600000
6 [   53.629226] [c4] Linux module skeleton loaded

```

Point d : Comparer les résultats obtenus par la commande "lsmod" avec ceux obtenus avec la commande "cat /proc/modules".

Exécution des commandes : Les deux commandes présentent le même résultat, seule la mise en forme est différente.

```

1 # cat /proc/modules
2 mymodule 687 0 - Live 0xbf004000 (O)
3
4 # lsmod mymodule.ko
5 Module                               Size  Used by    Tainted: G
6 mymodule                             687   0

```

Point e : Désinstaller le module (rmmod).

Exécution de la commande : On peut voir à l'aide des commandes *cat /proc/modules*, *lsmod* et *dmesg* que le module a bien été désinstallé.

```

1 # rmmod mymodule
2 # cat /proc/modules
3 # lsmod mymodule.ko
4 Module                               Size  Used by    Tainted: G
5 # dmesg | tail -n 4
6 [    7.877462] [c4] EXT4-fs (mmcblk0p1): re-mounted. Opts: errors=remount-ro,
    data=ordered
7 [    8.967876] [c7] pwm-samsung: tin parent at 66600000
8 [   53.629226] [c4] Linux module skeleton loaded
9 [   73.538490] [c5] Linux module skeleton unloaded

```

Point f : Adapter le Makefile du module pour autoriser l'installation du module avec les autres modules du noyau permettant l'utilisation de la commande "modprobe". Le module devra être installé dans le root filesystem utilisé en nfs par la cible.

Emplacement du code : */ModulesNoyau/exercice1-Module/pointF*

Pour que le module puisse être ajouté dans le noyau, il suffit d'ajouter les lignes suivantes au Makefile précédent :

```

1 MODPATH := /tftpboot/odroidxu3
2
3 install:
4 $(MAKE) -C $(KDIR) M=$(PWD) INSTALL_MOD_PATH=$(MODPATH) modules_install

```

Exécution du code : Le code doit être compilé et installé sur la machine hôte à l'aide des commandes suivantes :

```
1 $ make clean all
2 $ sudo make install
```

Il faut ensuite démarrer la cible en mode nfs pour y installer le module à l'aide de la commande *modprobe*

```
1 lmi@cse11:~/.ssh$ sudo minicom
2 ...
3 ODROIDXU3> run nfsboot
4 (Re)start USB...
5 ...
6 Welcome to Hardkernel ODROID_XU3 board
7 odroidxu3 login: root
8 # modprobe mymodule
9 [ 98.024580] [c2] Linux module skeleton loaded
10 # modprobe -r mymodule
11 [ 99.673514] [c0] Linux module skeleton unloaded
```

3.1.2 Exercice 2

Donnée : Adapter le module de l'exercice précédent afin qu'il puisse recevoir deux ou trois paramètres de votre choix. Ces paramètres seront affichés dans la console. Adapter également le rootfs afin de pouvoir utiliser la commande "modprobe".

Emplacement du code : */ModulesNoyau/exercice1-Module/exercice2-Parameter*

Exécution du code : Ce module prend deux paramètres d'entrée par défaut, text = "hello" et elements = 2. La valeur de ces éléments peut être modifiée lors de l'installation du module dans le noyau, comme le montre la démonstration suivante.

```
1 # modprobe mymodule
2 [ 922.323941] [c1] Linux module skeleton loaded
3 [ 922.326835] [c1] text:hello
4 [ 922.326835] elements:2
5 # modprobe -r mymodule
6 [ 925.223513] [c0] Linux module skeleton unloaded
7 # modprobe mymodule text="hello world" elements=10
8 [ 1196.597223] [c2] Linux module skeleton loaded
9 [ 1196.600182] [c2] text:hello world
10 [ 1196.600182] elements:10
11 # modprobe -r mymodule
12 [ 1199.518513] [c0] Linux module skeleton unloaded
```

Remarque : Pour tous les exercices suivants, on reprendra le même Makefile permettant d'installer le module directement dans le noyau à l'aide de la commande *modprobe*. Il faudra pour cela que la cible soit démarrée en mode nfs. Le module de chaque exercice devra être compilé et installé à l'aide des commandes suivantes :

```
1 $ make clean all
2 $ sudo make install
```

3.1.3 Exercice 3

Donnée : Trouver la signification des 4 valeurs affichées lorsque l'on tape la commande "cat /proc/sys/kernel/printk".

Exécution de la commande :

```
1 # cat /proc/sys/kernel/printk
2 10      4      1      7
```

Ces chiffres représentent le log level du kernel utilisé pour déterminer l'importance d'un message.

1. 10 : current loglevel
2. 4 : default loglevel
3. 1 : minimum loglevel
4. 7 : boot-time-default loglevel

Cela permet de définir la priorité des messages. Si un message a une priorité plus grande (chiffre plus petit) que le loglevel courant du kernel, il sera affiché dans la console. Dans le cas contraire, le message n'est pas affiché. L'exemple suivant le démontre en passant le loglevel à 1, le module n'affiche plus de message dans la console.

```
1 # echo 10 > /proc/sys/kernel/printk
2 # cat /proc/sys/kernel/printk
3 10      4      1      7
4 # modprobe mymodule
5 [ 2139.129459] [c2] Linux module skeleton loaded
6 [ 2139.132348] [c2] text:hello
7 [ 2139.132348] elements:2
8 # modprobe -r mymodule
9 [ 2143.683511] [c0] Linux module skeleton unloaded
10
11 # echo 1 > /proc/sys/kernel/printk
12 # cat /proc/sys/kernel/printk
13 1      4      1      7
14 # modprobe mymodule
15 # modprobe -r mymodule
16
17 # echo 10 > /proc/sys/kernel/printk
18 # modprobe mymodule
19 [ 2173.021095] [c2] Linux module skeleton loaded
```

```

20 [ 2173.024050] [c2] text:hello
21 [ 2173.024050] elements:2

```

Source : http://elinux.org/Debugging_by_printing

3.2 Gestion de la mémoire, bibliothèques et fonctions utiles

3.2.1 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.

Emplacement du code : */ModulesNoyau/exercice4-List*

Exécution du code :

```

1 # modprobe mymodule element_num=5 text="Hello world"
2 [ 2250.418060] [c2] New element added to the list
3 [ 2250.421107] [c2] New element added to the list
4 [ 2250.425536] [c2] New element added to the list
5 [ 2250.429937] [c2] New element added to the list
6 [ 2250.434368] [c2] New element added to the list
7 [ 2250.438773] [c2] ID of element : 0
8 [ 2250.438773] String is : Hello world
9 [ 2250.445642] [c2] ID of element : 1
10 [ 2250.445642] String is : Hello world
11 [ 2250.452489] [c2] ID of element : 2
12 [ 2250.452489] String is : Hello world
13 [ 2250.459415] [c7] ID of element : 3
14 [ 2250.459415] String is : Hello world
15 [ 2250.466444] [c7] ID of element : 4
16 [ 2250.466444] String is : Hello world
17 # modprobe -r mymodule
18 [ 2267.933575] [c1] Element popped
19 [ 2267.935161] [c1] Element popped
20 [ 2267.938196] [c1] Element popped
21 [ 2267.941283] [c1] Element popped
22 [ 2267.944313] [c1] Element popped
23 [ 2267.947308] [c1] Module removed

```

3.2.2 Exercice 5

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

1. SLAB : "as cache frendly as possible, benchmark frendly"

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_jUWHIf01qFIikEyHA

3.3 Accès aux entrées/sorties

3.3.1 Exercice 6

Donnée : 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 "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

Emplacement du code : */ModulesNoyau/exercice6-IO*

Complément : 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.

La zone mémoire réservée par le code a été nommée "uP register".

Exécution du code :

```

1 # modprobe mymodule
2 [ 30.336553] [c5] Linux module skeleton loaded
3 [ 30.339495] [c5] Memory allocated
4 [ 30.342733] [c5] uP register: Bit 31..12 : product id=0x65422
5 [ 30.348494] [c5] uP register: Bit 11..8 : package id=0x0
6 [ 30.353877] [c5] uP register: Bit 7..4 : major revision=0x0
7 [ 30.359597] [c5] uP register: Bit 3..0 : minor revision=0x1
8 # cat /proc/iomem
9 03000000-03048fff : /lpass@03810000
10 03810000-038100ff : /lpass@03810000
11 03830000-038300ff : samsung-i2s
12 03860000-03860fff : /pinctrl@03860000
13 03880000-03880fff : /amba/adma@03880000
14 03880000-03880fff : /amba/adma@03880000
15 10000000-10000fff : uP register

```

```

16 ...
17 # dmesg | tail -n 10
18 [ 9.041140] [c4] VFS: Mounted root (nfs filesystem) on device 0:13.
19 [ 9.048765] [c4] devtmpfs: mounted
20 [ 9.050869] [c4] Freeing unused kernel memory: 436K (c089c000 - c0909000)
21 [ 12.571761] [c4] pwm-samsung: tin parent at 66600000
22 [ 30.336553] [c5] Linux module skeleton loaded
23 [ 30.339495] [c5] Memory allocated
24 [ 30.342733] [c5] uP register: Bit 31..12 : product id=0x65422
25 [ 30.348494] [c5] uP register: Bit 11..8 : package id=0x0
26 [ 30.353877] [c5] uP register: Bit 7..4 : major revision=0x0
27 [ 30.359597] [c5] uP register: Bit 3..0 : minor revision=0x1
28 # modprobe -r mymodule
29 [ 88.338556] [c0] Linux module skeleton unloaded
30 [ 88.341621] [c0] Memory released

```

3.4 Threads du noyau

3.4.1 Exercice 7

Donnée : Développer un petit module permettant d'instancier un thread dans le noyau. Ce thread affichera un message toutes les 5 secondes. Il pourra être mis en sommeil durant ces 5 secondes à l'aide de la fonction « `ssleep(5)` » provenant de l'interface `<linux/delay.h>`.

Emplacement du code : */ModulesNoyau/exercice7-Thread*

Exécution du code :

```

1 # modprobe mymodule
2 [ 1434.469821] [c2] Thread created
3 # [ 1439.473573] [c1] Thread awake
4 [ 1444.478588] [c2] Thread awake
5 [ 1449.483589] [c3] Thread awake
6 # modprobe -r mymodule
7 [ 1454.488597] [c0] Thread awake
8 [ 1454.490139] [c1] Thread stopped

```

3.5 Mise en sommeil

3.5.1 Exercice 8

Donnée : Développer un petit module permettant d'instancier deux threads dans le noyau. Le premier thread attendra une notification de réveil du deuxième thread et se remettra en sommeil. Le 2ème thread enverra cette notification toutes les 5 secondes et se rendormira. On utilisera les waitqueues pour les mises en sommeil. Afin de permettre le debugging du module, chaque thread affichera un petit message à chaque réveil.

Emplacement du code : */ModulesNoyau/exercice8-MiseEnSommeil*

Exécution du code :

```

1 # modprobe mymodule
2 [ 1523.052680] [c2] Init wait queue
3 [ 1523.054851] [c2] Threads created
4 # [ 1528.058588] [c3] Thread2 (notif each 5s) awake
5 [ 1528.061596] [c1] Thread1 (wait notif) awake
6 [ 1533.063590] [c0] Thread2 (notif each 5s) awake
7 [ 1533.066587] [c1] Thread1 (wait notif) awake
8 [ 1538.068588] [c0] Thread2 (notif each 5s) awake
9 [ 1538.071587] [c1] Thread1 (wait notif) awake
10 # modprobe -r mymodule
11 [ 1543.073563] [c0] Thread2 (notif each 5s) awake
12 [ 1543.076562] [c2] Thread1 (wait notif) awake
13 [ 1548.078569] [c0] Thread2 (notif each 5s) awake
14 [ 1548.081573] [c1] Threads stopped

```

3.6 Gestion des interruptions

3.6.1 Exercice 9

Donnée : Développement d'un petit module permettant de capturer les pressions exercées sur les swtiches 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

Complément : Les switches de 1 à 4 sont interceptés.

Emplacement du code : */ModulesNoyau/exercice9-Interrupt*

Exécution du code : 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

4 Programmation noyau : Pilotes de périphériques

4.1 Pilotes orientés mémoire

4.1.1 Exercice 1

Donnée : Réaliser un pilote orienté mémoire permettant de mapper en espace utilisateur les registres de la FPGA en utilisant le fichier virtuel `/dev/mem`. Ce pilote permettra de lire l'identification du uP (chip id) décrit dans l'exercice no 6 du cours sur la programmation de modules noyau.

Complément : Ce pilote n'est pas un module noyau, il faut prendre le Makefile d'un des codes d'exemples, par exemple Fibonacci. La cible doit tout de même être démarrée en mode nfs.

Emplacement du code : `/PilotesPeripheriques/exercice1-mmap`

Exécution du code :

```
1 # ./app_a
2 uP register: Bit 31..12 : product id=0x65422
3 uP register: Bit 11..8 : package id=0x0
4 uP register: Bit 7..4 : major revision=0x0
5 uP register: Bit 3..0 : minor revision=0x1
```

Remarque : On obtient bien le même id qu'avec le module noyau de l'exercice 6 de la série précédente.

Pour utiliser correctement la commande `mmap`, il ne faut pas mettre l'offset à 0, mais à `0x1000000` (adresse du chipid de l'exercice6), sinon le code ne fonctionne pas, il essaie de lire une zone mémoire qu'il n'a pas le droit.

4.1.2 Exercice 2

Donnée : Sur la base de l'exercice 1, développer un pilote orienté caractère permettant de mapper en espace utilisateur ces registres (implémentation de l'opération de fichier « `mmap` »). Le driver orienté mémoire sera ensuite adapté à cette nouvelle interface. Remarque : à effectuer après les exercices des pilotes orientés caractère

Emplacement du code :

/PilotesPeripheriques/exercice2-mmapModule/user
/PilotesPeripheriques/exercice2-mmapModule/noyau

Remarque : Cet exercice a été compliqué à réaliser. Le code est un mélange de l'exercice 1 et 5. Il faut garder en tête que pour mapper les registres en espace utilisateur, il faut utiliser les fonction standard des `file_operations` (`open`, `close`, `mmap`) et en plus, ajouter les `vm_operations_struct` (`open`, `close`) pour agir sur la zone mémoire. Dans le pilote orienté caractère, on utilise la fonction `remap_pfn_range` pour mapper la zone mémoire.

Exécution du code :

```
1 # modprobe mymodule
2 [ 854.599130] [c2] mod: successfully loaded with major 249
3 # mknod /dev/mod c 249 0
4 # ./app_a /dev/mod
5 [ 1166.619318] [c0] mod: open
6 [ 1166.620800] [c0] mod: mmap
7 [ 1166.623246] [c0] VMA open, virt b6f71000, phys 10000000
8 [ 1166.628986] [c0] VMA close.
9 [ 1166.631259] [c0] mod: release
10 file /dev/mod open
11 Physical memory: Bit 31..12 : product id=0x65422
12 Physical memory: Bit 11..8 : package id=0x0
13 Physical memory: Bit 7..4 : major revision=0x0
14 Physical memory: Bit 3..0 : minor revision=0x1
15
16 Virtual memory: Bit 31..12 : product id=0x43108
17 Virtual memory: Bit 11..8 : package id=0x3
18 Virtual memory: Bit 7..4 : major revision=0x2
19 Virtual memory: Bit 3..0 : minor revision=0xa
20 file /dev/mod close
21 # modprobe -r mymodule
22 [ 1259.338388] [c0] mod: successfully unloaded
```

4.2 Pilotes orientés caractères

4.2.1 Exercice 3

Donnée : Implémenter un pilote de périphérique orienté caractère. Ce pilote sera capable de stocker dans une variable globale au module les données reçues par l'opération write et de les restituer par l'opération read. Pour tester le module, on utilisera les commandes « echo » et « cat ».

Emplacement du code : */PilotesPeripheriques/exercice3-ReadWrite*

Exécution du code :

```
1 $ make clean all
2 $ sudo make install
```

```
1 # modprobe mymodule
2 [ 4107.528630] [c2] mod: successfully loaded with major 249
3 # mknod /dev/mod c 249 0
4 # echo -n test module > /dev/mod
5 [ 4134.907095] [c1] mod: open()
6 [ 4134.908540] [c1] mod: write test module
7 [ 4134.912539] [c1] mod: release()
8 # cat /dev/mod
9 [ 4145.581863] [c0] mod: open()
10 [ 4145.583312] [c0] mod: read test module
11 [ 4145.587140] [c0] mod: read test module
12 [ 4145.590820] [c0] mod: release()
13 # modprobe -r mymodule
14 [ 4157.528779] [c1] mod: successfully unloaded
```

4.2.2 Exercice 4

Donnée : Etendre la fonctionnalité du pilote de l'exercice #3 afin que l'on puisse à l'aide d'un paramètre module spécifier le nombre d'instance. Pour chaque instance on créera une variable unique permettant de stocker les données échangées avec l'application en espace utilisateur.

Emplacement du code : */PilotesPeripheriques/...*

Exécution du code :

4.2.3 Exercice 5

Donnée : Développer une petite application en espace utilisateur permettant d'accéder à ces pilotes orientés caractère. L'application devra écrire un texte dans le pilote et le relire.

Complément : Le module noyau a été repris de l'exercice 3.

Emplacement du code :

/PilotesPeripheriques/exercice5-UserAccess/user
/PilotesPeripheriques/exercice5-UserAccess/noyau

Exécution du code :

```

1 lmi@cse11:~/workspace/cse11/environment/peripheral/exercice5/user$ make clean
  all
2 lmi@cse11:~/workspace/cse11/environment/peripheral/exercice5/user$ cd ../noyau/
3 lmi@cse11:~/workspace/cse11/environment/peripheral/exercice5/noyau$ make clean
  all
4 lmi@cse11:~/workspace/cse11/environment/peripheral/exercice5/noyau$ sudo make
  install

```

```

1 # modprobe mymodule
2 [ 231.111019] [c2] mod: successfully loaded with major 249
3 # mknod /dev/mod c 249 0
4 # ./app_a /dev/mod HELLO
5 [ 234.969067] [c0] mod: open()
6 [ 234.970719] [c0] mod: write HELLO
7 [ 234.973927] [c0] mod: read HELLO
8 [ 234.977039] [c0] mod: release()
9 file /dev/mod open
10 write HELLO
11 read HELLO
12 file /dev/mod close
13 # modprobe -r mymodule
14 [ 261.512478] [c0] mod: successfully unloaded

```

4.3 Opérations bloquantes

4.3.1 Exercice 6

Donnée : Développer un pilote et une application utilisant les entrées/sorties bloquantes pour signaler une interruption matérielle provenant de l'un des switches de la carte d'extension de ODR0ID-XU3. L'application utilisera le service select pour compter le nombre d'interruptions.

Remarque : les switches non pas d'anti-rebond, par conséquent il est fort probable que vous comptiez un peu trop d'impulsions ; effet à ignorer.

Emplacement du code : */PilotesPeripheriques/...*

Exécution du code :

4.4 sysfs

4.4.1 Exercice 7

Donnée : Développer un pilote de périphérique orienté caractère permettant de valider la fonctionnalité du sysfs. Le pilote offrira des attributs de périphérique afin pouvoir lire et écrire un bloc de données composé de quelques membres et de pouvoir modifier le contenu de la valeur entière. Seules les commandes « echo » et « cat » doivent être nécessaire pour manipuler ces attributs.

Emplacement du code : */PilotesPeripheriques/...*

Exécution du code :

4.5 ioctl (optionnel)

4.5.1 Exercice 8

Cet exercice n'a pas été réalisé.

4.6 procfs (optionnel)

4.6.1 Exercice 9

Cet exercice n'a pas été réalisé.

4.7 Gestionnaires de périphériques

4.7.1 Exercice 10

Donnée : Implémenter à l'intérieur d'un pilote de périphérique orienté caractère, les mécanismes nécessaires à la création du fichier d'accès au pilote (remplacement de la commande « mknod ») par l'utilitaire « mdev » de la BusyBox.

Emplacement du code : */PilotesPeripheriques/...*

Exécution du code :

--