Construction de systèmes embarqués sous Linux

Rapport de laboratoire

Master HES-SO

Émilie Gsponer, Grégory Emery

16 octobre 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 Travaux pratiques 1

2.1 Gestion de la mémoire, bibliothèques et fonctions utiles

2.1.1 Exercice 4

Donnée : Créer dynamiquement des éléments dans le noyau. Adapter un module noyau afin que l'on puisse lors de son installation spécifier un nombre d'éléments à créer ainsi qu'un texte initial à stocker dans les éléments précédemment alloués. Chaque élément contiendra également un numéro unique, Les éléments seront créés lors de l'installation du module et 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.

2.1.2 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"

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

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

2.2 Accès aux entrées/sorties

2.2.1 Exercice 6

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

```
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

Voici le code :

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/io.h>
#include <linux/ioport.h>

const unsigned long addr = 0x10000000;
const unsigned long mem_len = 0x100;

static int __init skeleton_init(void)
{
  void* regs;
  int memId;

  pr_info("Linux module skeleton loaded\n");

  // Allocate memory region
  request_mem_region(addr,mem_len,"uP register");

  pr_info("Memory allocated\n");

  // Mapp memory with linux noyau
  regs = ioremap(addr, mem_len);
  if (regs == NULL) {
    pr_info ("Error while trying to map processor chipid register...\n");
    return -EFAULT;
```

```
27    }
28
29    // Read memory id;
30    memId = ioread32(regs+0x00);
31    pr_info("uP register: Bit 31..12 : product id=0x%05x\n",(memId>>12)&0x7ffff);
32    pr_info("uP register: Bit 11..8  : package id=0x%01x\n",(memId>>8)&0xf);
33    pr_info("uP register: Bit 7..4   : major revision=0x%01x\n",(memId>>4)&0xf);
34    pr_info("uP register: Bit 3..0   : minor revision=0x%01x\n",(memId)&0xf);
35
36    iounmap(regs);
37
38    return 0;
39 }
40
41 static void __exit skeleton_exit(void)
42 {
43    pr_info("Linux module skeleton unloaded\n");
44
45    // Free memory region
46    release_mem_region(addr,mem_len);
47
48    pr_info("Memory released\n");
49 }
50
51 module_init(skeleton_init);
52 module_exit(skeleton_exit);
53
54 MODULE_AUTHOR("Emilie Gsponer");
55 MODULE_DESCRIPTION("Module Skeleton");
56 MODULE_LICENSE("GPL");
```

code/skeleton_ser1ex6.c

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

2.3 Gestion des interruptions

2.3.1 Exercice 9

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_resquest(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 :

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

– Informations sur les switches de la carte d'extension
  – sw1 - gpio_nr=2, pin_nr=5, io_nr=29
  – sw2 - gpio_nr=2, pin_nr=6, io_nr=30
  – sw3 - gpio_nr=1, pin_nr=6, io_nr=22
  – sw4 - gpio_nr=1, pin_nr=2, io_nr=18

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

```
/*skeleton.c*/
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/moduleparam.h>
#include <linux/interrupt.h>
#include <linux/io.h>
#include <linux/gpio.h>

#define CHECK_RET(ret) if(ret != 0) return -1;

/*First argument, 50 length's string*/
static char* text="dummy help";
module_param(text, charp, 0);
/*Second argument, int */
static int element_num = 1;
module_param(element_num, int, 0);

static int     Pin_nr       = 4;
static int     SW_pin_nr[4]  = {5, 6, 6, 2};
static int     SW_io_nr[4]   = {29, 30, 22, 18};
static int     SW_ID[4]     = {100, 101, 102, 103};
static char *   SW_name[4]    = {"SW1","SW2", "SW3","SW4"};

irqreturn_t switch_irq_handler(int irq, void *dev_id){
  pr_info("Some switch has been pressed\n");
  return IRQ_HANDLED;
}

static int __init skeleton_init(void){
  int ret;
  int i;
  ret = -1;
  pr_info("Interrupt handler module loaded in kernel");
  pr_info("Configuring pins");
  for(i = 0; i < Pin_nr; i++) {
    if(i < 2)   {
      ret = gpio_request(EXYNOS5_GPX2(SW_pin_nr[i]), SW_name[i]);
      ret = gpio_direction_input(EXYNOS5_GPX2(SW_pin_nr[i]));
    } else  {
      ret = gpio_request(EXYNOS5_GPX1(SW_pin_nr[i]), SW_name[i]);
      ret = gpio_direction_input(EXYNOS5_GPX1(SW_pin_nr[i]));
```

```
43      }
44      CHECK_RET(ret);
45    }
46    pr_info("Configuring switches interrupts");
47    for(i = 0; i < Pin_nr; i++) {
48      ret = request_irq(gpio_to_irq(SW_io_nr[i]),
49        switch_irq_handler,
50        IRQF_SHARED|IRQF_TRIGGER_RISING,
51        SW_name[i],
52        &SW_ID[i]);
53      CHECK_RET(ret);
54    }
55    pr_info("Pins and interrupts have been configured.");
56    return ret;
57  }
58
59  static void __exit skeleton_exit(void){
60    int i;
61    pr_info("Freeing interrupts");
62    for(i = 0; i < Pin_nr; i++)
63      free_irq(gpio_to_irq(SW_io_nr[i]), &SW_ID[i]);
64    pr_info("Interrupts freed");
65  }
66
67  module_init(skeleton_init);
68  module_exit(skeleton_exit);
69
70  MODULE_AUTHOR("Greg");
71  MODULE_DESCRIPTION("MISC");
72  MODULE_LICENSE("GPL");
```

code/skeleton_ser1ex9.c

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

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

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

FIGURE 1 – Affichage du chargement du module, des pressions sur les boutons et de la suppression du module