Construction de systèmes embarqués sous Linux

Rapport de laboratoire

Master HES-SO

Émilie Gsponer, Grégory Emery

4 décembre 2015
version 1.0

Table des matières

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 [1] et U-Boot [2] 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 oeuve 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-ba62-403debdf56a5 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

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

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

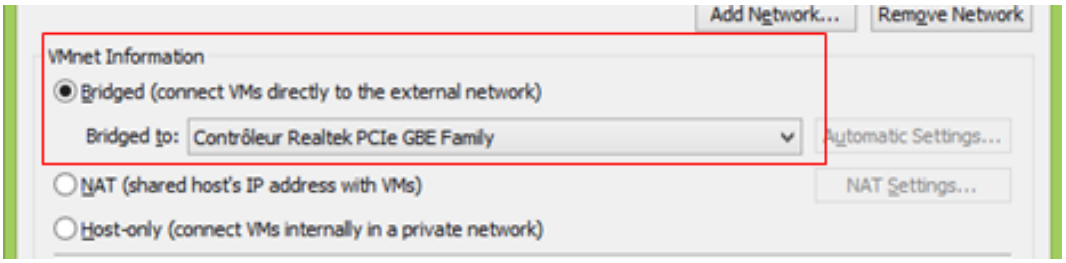

FIGURE 1 – Configuration de la carte réseau

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

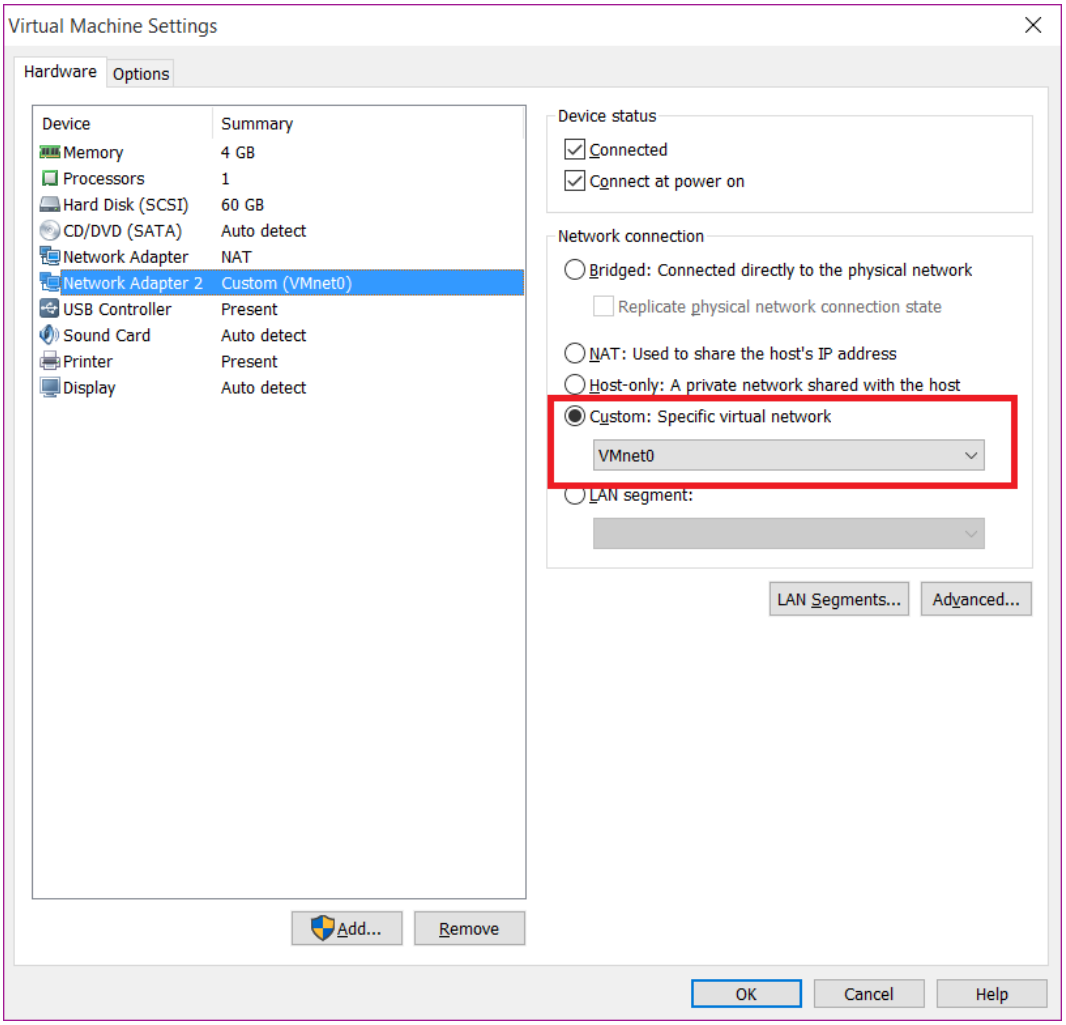

FIGURE 2 – Association de la carte réseau à la machine virtuelle

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

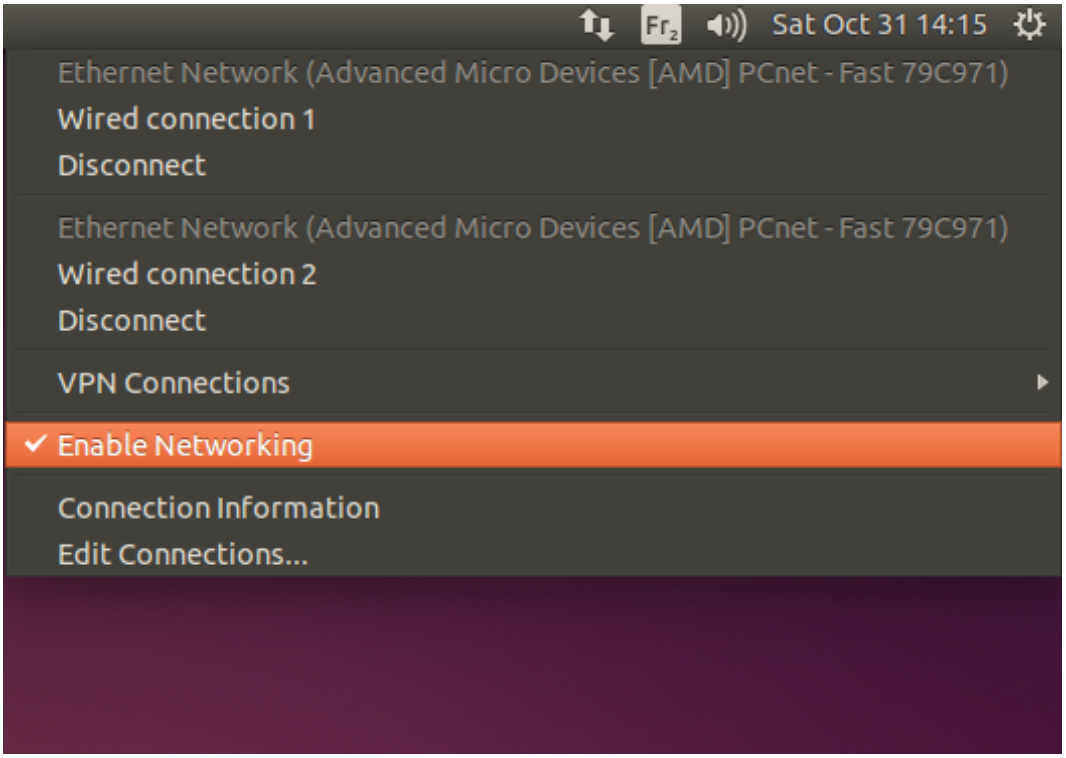

FIGURE 3 – Activation du réseau

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 :

```
$ sudo minicom

Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Jan  1 2014, 17:13:22.
Port /dev/ttyUSB0, 13:10:43

Press CTRL-A Z for help on special keys


Welcome to Hardkernel ODROID_XU3 board
odroidxu3 login: root
#
```

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 :

```
# reboot
```

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

```
lmi@csel1:~$ ssh root@192.168.0.11
#
```

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 :

```
lmi@csel1:~$ sudo minicom
...
# reboot
...
Press 'Enter' or 'Space' to stop autoboot:  0

ODROIDXU3> usb start
(Re)start USB...
USB:   Register 1313 NbrPorts 3
USB EHCI 1.00
scanning bus for devices... The request port(2) is not configured
The request port(2) is not configured
4 USB Device(s) found
scanning bus for storage devices... 0 Storage Device(s) found
scanning bus for ethernet devices... 1 Ethernet Device(s) found

ODROIDXU3> ping 192.168.0.4
Waiting for Ethernet connection... done.
Using sms0 device
host 192.168.0.4 is alive

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.

```
# mkdir /usr/workspace
# vi /etc/fstab
  # /etc/fstab: static file system information.
  #
  # <file system> <mount pt>     <type>   <options>          <dump> <pass>
  /dev/root       /              ext2     rw,noauto          0      1
  proc            /proc          proc     defaults           0      0
  devpts          /dev/pts       devpts   defaults,gid=5,mode=620   0      0
  tmpfs           /dev/shm       tmpfs    mode=0777          0      0
  tmpfs           /tmp           tmpfs    mode=1777          0      0
  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,wsize=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 :

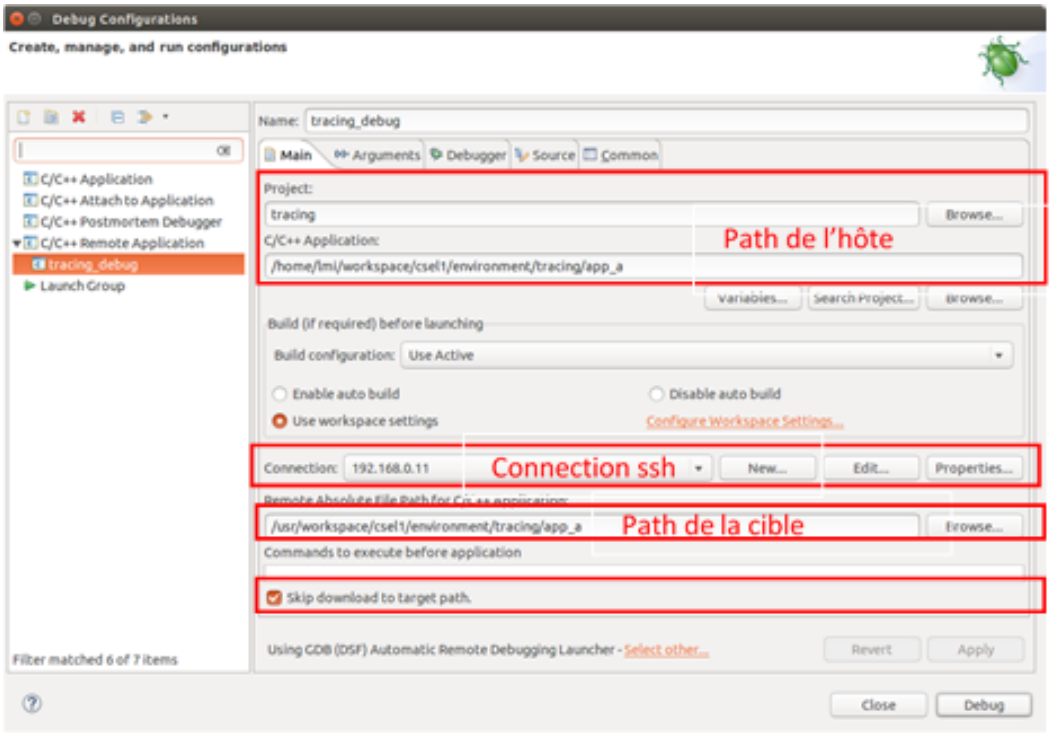

FIGURE 4 – Configuration du debugger

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

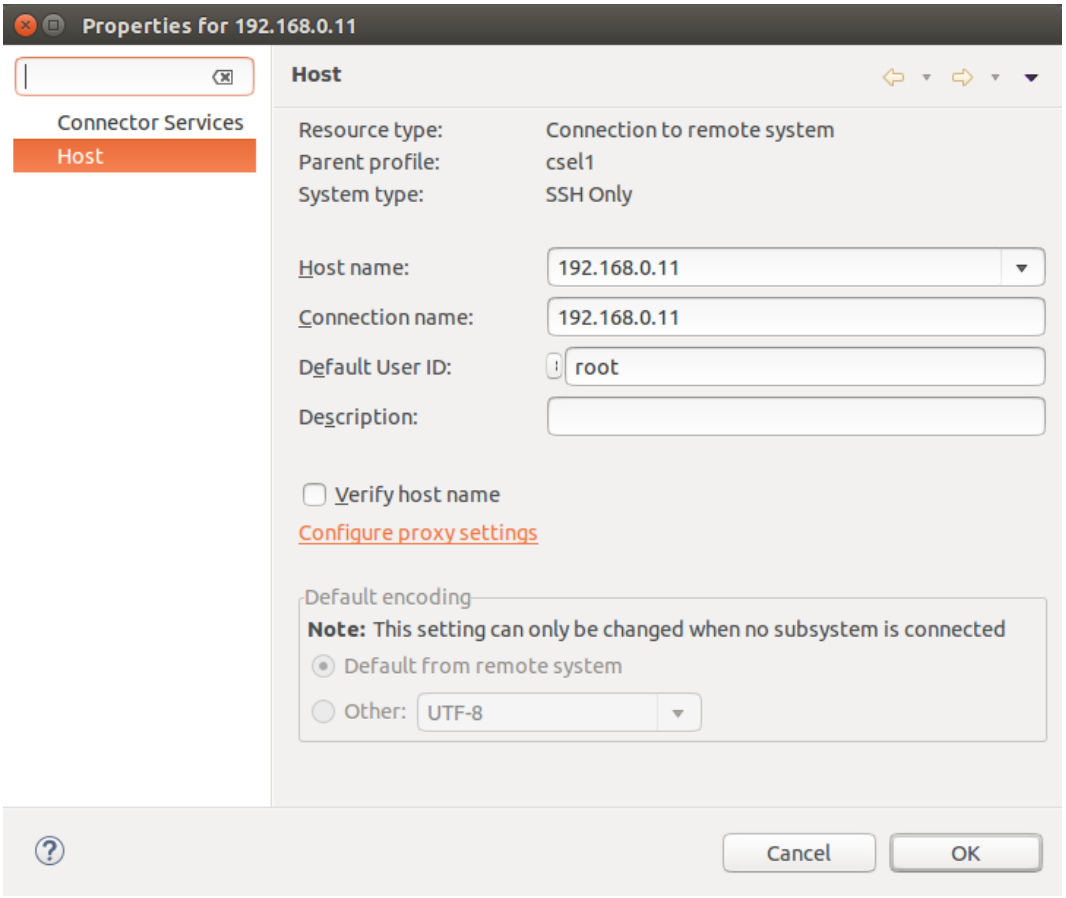

FIGURE 5 – Configuration de l'accès ssh

Il faut encore aller configurer le debugger gdb :

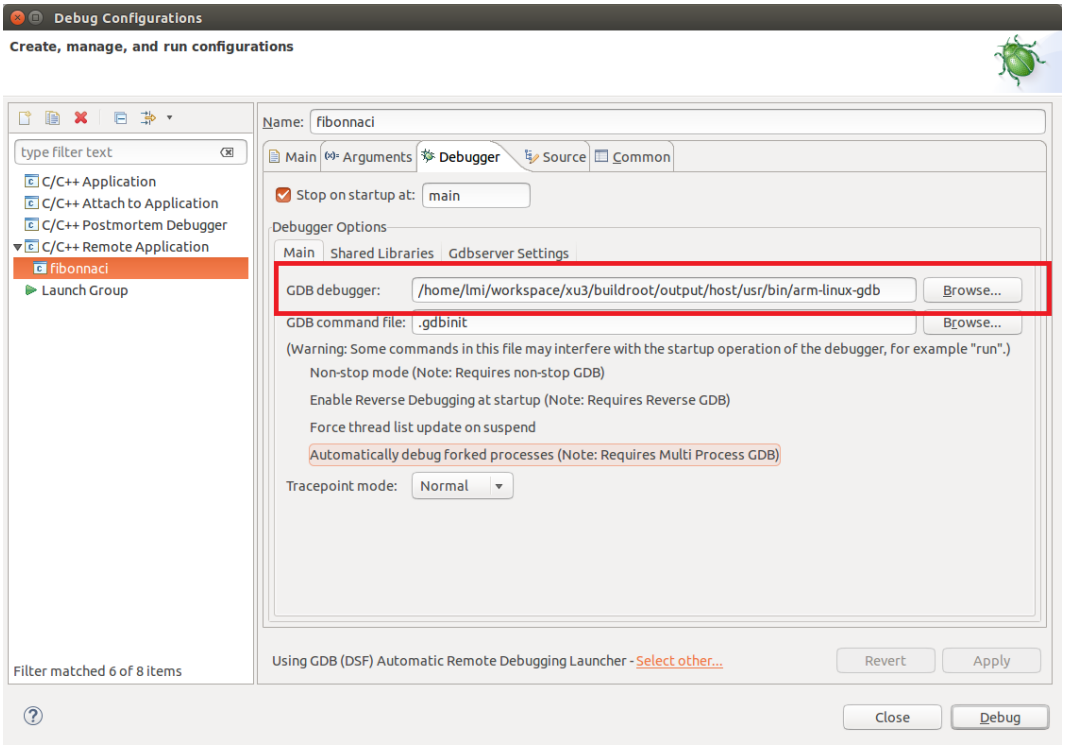

FIGURE 6 – Configuration du server gdb

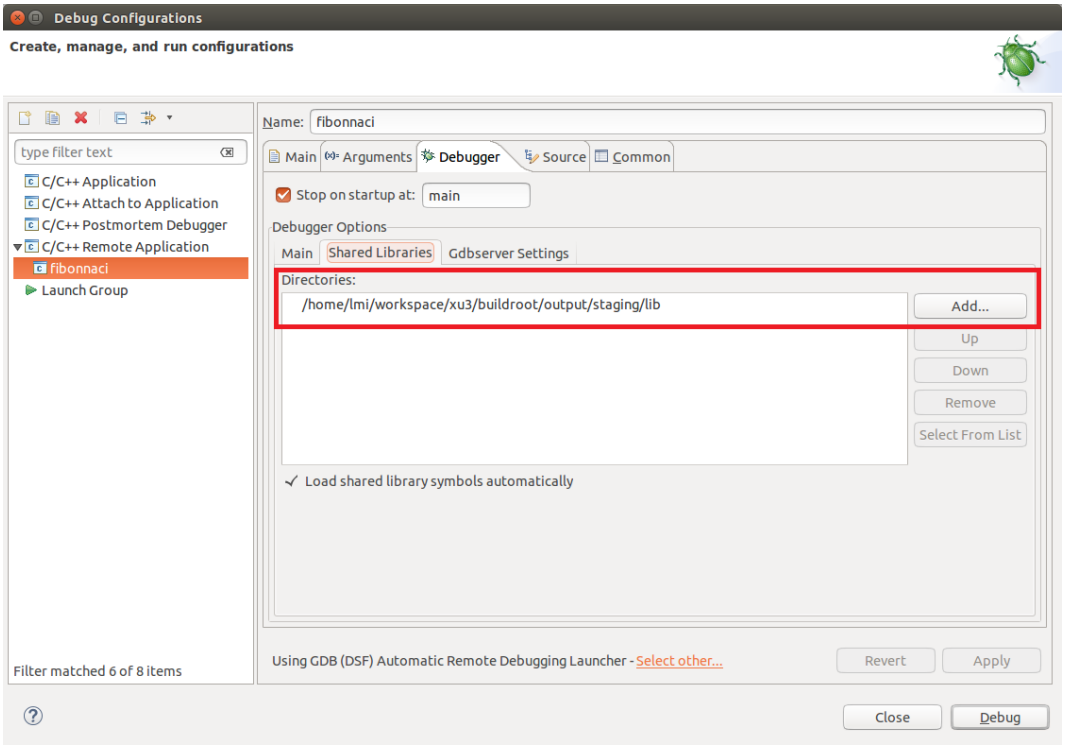

FIGURE 7 – Configuration des shared library

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

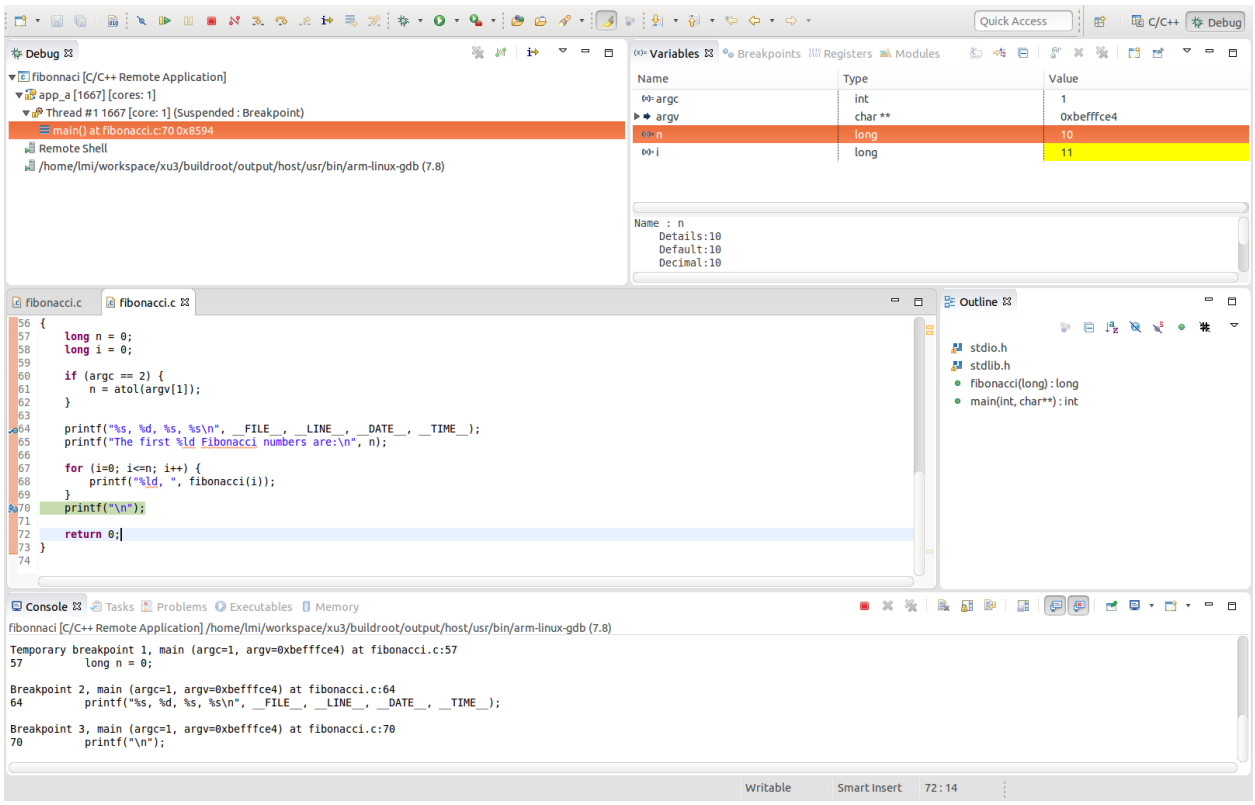

FIGURE 8 – Debug de l'exemple Fibonacci

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 arrêtée avant démarrage

2.6.1 Fibonacci

```
lmi@csel1:~/workspace/csel1/environment/samples/fibonacci$ make clean all

lmi@csel1:~$ ssh root@192.168.0.11
# cd ../usr/workspace/csel1/environment/samples/fibonacci/
# ./app_a 10
fibonacci.c, 64, Oct 31 2015, 15:27:54
The first 10 Fibonacci numbers are:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,
```

2.6.2 Tracing

Avec la trace active

```
lmi@csel1:~/workspace/csel1/environment/samples/tracing$ make DEBUG=1 clean all

# cd ../tracing/
# ./app_a 12
fibonacci.c, 70, Oct 31 2015, 15:35:42
The first 12 Fibonacci numbers are:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
```

Avec la trace inactive

```
lmi@csel1:~/workspace/csel1/environment/samples/tracing$ make clean all

# ./app_a 12
The first 12 Fibonacci numbers are:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
```

2.6.3 Core dumps

```
lmi@csel1:~/workspace/csel1/environment/samples/core_dumps$ make clean all

# cd ../core_dumps/
# ulimit -c unlimited
# ./app_a
Segmentation fault (core dumped)
# gdb app_a core
...
Core was generated by `./app_a'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000083e0 in access_data () at core_dumps.c:31
31      *p=10;
(gdb) bt
#0  0x000083e0 in access_data () at core_dumps.c:31
#1  0x00008424 in call (n=0) at core_dumps.c:37
#2  0x00008420 in call (n=1) at core_dumps.c:36
#3  0x00008420 in call (n=2) at core_dumps.c:36
#4  0x00008420 in call (n=3) at core_dumps.c:36
#5  0x00008420 in call (n=4) at core_dumps.c:36
#6  0x00008420 in call (n=5) at core_dumps.c:36
#7  0x00008420 in call (n=6) at core_dumps.c:36
#8  0x00008420 in call (n=7) at core_dumps.c:36
#9  0x00008420 in call (n=8) at core_dumps.c:36
#10 0x00008420 in call (n=9) at core_dumps.c:36
#11 0x00008420 in call (n=10) at core_dumps.c:36
#12 0x00008448 in main () at core_dumps.c:43
```

2.6.4 Backtrace

Cet exemple s'effectue uniquement sur la machine hôte

```
lmi@csel1:~/workspace/csel1/environment/samples/core_dumps$ cd ../backtrace/
lmi@csel1:~/workspace/csel1/environment/samples/backtrace$ make clean all
...
lmi@csel1:~/workspace/csel1/environment/samples/backtrace$ ./app_h
backtrace() returned 17 addresses
./app_h[0x80484fc]
[0xb7727404]
./app_h[0x804854a]
./app_h[0x8048571]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804856c]
./app_h[0x804859c]
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3)[0xb757ca83]
./app_h[0x8048401]
Segmentation fault (core dumped)
lmi@csel1:~/workspace/csel1/environment/samples/backtrace$ addr2line -e app_h 0
    x804859c
/home/lmi/workspace/csel1/environment/samples/backtrace/main.c:61
```

2.6.5 System calls

```
lmi@csel1:~/workspace/csel1/environment/samples/system_calls$ make clean all

# cd ../system_calls/
# ./app_a
current temperature: 46.00 degree Celcius

# strace ./app_a
execve("./app_a", ["./app_a"], [/* 21 vars */]) = 0
brk(0)                                  = 0x11000
...
+++ exited with 0 +++

# strace -e trace=mmap2 ./app_a
...
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    xb6ff9000
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
    xb6ffa000
current temperature: 47.00 degree Celcius
+++ exited with 0 +++
```

2.6.6 Memory leaks

```
lmi@csel1:~/workspace/csel1/environment/samples/memory_leaks$ make clean all

# cd ../memory_leaks/
# ./app_a
# valgrind --leak-check=full ./app_a
==1714== Memcheck, a memory error detector
==1714== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==1714== Using Valgrind-3.10.0 and LibVEX; rerun with -h for copyright info
==1714== Command: ./app_a
==1714==
==1714==
==1714== HEAP SUMMARY:
==1714==     in use at exit: 31,880 bytes in 3,985 blocks
==1714==   total heap usage: 4,000 allocs, 15 frees, 32,000 bytes allocated
==1714==
==1714== 31,880 (8 direct, 31,872 indirect) bytes in 1 blocks are definitely
    lost in loss record 2 of 2
==1714==    at 0x483535C: malloc (in /usr/lib/valgrind/vgpreload_memcheck-arm-
    linux.so)
==1714==
==1714== LEAK SUMMARY:
==1714==    definitely lost: 8 bytes in 1 blocks
==1714==    indirectly lost: 31,872 bytes in 3,984 blocks
==1714==      possibly lost: 0 bytes in 0 blocks
==1714==    still reachable: 0 bytes in 0 blocks
==1714==         suppressed: 0 bytes in 0 blocks
==1714==
==1714== For counts of detected and suppressed errors, rerun with: -v
==1714== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

2.7 Mise en production de l'ODROID-XU3

```
lmi@csel1:~/workspace/csel1/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 :

```
#!/bin/sh
#
# Daemon application
#
```

```
case "$1" in
  start)
    /usr/workspace/csel1/environment/samples/daemon/app_a
    ;;
  stop)
    killall app_a
    ;;
  restart|reload)
    killall app_a
    /usr/workspace/csel1/environment/samples/daemon/app_a
    ;;
  *)
    echo $"Usage: $0 {start|stop|restart}"
    exit 1
esac

echo "Daemon application launched"

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.

```
# cp /usr/workspace/csel1/environment/samples/daemon/S60appl /etc/init.d
#reboot
...
Starting logging: OK
Starting mdev...
Initializing random number generator... done.
Starting network...
ip: RTNETLINK answers: File exists
Starting sshd: OK
Daemon application launched
[    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 :

```
$ make clean all
```

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

```
$ modinfo mymodule.ko
filename:       /home/lmi/workspace/csel1/environment/module_noyau/exercice1/
    pointa/mymodule.ko
license:        GPL
description:    Module Skeleton
author:         Emilie Gsponer
depends:
vermagic:       3.10.63 SMP preempt mod_unload ARMv7 p2v8
```

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

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

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

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

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

```
# cat /proc/modules
mymodule 687 0 - Live 0xbf004000 (O)

# lsmod mymodule.ko
Module                  Size  Used by    Tainted: G
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.

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

```
MODPATH := /tftpboot/odroidxu3

install:
$(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 :

```
$ make clean all
$ sudo make install
```

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

```
lmi@csel1:~/.ssh$ sudo minicom
...
ODROIDXU3> run nfsboot
(Re)start USB...
...
Welcome to Hardkernel ODROID_XU3 board
odroidxu3 login: root
# modprobe mymodule
[   98.024580] [c2] Linux module skeleton loaded
# modprobe -r mymodule
[   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.

```
# modprobe mymodule
[  922.323941] [c1] Linux module skeleton loaded
[  922.326835] [c1] text:hello
[  922.326835] elements:2
# modprobe -r mymodule
[  925.223513] [c0] Linux module skeleton unloaded
# modprobe mymodule text="hello world" elements=10
[ 1196.597223] [c2] Linux module skeleton loaded
[ 1196.600182] [c2] text:hello world
[ 1196.600182] elements:10
# modprobe -r mymodule
[ 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 :

```
$ make clean all
$ 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 :

```
# cat /proc/sys/kernel/printk
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.

```
# echo 10 > /proc/sys/kernel/printk
# cat /proc/sys/kernel/printk
10      4       1       7
# modprobe mymodule
[ 2139.129459] [c2] Linux module skeleton loaded
[ 2139.132348] [c2] text:hello
[ 2139.132348] elements:2
# modprobe -r mymodule
[ 2143.683511] [c0] Linux module skeleton unloaded

# echo 1 > /proc/sys/kernel/printk
# cat /proc/sys/kernel/printk
1       4       1       7
# modprobe mymodule
# modprobe -r mymodule

# echo 10 > /proc/sys/kernel/printk
# modprobe mymodule
[ 2173.021095] [c2] Linux module skeleton loaded
```

```
[ 2173.024050] [c2] text:hello
[ 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 chainés 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 :

```
# modprobe mymodule element_num=5 text="Hello world"
[ 2250.418060] [c2] New element added to the list
[ 2250.421107] [c2] New element added to the list
[ 2250.425536] [c2] New element added to the list
[ 2250.429937] [c2] New element added to the list
[ 2250.434368] [c2] New element added to the list
[ 2250.438773] [c2] ID of element : 0
[ 2250.438773] String is : Hello world
[ 2250.445642] [c2] ID of element : 1
[ 2250.445642] String is : Hello world
[ 2250.452489] [c2] ID of element : 2
[ 2250.452489] String is : Hello world
[ 2250.459415] [c7] ID of element : 3
[ 2250.459415] String is : Hello world
[ 2250.466444] [c7] ID of element : 4
[ 2250.466444] String is : Hello world
# modprobe -r  mymodule
[ 2267.933575] [c1] Element poped
[ 2267.935161] [c1] Element poped
[ 2267.938196] [c1] Element poped
[ 2267.941283] [c1] Element poped
[ 2267.944313] [c1] Element poped
[ 2267.947308] [c1] Module removed
```

3.2.2 Exercice 5

Donnée : Indiquer les différents alocateurs 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_jUWHIfO1qFIIikEyHA

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 déboggage "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 :

```
# modprobe mymodule
[   30.336553] [c5] Linux module skeleton loaded
[   30.339495] [c5] Memory allocated
[   30.342733] [c5] uP register: Bit 31..12 : product id=0x65422
[   30.348494] [c5] uP register: Bit 11..8  : package id=0x0
[   30.353877] [c5] uP register: Bit 7..4   : major revision=0x0
[   30.359597] [c5] uP register: Bit 3..0   : minor revision=0x1
# cat /proc/iomem
03000000-03048fff : /lpass@03810000
03810000-038100ff : /lpass@03810000
03830000-038300ff : samsung-i2s
03860000-03860fff : /pinctrl@03860000
03880000-03880fff : /amba/adma@03880000
03880000-03880fff : /amba/adma@03880000
10000000-100000ff : uP register
```

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

```
# modprobe mymodule
[ 1434.469821] [c2] Thread created
# [ 1439.473573] [c1] Thread awake
[ 1444.478588] [c2] Thread awake
[ 1449.483589] [c3] Thread awake
# modprobe -r  mymodule
[ 1454.488597] [c0] Thread awake
[ 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 :

```
# modprobe mymodule
[ 1523.052680] [c2] Init wait queue
[ 1523.054851] [c2] Threads created
# [ 1528.058588] [c3] Thread2 (notif each 5s) awake
[ 1528.061596] [c1] Thread1 (wait notif) awake
[ 1533.063590] [c0] Thread2 (notif each 5s) awake
[ 1533.066587] [c1] Thread1 (wait notif) awake
[ 1538.068588] [c0] Thread2 (notif each 5s) awake
[ 1538.071587] [c1] Thread1 (wait notif) awake
# modprobe -r  mymodule
[ 1543.073563] [c0] Thread2 (notif each 5s) awake
[ 1543.076562] [c2] Thread1 (wait notif) awake
[ 1548.078569] [c0] Thread2 (notif each 5s) awake
[ 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 :

```
gpio_resquest(EXYNOS5_GPX<gpio_nr>(<pin-nr>));
```

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

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

```
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 :

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

Remarque : On obtient bien le même id qu'avec le module noyau de l'exercie 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 :

```
# modprobe mymodule
[  854.599130] [c2] mod: successfully loaded with major 249
# mknod /dev/mod c 249 0
# ./app_a /dev/mod
[ 1166.619318] [c0] mod: open
[ 1166.620800] [c0] mod: mmap
[ 1166.623246] [c0] VMA open, virt b6f71000, phys 10000000
[ 1166.628986] [c0] VMA close.
[ 1166.631259] [c0] mod: release
file /dev/mod open
Physical memory: Bit 31..12 : product id=0x65422
Physical memory: Bit 11..8  : package id=0x0
Physical memory: Bit 7..4   : major revision=0x0
Physical memory: Bit 3..0   : minor revision=0x1

Virtual memory: Bit 31..12 : product id=0x43108
Virtual memory: Bit 11..8  : package id=0x3
Virtual memory: Bit 7..4   : major revision=0x2
Virtual memory: Bit 3..0   : minor revision=0xa
file /dev/mod close
# modprobe -r mymodule
[ 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 :

```
$ make clean all
$ sudo make install
```

```
# modprobe mymodule
[ 4107.528630] [c2] mod: successfully loaded with major 249
# mknod /dev/mod c 249 0
# echo -n test module > /dev/mod
[ 4134.907095] [c1] mod: open()
[ 4134.908540] [c1] mod: write test module
[ 4134.912539] [c1] mod: release()
# cat /dev/mod
[ 4145.581863] [c0] mod: open()
[ 4145.583312] [c0] mod: read test module
[ 4145.587140] [c0] mod: read test module
[ 4145.590820] [c0] mod: release()
# modprobe -r mymodule
[ 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 :

```
# mknod /dev/mydev_0 c 248 0
# mknod /dev/mydev_1 c 248 1
#
#
# echo -n "bonjour, monsieur" > /dev/mydev_0
```

```
[12232.424660] [c1] skeleton : open operation, major version : 248, minor
    version : 0
[12232.430827] [c1] skeleton : opened for reading and writing...
[12232.436572] [c1] minor : 0
[12232.436572] count = 17
[12232.441568] [c1] skeleton : efault value=-14
[12232.445802] [c1] skeleton : write operation... written=17
[12232.451198] [c1] skeleton : release operation
# echo -n "bien le bonjour, mademoiselle" > /dev/mydev_1
[12254.024382] [c1] skeleton : open operation, major version : 248, minor
    version : 1
[12254.030622] [c1] skeleton : opened for reading and writing...
[12254.036288] [c1] minor : 1
[12254.036288] count = 29
[12254.041274] [c1] skeleton : efault value=-14
[12254.045522] [c1] skeleton : write operation... written=29
[12254.050909] [c1] skeleton : release operation
#
#
# cat /dev/mydev_0
[12278.473668] [c0] skeleton : open operation, major version : 248, minor
    version : 0
[12278.479815] [c0] skeleton : opened for reading and writing...
[12278.485576] [c0] skeleton : read operation... read=17
[12278.490663] [c6] skeleton : read operation... read=0
[12278.495527] [c6] skeleton : release operation
bonjour, monsieur#
#
#
# cat /dev/mydev_1
[12291.694063] [c0] skeleton : open operation, major version : 248, minor
    version : 1
[12291.700284] [c7] skeleton : opened for reading and writing...
[12291.706190] [c7] skeleton : read operation... read=29
[12291.710985] [c7] skeleton : read operation... read=0
[12291.715919] [c7] skeleton : release operation
bien le bonjour, mademoiselle#
#
```

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 :

```
lmi@csel1:~/workspace/csel1/environment/peripheral/exercice5/user$ make clean
    all
lmi@csel1:~/workspace/csel1/environment/peripheral/exercice5/user$ cd ../noyau/
lmi@csel1:~/workspace/csel1/environment/peripheral/exercice5/noyau$ make clean
    all
lmi@csel1:~/workspace/csel1/environment/peripheral/exercice5/noyau$ sudo make
    install
```

```
# modprobe mymodule
[  231.111019] [c2] mod: successfully loaded with major 249
# mknod /dev/mod c 249 0
# ./app_a /dev/mod HELLO
[  234.969067] [c0] mod: open()
[  234.970719] [c0] mod: write HELLO
[  234.973927] [c0] mod: read HELLO
[  234.977039] [c0] mod: release()
file /dev/mod open
write HELLO
read HELLO
file /dev/mod close
# modprobe -r mymodule
[  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 ODROID-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.

Remarque : L'application utilisateur attend de recevoir 5 interruptions des boutons avant s'arrêter.

Emplacement du code :
/PilotesPeripheriques/exercice6-SelectButton/user
/PilotesPeripheriques/exercice6-SelectButton/noyau

Exécution du code :

```
# modprobe mymodule
[   90.125005] [c2] mod: successfully loaded with major 249
[   90.129074] [c2] Interrupt handler module loaded in kernel
```

```
[    90.134167] Configuring pins[    90.137059] _gpio_request: gpio-176 (SW1)
    sta6
[    90.142048] [c2] _gpio_request: gpio-177 (SW2) status -16
[    90.147391] [c2] _gpio_request: gpio-168 (SW3) status -16
[    90.152778] [c2] _gpio_request: gpio-164 (SW4) status -16
[    90.158239] [c4] Configuring switches interrupts
[    90.162625] Pins and interrupts have been configured.[    90.167598] Init
    waie
# ./app_a /dev/mod
[    97.905744] [c0] mod: open
[    97.907156] [c0] Threads started
file /dev/mod open
[   102.317965] [c0] Some switch has been pressed
[   102.320893] [c2] Thread awake
Interrupt 1
[   103.728868] [c0] Some switch has been pressed
[   103.731800] [c0] Thread awake
Interrupt 2
[   104.686291] [c0] Some switch has been pressed
Interrupt 3
[   106.084775] [c0] Some switch has been pressed
[   106.087702] [c1] Thread awake
Interrupt 4
[   107.048387] [c0] Some switch has been pressed
[   107.051374] [c1] mod: release
Interrupt 5
[   107.707804] [c0] Some switch has been pressed
[   107.710718] [c0] Thread awake
[   107.713669] [c1] Threads stopped
file /dev/mod close
# modprobe -r mymodule
[   119.950452] [c0] Freeing interrupts
[   119.952344] Interrupts freed[   119.955165] mod: successfully unloaded
```

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 :

```
# echo -n "salut" > /sys/devices/platform/skeleton/attr
#
```

```
#
# cat /sys/devices/platform/skeleton/attr
salut#
```

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 :

5 Programmation système : Système de fichiers

5.1 Contexte

La carte ODROID-XU3 Lite est équipé d'un petit ventilateur afin d'évacuer la chaleur produite par l'activité du microprocesseur. La vitesse du ventilateur est contrôlée par un PWM (Pulser-Width Modulation), dont le rapport de cycle (duty) dépendra de la température du microprocesseur. Le ventilateur sera régulé sur la base de 20kHz. Sur la carte ODROID-XU3 Lite ce PWM peut être réalisé avec la porte d'entrée/sortie "gpb2_0".

5.2 Travail à réaliser

Sur le site moodle vous trouverez une petite application qui contrôle la vitesse du ventilateur. Ce code n'a pas été très bien programmé et utilise le 100% d'un cœur du processeur (à

mesurer avec top). Concevez une application permettant de gérer la vitesse de rotation du ventilateur de l'ODROID-XU3 à l'aide des trois boutons poussoir. Quelques indications pour la réalisation de l'application :

1. Au démarrage le duty cycle du ventilateur sera réglé à 50%
2. Utilisation des boutons poussoir
   (a) « sw1 » pour augmenter la vitesse du ventilateur
   (b) « sw2 » pour remettre la vitesse du ventilateur à sa valeur initiale
   (c) « sw3 » pour diminuer la vitesse du ventilateur
   (d) une pression continue exercée sur un bouton indiquera une auto incrémentation ou décrémentation du duty cyle.
3. Tous les changements de vitesse du ventilateur seront logger avec syslog de Linux.
4. Le multiplexage des entrées/sorties devra être utilisé.

5.3 Travail réalisé

5.3.1 Description

Tous les points de la donnée ont été implémentés. La fréquence du ventilateur a été augmentée à 50 kHz, car à 20kHz le ventilateur émet un son strident très agaçant.
Pour le point de l'auto incrémentation, la duty cycle est augmenté/diminué de 2% chaque 40 us quand un bouton est maintenu appuyé. Si par hasard, on maintient appuyé deux boutons en même temps, le programme attend qu'un des deux soit relâché pour changer le duty cycle.
En plus des points demandés, les LEDs de la carte d'extensions sont utilisées.

1. LED1 : S'allume et s'éteint en fonction de la pression sur le switch 1.
2. LED2 : Clignote à la fréquence de la PWM. La vitesse est trop rapide pour la voir clignoter, on peut par contre observer un changement de luminosité entre un duty cycle proche de 0% et un proche de 100%.
3. LED3 : S'allume et s'éteint en fonction de la pression sur le switch 3.

Emplacement du code : */FanControl*

5.3.2 Configuration des GPIO

Une des recherches à faire pour ce projet était de trouvé à quelle GPIO correspondait les boutons, les LEDs et le ventilateur.
Pour cela, la schématique de la carte d'extension a été utilisée.

Source : http://dn.odroid.com/ODROID-XU/Expansion_Board/ExpansionBoard.pdf

Si on prend comme exemple la LED D1, on trouve dans le schéma qu'elle correspond à la pin XE.INT23. Pour obtenir la GPIO correspondante, cette page a été très utile.

Source : http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu

On apprend grâce à ce site que la pin XE.INT23 correspond au GPX2.7.
Pour trouver le numéro de GPIO correspondant, il suffit d'entrer les commandes suivantes sur la cible :

```
# mount -t debugfs none /sys/kernel/debug
# cat /sys/kernel/debug/gpio
...
GPIOs 16-23, platform/13400000.pinctrl, gpx1:
gpio-17  (dwc3_id_gpio         ) in  hi
gpio-19  (sysfs                ) out lo
gpio-22  (sysfs                ) in  hi
gpio-23  (scl                  ) out hi

GPIOs 24-31, platform/13400000.pinctrl, gpx2:
gpio-27  (red:activity         ) out lo
gpio-28  (sysfs                ) out hi
gpio-29  (sysfs                ) in  hi
gpio-30  (sysfs                ) in  hi
gpio-31  (sysfs                ) out lo
...
```

Ces commandes nous renseignent sur le numéro de GPIO. Si l'on reprend l'exemple de la LED D1, on va lire que le GPX2 correspond au numéro 24. Comme la LED est sur le GPX2.7, il faut encore rajouter l'offset. On obtient donc 24+7 = **31** comme numéro de GPIO pour la LED D1.
Pour configurer les entrées/sorties, le code pour atteindre la pwm fourni dans l'exemple a été repris. Pour les boutons ils suffit d'ajouter le flanc de détection pour l'interruption.

5.3.3 Exécution du code

Le code s'exécute dans l'espace utilisateur et non plus dans le noyau comme les exercices précédents. Le code doit être compilé sur l'hôte et lancé sur la cible (en mode nfs). L'application n'affiche rien directement dans la console, tout est dans le syslog.

```
lmi@csel1:~/workspace/csel1/environment/fanCtrl$ make clean all
```

```
# cd /usr/workspace/csel1/environment/fanCtrl/
# ./app_a
```

5.3.4 Syslog

Une fois l'application lancée, on peut aller voir le *syslog*. Le duty cycle du ventilateur y est affiché chaque fois qu'il change. Comme l'application est lancée depuis la connexion sérielle, il faut accéder à la cible par connexion ssh. Chaque pression sur un des boutons fera afficher une nouvelle ligne.

```
lmi@csel1:~$ ssh root@192.168.0.11
# tail -f /var/log/messages
Jan  1 00:06:37 odroidxu3 local1.notice fan control[1625]: duty :64
Jan  1 00:06:37 odroidxu3 local1.notice fan control[1625]: duty :66
Jan  1 00:06:37 odroidxu3 local1.notice fan control[1625]: duty :68
Jan  1 00:06:37 odroidxu3 local1.notice fan control[1625]: duty :70
Jan  1 00:06:37 odroidxu3 local1.notice fan control[1625]: duty :72
...
```

5.3.5 Mesure de performance

La commande *top* permet de mesurer les performances de l'application. Le code fourni en exemple utilise 100% d'un cœur du processeur.

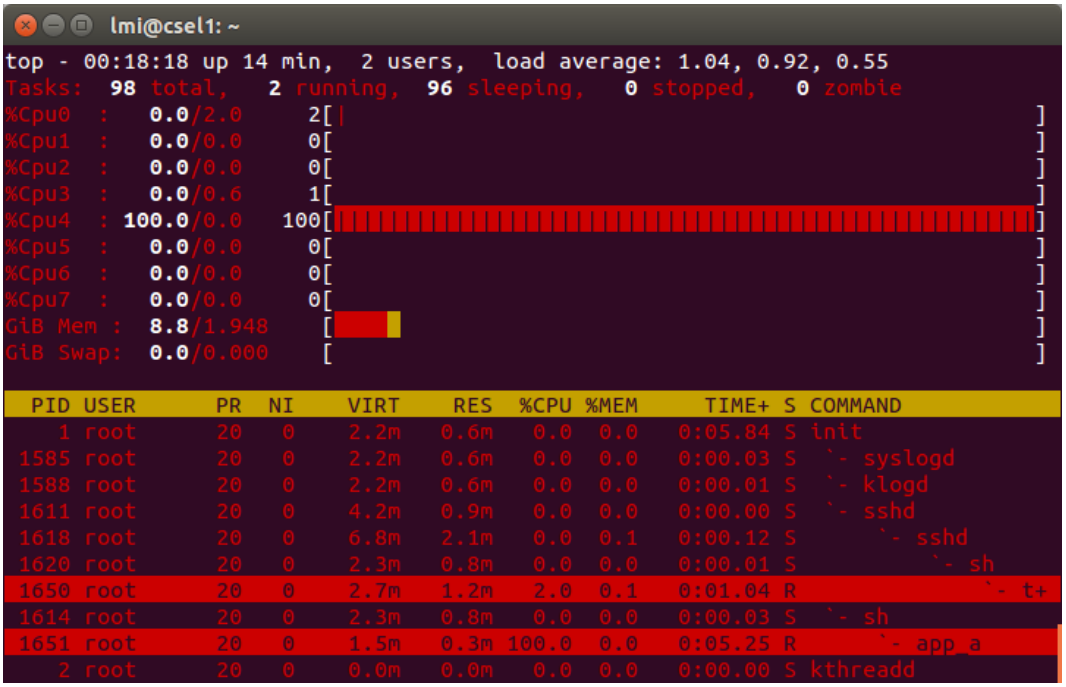

FIGURE 10 – Occupation du cœur avant modifications

L'image ci-dessous montre que notre application est meilleure, elle consomme moins de 3% d'un cœur.

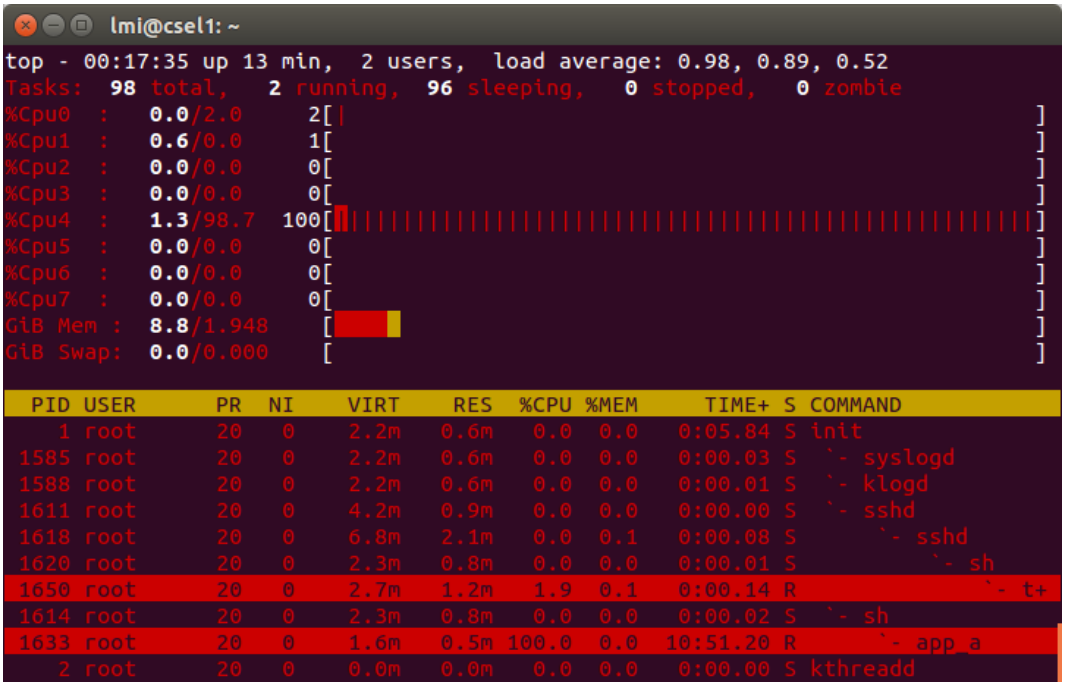

FIGURE 11 – Occupation du cœur après modifications

5.3.6 Amélioration possibles

Les switch n'ont pas d'anti-rebond. Parfois, l'état du bouton du programme ne correspond pas à l'état physique du bouton. La conséquence est que le duty cylcle s'incrémente/décrémente alors qu'aucun bouton n'est appuyé. Il faudrait implémenter un anti-rebond software. Ce point n'a pas été réalisé, car ce n'était pas l'objectif du labo. Si le problème se présente, il suffit de relancer le programme.

6 Programmation système : Multiprocessing

Warning : Pour cette exercice, il a fallu reconfigurer le noyau linux en activant certaines options. Pour cela, se référer à la slide 72 du support de cours. Cette opération prend un temps considérable.

6.1 Processus, signaux et communication

6.1.1 Exercice 1

Donnée : Concevez et développez une petite application mettant en œuvre un des services de communication proposé par Linux (p.ex. « socketpair ») entre un processus parent et un processus enfant. Le processus enfant devra émettre quelques messages sous forme de texte vers le processus parent, lequel les affichera sur la console. Le message « exit » permettra de terminer l'application. Cette application devra impérativement capturer tous les signaux et les ignorer. Seul un message d'information sera affiché sur la console.

Emplacement du code : */Multiprocessing/Exercice1-Comm*

Exécution du code :
Le processus enfant demande à l'utilisateur de tapper quelque chose, qui est ensuite retransmis au parent, qui l'affiche.

```
lmi@csel1:~$ cd workspace/csel1/environment/multiproc/exercice1
lmi@csel1:~/workspace/csel1/environment/multiproc/exercice1$ make clean all
```

```
# cd /usr/workspace/csel1/environment/multiproc/exercice1/
# ./app_a
In the main process.

In the parent process

In the children process.

Enter a msg to send to the parent :
Hello
Hello will be written to the parent.
Enter a msg to send to the parent :
Received 50 from the children. String is : Hello
Bla
Bla will be written to the parent.
Enter a msg to send to the parent :
Received 50 from the children. String is : Bla
Received signal is : Interrupt
Received signal is : Interrupt
Bla will be written to the parent.
Enter a msg to send to the parent :
Received 50 from the children. String is : Bla
Received signal is : Stopped
Received signal is : Stopped
Bla will be written to the parent.
Enter a msg to send to the parent :
Received 50 from the children. String is : Bla
exit
exit will be written to the parent.
Received 50 from the children. String is : exit
#
```

Ce second output montre que le SIGINT est géré pour afficher un message en écrivant *Received signal is :* suivi du type de signal. Dans le cas suivant, c'est un ctrl-c qui a été envoyé.

```
Enter a msg to send to the parent : Received 50 from the children. String is :
^CReceived signal is : Interrupt
Received signal is : Interrupt
will be written to the parent.
Enter a msg to send to the parent : Received 50 from the children. String is :
```

6.2 CGroups

6.2.1 Exercice 2

Donnée : Concevez une petite application permettant de valider la capacité des groupes de contrôle de limiter l'utilisation de la mémoire.

Emplacement du code : */Multiprocessing/Exercice2-cgroups*

Réponse aux questions :

a.Quel effet a la commande «echo $$ > ...» sur les cgroups ?
Par cette commande, on demande aux cgroups de surveiller la tache représentée par le terminal de l'ordroid. Comme le programme que nous lançons depuis la ligne de commande devient le fils du terminal dans lequel il est lancé, on surveille aussi le fils car il est automatiquement affecté à la hiérarchie du parent.

b.Quel est le comportement du sous-système «memory» lorsque le quota de mémoire est épuisé ? Pourrait-on le modifier ? Si oui, comment ?
Le comportement par défaut est de tuer le processus dépassant les limites. Mais il est possible de modifier ce comportement en mettant un **1** dans «memory.oom_control». Si la modification a été apportée, le processus qui viendrait à dépasser les limites de mémoire se verrait mis en pause.

L'output en dessous montre qu'une fois le programme *app_a* lancé, après vingt allocations, il est tué à cause de son excès de consommation mémoire.

```
# ./app_a
Ten more megas allocated

Ten more megas allocated

Killed
```

c.Est-il possible de surveiller/vérifier l'état actuel de la mémoire ? Si oui, comment ?
Plusieurs commandes peuvent être utilisées pour visualiser l'état de la mémoire :

- *free -m*
- *cat /proc/meminfo*

La première produit ce genre d'output :

```
# free -m
total        used        free      shared  buff/cache   available
Mem:           1990         156        1808           0          24        1813
Swap:             0           0           0
```

Tandis que la seconde produit ça :

```
# cat /proc/meminfo
MemTotal:        2038540 kB
MemFree:         1852220 kB
Buffers:               0 kB
Cached:             5272 kB
SwapCached:            0 kB
Active:             3308 kB
Inactive:           2952 kB
Active(anon):       1012 kB
Inactive(anon):       88 kB
Active(file):       2296 kB
Inactive(file):     2864 kB
Unevictable:           0 kB
Mlocked:               0 kB
HighTotal:       1296384 kB
HighFree:        1147292 kB
LowTotal:         742156 kB
LowFree:          704928 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:          1000 kB
Mapped:             2084 kB
Shmem:               112 kB
Slab:              20200 kB
SReclaimable:       9264 kB
SUnreclaim:        10936 kB
KernelStack:         840 kB
PageTables:          124 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     1019268 kB
Committed_AS:       5752 kB
VmallocTotal:     245760 kB
VmallocUsed:       16520 kB
VmallocChunk:     103580 kB
```

Si on veut monitorer la consommation de notre application, on peut aussi utiliser Valgrind et l'outil *massif* avec la commande suivante :

```
valgrind --tool=massif ./app_a
```

Le programme s'execute normalement et l'outil de visualisation produit un fichier contenant tout le logging de la mémoire utilisée par l'application. On peut la voir avec la commande :

```
ms_print massif.out.<app_a pid>
```

Ce qui produit dans notre cas :

```
--------------------------------------------------------------------------------


Command:            ./app_a
Massif arguments:   (none)
ms_print arguments: massif.out.1695
--------------------------------------------------------------------------------



MB
47.68^                                                                      :#
|                                                                        ::@:#
|                                                                    :::::  @:#
|                                                                :::::  ::  @:#
|                                                            ::::::  ::  ::  @:#
|                                                        :@@::  :::  ::  ::  @:#
|                                                    :::::@ ::  :::  ::  ::  @:#
|                                                :::::  ::@ ::  :::  ::  ::  @:#
|                                            ::::::  ::  ::@ ::  :::  ::  ::  @:#
|                                        :@@:::  ::  ::  ::@ ::  :::  ::  ::  @:#
|                                    ::::::@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|                                :::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|                            :::::::::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|                        ::@:  ::  :::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|                    :::::  @:  ::  :::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|                ::::  :::  @:  ::  :::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|            :::::  ::  :::  @:  ::  :::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|        ::@::  ::  ::  :::  @:  ::  :::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|    :::: @::  ::  ::  :::  @:  ::  :::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
|   ::: :: @::  ::  ::  :::  @:  ::  :::  :: :@ :::  ::  ::  ::@ ::  :::  ::  ::  @:#
0 +-------------------------------------------------------------------------------->Mi
0                                                                            13.57

...
```

Pour nous rassurer, avec la commande et l'output suivant, on constate quand même que la mémoire a bien été libérée :

```
# valgrind ./app_a
==1701== Memcheck, a memory error detector
==1701== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==1701== Using Valgrind-3.10.0 and LibVEX; rerun with -h for copyright info
==1701== Command: ./app_a
==1701==
Ten more megas allocated

Ten more megas allocated

Ten more megas allocated

Ten more megas allocated

Press enter to continue...

```

```
17
18 Ten more megas freed
19
20 Ten more megas freed
21
22 Ten more megas freed
23
24 Ten more megas freed
25
26 Ten more megas freed
27
28 ==1701==
29 ==1701== HEAP SUMMARY:
30 ==1701==     in use at exit: 0 bytes in 0 blocks
31 ==1701==   total heap usage: 50 allocs, 50 frees, 50,000,000 bytes allocated
32 ==1701==
33 ==1701== All heap blocks were freed -- no leaks are possible
34 ==1701==
35 ==1701== For counts of detected and suppressed errors, rerun with: -v
36 ==1701== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Pour les derniers outputs, la limite d'allocation mémoire a été élevée à 100MB. Aucune solution n'a été trouvée autrement pour monitorer la mémoire du processus durant son exécution avec le contrôle de la mémoire. Valgrind est autant tué que l'application, sinon.

6.2.2 Exercice 3

Donnée : Afin valider la capacité des groupes de contrôle de limiter l'utilisation des CPU, concevez une petite application composée au minimum de 2 processus utilisant le 100% des ressources du processeur.

Emplacement du code : */Multiprocessing/Exercice3-CPU*

Monter les cgroup :

```
1  # mount -t tmpfs none /sys/fs/cgroup
2  # mkdir /sys/fs/cgroup/memory
3  # mount -t cgroup -o memory memory /sys/fs/cgroup/memory
4  # mkdir /sys/fs/cgroup/memory/mem
5  # mkdir /sys/fs/cgroup/cpuset
6  # mount -t cgroup -o cpu,cpuset cpuset /sys/fs/cgroup/cpuset
7  # mkdir /sys/fs/cgroup/cpuset/high
8  # mkdir /sys/fs/cgroup/cpuset/low
9  # echo 4 > /sys/fs/cgroup/cpuset/high/cpuset.cpus
10 # echo 0 > /sys/fs/cgroup/cpuset/high/cpuset.mems
11 # echo 3 > /sys/fs/cgroup/cpuset/low/cpuset.cpus
12 # echo 0 > /sys/fs/cgroup/cpuset/low/cpuset.mems
```

Réponse aux questions :

a. Les 4 dernières lignes sont obligatoires pour que les prochaines commandes fonctionnent correctement. Pouvez-vous en donner la raison ?

Les deux commandes écrivant dans le cpuset.cpus assignent un numéro de cœur pour le cpuset. Les autres écrivant dans le cpuset.mems limite l'espace mémoire et le crée. Cela crée l'architecture du CGroups. Pour l'avoir testé, si on n'affecte pas un cpuset.mems, l'application n'a pas de place en mémoire.

```
# echo $$ > /sys/fs/cgroup/cpuset/low/tasks
sh: write error: No space left on device
```

b. Ouvrez deux shells distincte et placez en une dans le cgroup high et l'autre dans le cgroup low.
Lancez ensuite votre application dans chacune des shells.
Quel devrait être le bon comportement ? Pouvez-vous le vérifier ?

Installation de l'application avec la connexion sérielle :
Oui, on peut le vérifier, les commandes sont visibles un peu plus loin. Le comportement attendu est une utilisation à 100% des cœurs 3 et 4 (attribués lors de la création du CGroup) du processeur avec 50% pour le parent et 50% pour le parent. Pour l'instant, on n'a pas assigné de limitation dans l'utilisation du temps CPU.

```
lmi@csel1:~$ sudo minicom
...
# echo $$ > /sys/fs/cgroup/cpuset/high/tasks
# ./app_a
In the main process.

In the parent process

In the children process.
```

Installation de l'application avec la connexion ssh :

```
lmi@csel1:~$ ssh root@192.168.0.11
# echo $$ > /sys/fs/cgroup/cpuset/low/tasks
# cd /usr/workspace/csel1/environment/multiproc/exercice3/
# echo $$ > /sys/fs/cgroup/cpuset/low/tasks
# ./app_a
In the main process.

In the parent process

In the children process.
```

L'utilisation du CPU peut être affichée avec la commande top en lançant l'application en arrière plan :

```
# ./app_a &
# In the main process.

In the parent process

In the children process.

# top
```

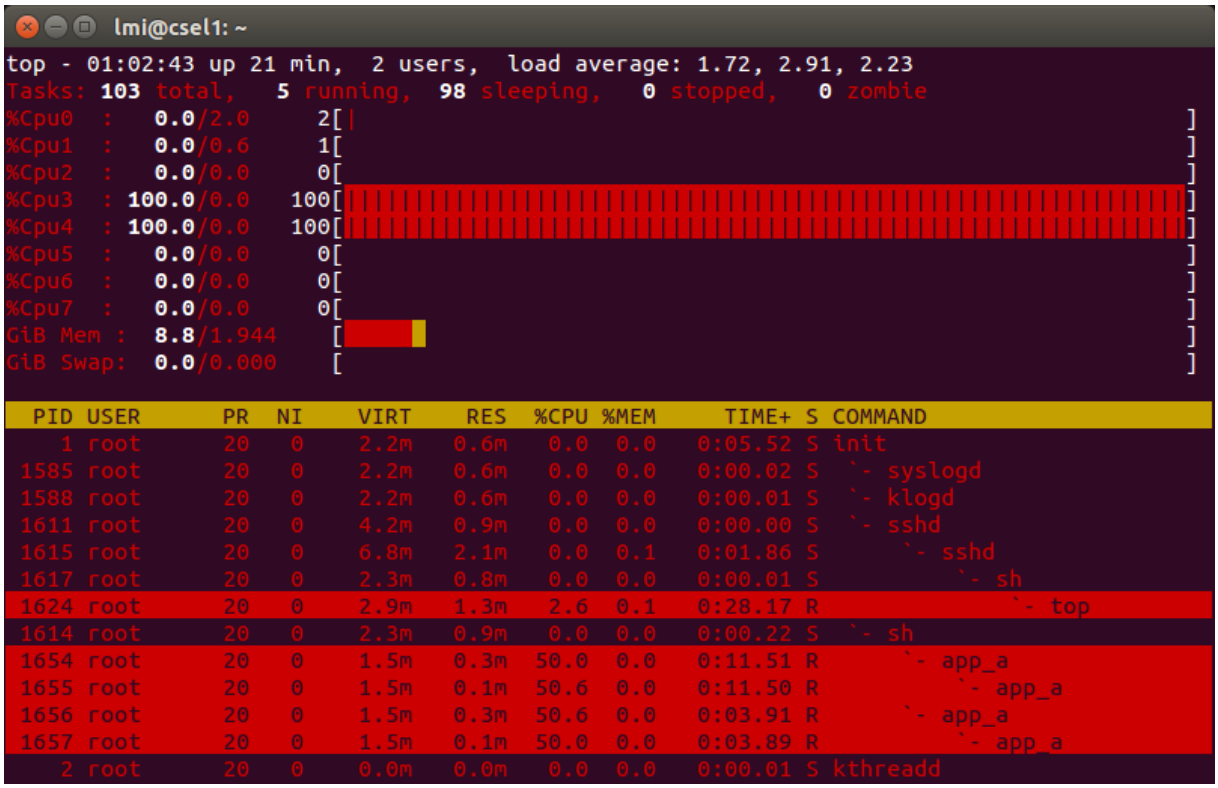

FIGURE 12 – Utilisation du processeur

c. Sachant que l'attribut cpus.shares permet de répartir le temps CPU entre différents cgroups, comment devrait-on procéder pour lancer deux tâches distinctes sur le cœur 6 de notre processeur et attribuer 75% du temps CPU à la première tâche et 25% à la deuxième ?

Voici les commandes utilisées pour la configuration demandée. Par défaut, l'attribut cpu.shares contient 1024, qui représente 50% du temps CPU. Pour plus de lisibilité, l'application a été compilée une fois sous le nom app_a et une fois sous le nom app_b.

Dans la connection ssh :

```
# top
```

Dans la connection série :

```
# echo 512 > /sys/fs/cgroup/cpuset/low/cpu.shares
# echo 1536 > /sys/fs/cgroup/cpuset/high/cpu.shares
# echo $$ > /sys/fs/cgroup/cpuset/low/tasks
# ./app_b &
# In the main process.

In the parent process

In the children process.

# echo $$ > /sys/fs/cgroup/cpuset/high/tasks
# ./app_a &
# In the main process.

In the parent process

In the children process.

#
```

Voici le résultat obtenu, l'app_a prend 75% du temps CPU (37.7% parent, 37.7% enfant) et l'app_b 25%.

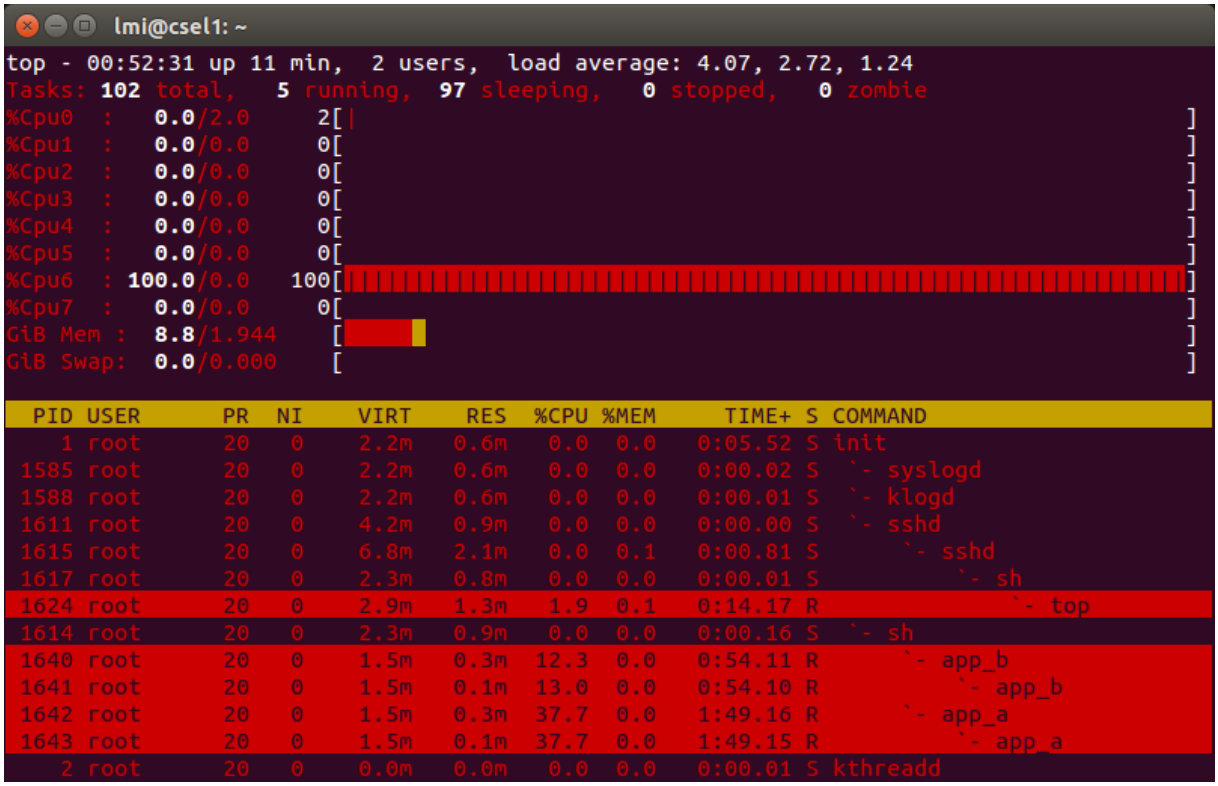

FIGURE 13 – Répartition du temps processeur

7 Optimisation : Performances

7.1 Installation de perf

7.2 Prise en main de perf

7.2.1 Exercice 1

Donnée : Décrivez brièvement ce que sont les évènements suivants :

1. instructions :
2. cache-misses :
3. branch-misses :
4. L1-dcache-load-misses :
5. cpu-migrations :
6. context-switches :

7.2.2 Exercice 2

Donnée : Compilez (en utilisant le Makefile fourni) et exécutez le programme situé dans le dossier exercices/01 en utilisant la commande “perf stat”

Exécution du code :

7.2.3 Exercice 3

Donnée : Ouvrez maintenant le fichier "main.c" et analysez le code. Ce programme contient une erreur triviale qui empêche une utilisation optimale du cache. De quelle erreur s'agit-il ?

Exécution du code :

7.2.4 Exercice 4

Donnée : Corrigez "l'erreur", recompilez et mesurez à nouveau le temps d'exécution (soit avec perf stat, soit avec la commande time). Quelle amélioration constatez-vous ?

Emplacement du code :

Exécution du code :

7.2.5 Exercice 5

Donnée : Relevez les valeurs du compteur L1-dcache-load-misses pour les deux versions de l'application. Quelle facteur constatez-vous entre les deux valeurs ?

Exécution du code :

7.2.6 Exercice 6

Donnée : Lors de la présentation de l'outil perf, on a vu que celui-ci permettait de profiler une application avec très peu d'impacts sur les performances. En utilisant la commande time, mesurez le temps d'exécution de notre application ex1 avec et sans la commande perf stat.

Exécution du code :

7.3 Analyse et optimisation d'un programme

Sur la base du programme situé dans le dossier exercices/02/

7.3.1 Exercice 1

Donnée : Décrire en quelques mots ce que fait ce programme

7.3.2 Exercice 2

Donnée : Compilez le programme à l'aide du Makefile joint. Mesurez le temps d'exécution

Exécution du code :

7.3.3 Exercice 3

Donnée : Avant la fonction int main(), ajoutez la méthode suivante : static int compare (const void* a, const void* b) return (short*)a - (short*)b ; Avant « long long sum = 0 ; », ajoutez le code suivant : qsort(data, SIZE, sizeof(data[0]), compare) ; Compilez et mesurez le temps d'exécution de la version modifiée

Emplacement du code :

Exécution du code :

7.3.4 Exercice 4

Donnée : Vous observez sans doute une nette amélioration sur le temps d'exécution. A l'aide de l'outil perf et de sa sous-commande 'stat', en utilisant différents compteurs déterminez pourquoi le programme modifié s'exécute plus rapidement.

Emplacement du code :

Exécution du code :

7.4 Parsing de logs apache

7.4.1 Exercice 1

Donnée : Compilez l'application en utilisant le Makefile fourni et mesurez sur la machine virtuelle son temps d'exécution avec la commande "time".

Exécution du code :

7.4.2 Exercice 2

Donnée : Nous allons maintenant profiler l'application avec perf record avec l'option -g.

```
1 $ perf record -g ./read-apache-logs access_log_NASA_Jul95_samples
```

L'exécution de cette commande doit produire un fichier de résultat perf, nommé perf.data. Si l'on exécute une nouvelle fois la commande, ce fichier sera copié vers perf.data.old et un nouveau perf.data correspondant à la dernière exécution sera créé. A quoi sert l'option -g ? En quoi son résultat être utile ?

Exécution du code :

7.4.3 Exercice 3

Donnée : Nous pouvons maintenant analyser les données collectées par perf avec la commande perf report. Cette commande lance un outil interactif (interface console de type "ncurses") : il est donc possible d'interagir avec les touches du clavier.Chaque ligne représente une fonction, celles récoltant le plus d'évènements sont montrées en haut. En vous basant sur les informations disponibles dans le wiki de perf (https://perf.wiki.kernel.org/index.php/Tutorial), décrivez ce que représente chaque colonne.

Exécution du code :

7.4.4 Exercice 4

Donnée : Sur la capture ci-dessus, on voit par exemple que la majorité des cycles de l'application sont passés dans la fonction std : :string : :size() qui est contenue dans la librairie standard. Il nous manque cependant une information capitale : quelle fonction de notre application fait appel à cette fonction ? Grâce à l'option -g passée à perf record, nous pouvons afficher le call-graph complet. Il suffit de naviguer dans l'interface et développer la ligne std : :string : :size() avec la touche "Enter". Avec les instructions précédentes, déterminez quelle fonction de notre application fait (indirectement) appel à std : :string : :size().

Exécution du code :

7.4.5 Exercice 5

Donnée : Maintenant que vous savez quelle fonction utilise le plus de ressources CPU, trouvez une optimisation du code permettant de réduire drastiquement le temps d'exécution (vous devriez arriver à quelques dixièmes de secondes pour le fichier sample).

Emplacement du code :

Exécution du code :