Systèmes embarqués 3/7

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

25 novembre 2016

Le noyau comme boîte à outils

- Le noyau est le seul à gérer certains périphériques (interruptions, DMA),
- le noyau fournit des méthodes “standardisées” pour accéder à divers périphérqiues (*timer*, `gpio`)
- méthodes de partage des ressources : sémaphore et mutex

Gestion des interruptions et propagation du signal vers les processus de l'espace utilisateur qui se sont enregistrés.

Méthodes fournies par le noyau

Nombreuses fonctionnalités fournies par le noyau pour la gestion des tâches (ordonnanceur), *timer* ou communication avec l'espace utilisateur.

- concepts se rapprochant de la programmation microcontrôleur

```
struct timer_list exp_timer;

init_timer_on_stack (&exp_timer );
exp_timer.expires = jiffies+delay*HZ ;
exp_timer.data = 0;
exp_timer.function = do_something ;
add_timer (&exp_timer);
```

- tâches (*tasklet*) : interruptions logicielles

```
char my_tasklet_data[]="my_tasklet_function was called";
void my_tasklet_function( unsigned long data )
{ printk( "%s\n", (char *)data ); return; }

DECLARE_TASKLET( my_tasklet, my_tasklet_function,
 (unsigned long) &my_tasklet_data );

int init_module( void )
{ tasklet_schedule( &my_tasklet ); return 0; }

void cleanup_module( void )
{ tasklet_kill( &my_tasklet ); return; }
```

Module noyau : accès au matériel

Solution 1 : `ioremap` pour obtenir l'adresse virtuelle correspondant aux registres

Solution 2 : utiliser un pilote existant dans le noyau

```
CONFIG_GPIO_SUNXI:

This option enables support for GPIOs on the Allwinner
SOCs (sun4i/sun5i/sun7i). The GPIOs must be
defined in [gpio_para] section of sysconfig.fex
file (gpio_used/gpio_num/gpio_pin_x variables)


Symbol: GPIO_SUNXI [=y]
Type  : tristate
Prompt: GPIO Support for sunxi platform
  Defined at drivers/gpio/Kconfig:181
  Depends on: GPIOLIB [=y] && (ARCH_SUN4I [=n] || ARCH_SUN5I [=y] || ARCH_S
  Location:
    -> Device Drivers
      -> GPIO Support (GPIOLIB [=y])
```

ATTENTION : utiliser des fonctionnalités du noyau *impose* une license GPL `MODULE_LICENSE("GPL");`

Module noyau : accès au matériel

Solution 1 : `ioremap` pour obtenir l'adresse virtuelle correspondant aux registres

Solution 2 : utiliser un pilote existant dans le noyau

```
CONFIG_GPIO_SUNXI:

This option enables support for GPIOs on the Allwinner
SOCs (sun4i/sun5i/sun7i). The GPIOs must be
defined in [gpio_para] section of sysconfig.fex
file (gpio_used/gpio_num/gpio_pin_x variables)
```

Pour utiliser GPIO : définition des broches dans le *devicetree*[1] ou (port-'A')×32+broche

1. linux-4.4.2/arch/arm/boot/dts/sun5i-a13-olinuxino-micro.dts

Module noyau : accès au matériel

Solution 1 : `ioremap` pour obtenir l'adresse virtuelle correspondant aux registres

Solution 2 : utiliser un pilote existant dans le noyau

Pour utiliser GPIO : définition des broches dans le *devicetree*[2]

```
        leds {
                compatible = "gpio-leds";
                pinctrl-names = "default";
                pinctrl-0 = <&led_pins_olinuxinom>;

                power {
                        label = "a13-olinuxino-micro:green:power";
                        gpios = <&pio 6 9 GPIO_ACTIVE_HIGH>;
                        default-state = "on";
                };
...
&pio {
        led_pins_olinuxinom: led_pins@0 {
                allwinner,pins = "PG9";
                allwinner,function = "gpio_out";
                allwinner,drive = <SUN4I_PINCTRL_20_MA>;
                allwinner,pull = <SUN4I_PINCTRL_NO_PULL>;
        };
```

2. linux-4.4.2/arch/arm/boot/dts/sun5i-a13-olinuxino-micro.dts

Module noyau : utilisation de gpiolib

Test de tous les GPIO (22 = EINVAL, 16 = EBUSY) [3]

```
for (gpio =0;gpio<7*32;gpio++) {
  err=gpio_is_valid(gpio);
  err=gpio_request_one(gpio, GPIOF_IN, "jmf_irq");
  if (err!=-22)
    {irq = gpio_to_irq(gpio);
     gpio_free(gpio);  // libere la GPIO pour la prochaine fois
    }
  }
```

3. /usr/include/asm-generic/errno-base.h

Module noyau : gestion des interruptions

```
static irqreturn_t irq_handler(int irq, void *dev_id)
{...
  return IRQ_HANDLED;
}


gpio =('B'-'A')*32+2;   // PB02
err=gpio_is_valid(gpio);
err=gpio_request_one(gpio, GPIOF_IN, "jmf_irq");
if (err!=-22)
  {irq = gpio_to_irq(gpio);
   irq_set_irq_type(irq, IRQ_TYPE_EDGE_BOTH);
   err = request_irq(irq, irq_handler, IRQF_SHARED, "GPIO jmf", &dummy);
   dummy=0;
  }
```

Les signaux : prévenir d'un évènement

Équivalent logiciel en espace utilisateur : les signaux

```
#include <signal.h>

void my_handler (int sig) {
    printf ("I got SIGINT, number %d\n", sig);}

int main ( void ) {
    signal (SIGINT, my_handler);
    while (1) {}
}
```

qui donne chaque fois qu'on appuie sur CTRL-C (tuer par `kill -9 PID`)

```
jmfriedt@(none):~$ ./sigint
I got SIGINT, number 2
I got SIGINT, number 2
```

Distribution du signal d'interruption

Distribution d'interruption par signaux

1. un unique point de gestion d'une interruption matérielle (module noyau)
2. une multitude de processus utilisateurs réagissent à l'interruption
3. chaque processus s'enregistre auprès du module
4. le module réagit au plus vite à l'interruption ...
5. ... puis, quand il a le temps, prévient tous les processus identifés par leur ID.

Distribution du signal d'interruption

Solution alternative (*thread*)

1. un processus désire être informé de l'interruption au cours de son exécution
2. génère un thread qui bloque en lecture sur un point d'entrée de `/sys/class` ou `/dev`
3. quand l'interruption se déclenche, le module débloque le processus en lecture ... [4] [5]
4. ... et le thread prévient son père de l'évènement.

4. `http://www.makelinux.net/ldd3/chp-6-sect-2`
5. `http://www.makelinux.net/ldd3/chp-5-sect-3` pour les sémaphores en espace noyau

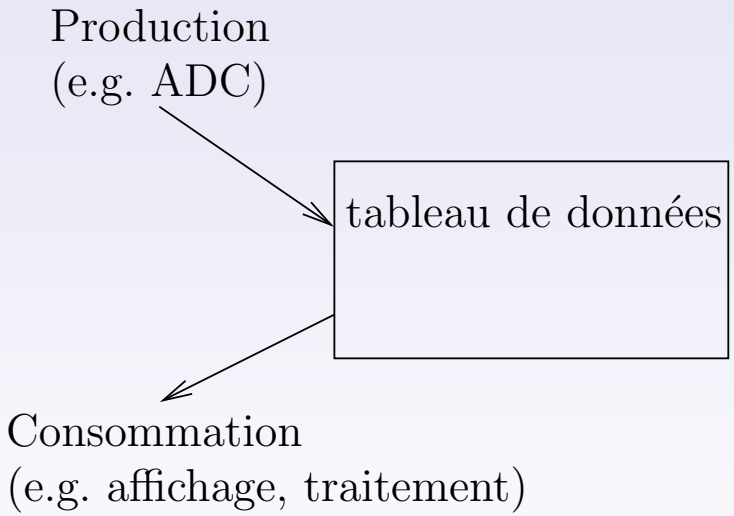
Protection des données

Plusieurs tâches gèrent une même donnée ⇒ risque d'incohérence
Trois méthodes pour protéger une donnée :

- sémaphore (compteur)
- mutex (binaire avec passage en mode veille de la tâche)
- spin-lock (binaire avec test du vérou)

Producteur-consommateur :

1. une tâche produit des données
2. un utilisateur demande ces données
3. la lecture est **bloquée** tant que les données n'ont pas été produites



Protection des données : sémaphore

- Abaisser le sémaphore bloque s'il est nul
- Relever le sémaphore débloque le processus en attent
- Le compteur peut croître pour débloquer plusieurs processus

```
#include <linux/semaphore.h>
struct semaphore mysem;

sema_init(&mysem, 0);       // init the semaphore as empty

down (&mysem); // bloque le consommateur
...
up (&mysem); // producteur debloque
```

Protection des données : mutex

Bloque-débloque une zone de code qui accède à une ressource commune :
une seule instance d'un mutex peut accéder à ce segment de code à un instant donné

```
#include <linux/mutex.h>
struct mutex mymutex;

mutex_init(&mymutex);

mutex_lock(&mymutex);   // bloque
...
mutex_unlock(&mymutex); // debloque
```

Protection des données : spinlock

Le mutex met la tâche en veille ⇒ procédure lourde et lente
Pour des opérations rapides (interruptions), sonder continuellement le vérou : `spinlock`

```
#include <linux/spinlock.h>
static DEFINE_SPINLOCK(myspin);

spin_lock_init(&myspin);

spin_lock(&myspin);
...
spin_unlock(&myspin);

MODULE_LICENSE("GPL");
```

Problème de *deadlock* : deux processus faisant appel à un mécanisme de bloquage sont imbriqués.

Protection des données : spinlock

Le mutex met la tâche en veille ⇒ procédure lourde et lente
Pour des opérations rapides (interruptions), sonder continuellement le vérou : `spinlock`

```
#include <linux/spinlock.h>
static DEFINE_SPINLOCK(myspin);

spin_lock_init(&myspin);

spin_lock(&myspin);
...
spin_unlock(&myspin);

MODULE_LICENSE("GPL");
```

Consommateur	Producteur
Mutex protège	Mutex protège
Sémaphore bloque	...
...	Mutex libère
Mutex libère	Sémaphore débloque

Problème de *deadlock* : deux processus faisant appel à un mécanisme de bloquage sont imbriqués.

Mise en pratique

1. Module noyau sous GNU/Linux
2. Création dynamique du point d'entrée dans `/dev`
3. License GNU pour pouvoir exploiter les fonctionnalités d'autres modules noyau (abstraction du matériel)
4. Protéger les accès à la mémoire lors des échanges de données

Exercice :

1. déclarer un timer qui se réveille toutes les secondes,
2. déclarer un sémaphore qui débloque `read` lorsque le *timer* est atteint
3. déclarer un tableau de données rempli par une fonction déclenchée par le *timer*
4. protéger l'accès à ce tableau par un *mutex*
5. déplacer la production de données dans une *task* au lieu d'effectuer les opérations dans le gestionnaire du *timer*