Électronique numérique

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

17 février 2017

Plan des interventions :

7 cours/TP (séances de 4 h) :

1. Exploitation d'un signal échantillonné en temps discret
   fréquence d'échantillonnage et filtrage
2. Quantification des coefficients, risques de dépassement de capacité.
   Conception et mise en œuvre d'un filtre FIR
3. Environnements exécutifs : FreeRTOS, RTEMS, Nuttx
4. FreeRTOS, RTEMS, Nuttx
   Multitâche et méthodes associées pour garantir l'intégrité des données partagées
5. Système embarqué sous GNU/Linux – système d'exploitation compatible POSIX
   Architecture d'un système d'exploitation, rôle du noyau
   Connectivité internet et configuration réseau
6. Accès aux ressources matérielles depuis l'espace utilisateur (MMU) - - `/dev/mem`
7. Accès aux ressources matérielles depuis un serveur

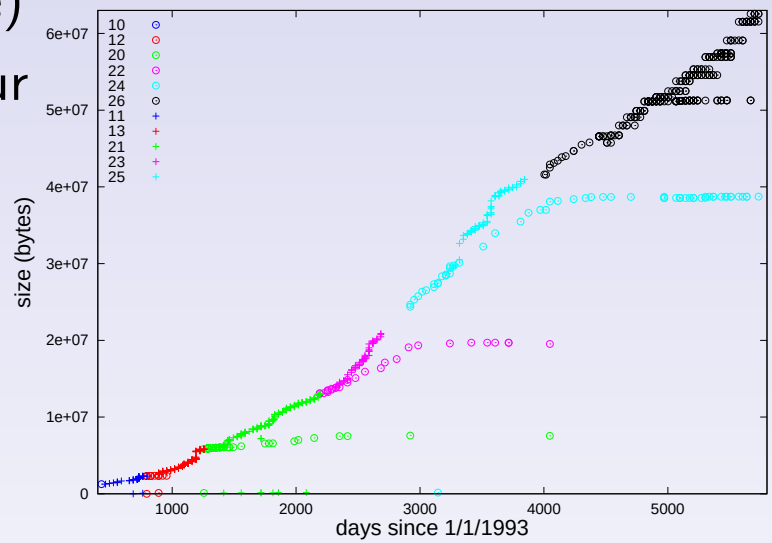
Pilote noyau de type caractère (*char device*) ?

Introduction

- Développeur $\Rightarrow$ s'intéresser à l'interface matériel-utilisateur
- Seul le noyau a accès à toutes les ressources de la machine
- Distribuer les ressources et s'assurer de la cohérence des accès
- Ajout de fonctionnalités au noyau Linux : les modules

Porter GNU/Linux à une nouvelle architecture

- une chaîne de compilation
- tenir compte de l'architecture mémoire de la cible (option -T de `ld`)
- développer un bootloader (initialisation des périphériques et chargement du noyau en mémoire) [1]
- développer les modules noyau pour supporter le matériel spécifique à la nouvelle architecture [2].

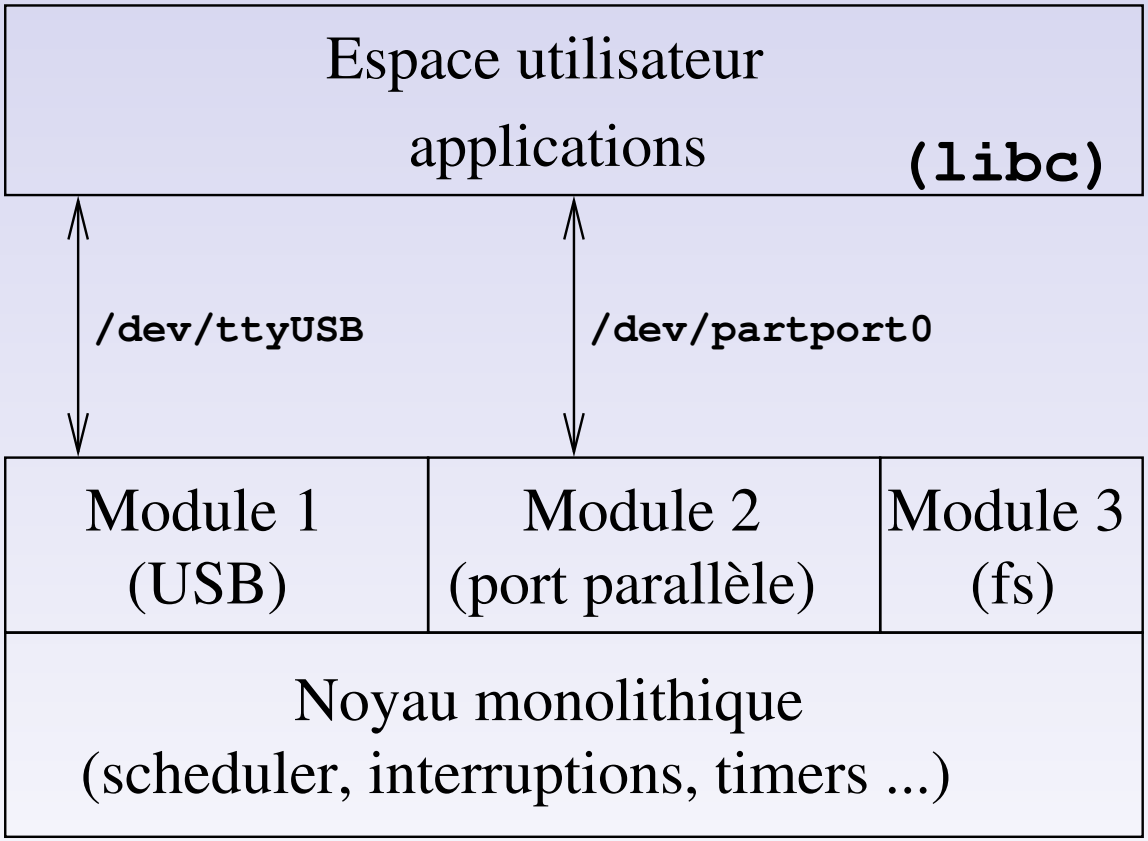


Évolution de la taille de l'archive `.tar.gz` du noyau linux (`www.kernel.org`)

1. S. Guinot, J.-M Friedt, *GNU/Linux sur Playstation Portable*, GNU/Linux Magazine France **114**, Mars 2009, pp.30-40

2. J. Corbet, A. Rubini & G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*, O'Reilly (2005), disponible à `http://lwn.net/Kernel/LDD3/`

Architecture d'un OS



fonctions système, réseau, gestion processus et mémoire, vfs

Le répertoire /dev

- contient des pseudo-fichiers faisant le lien entre un module noyau et l'espace utilisateur
- chaque *device* est défini par un *major number* et, dans sa classe, un *minor number*
- du côté du noyau, un pilote (*driver*) s'est lié au même major number et fournit des méthodes standard
  - `open()`
  - `close()`
  - `write()`
  - `read()`
  - `ioctl()`

```
static struct file_operations qadc_fops =
{
        owner:          THIS_MODULE,
        read:           qadc_read,
        open:           qadc_open,
        ioctl:          qadc_ioctl,
        release:        qadc_release,
};

static int __init qadc_init (void)
{...
 register_chrdev (qadc_major, "ppp", &qadc_fops);
 ...}

module_init (qadc_init);
module_exit (qadc_exit);
```

Structure de base

- deux points d'entrée et de sortie :
  `module_init (fonction_debut);` et
  `module_exit (fonction_fin);`
- initialisation du point de liaison avec l'espace utilisateur
  `register_chrdev (90 , "jmf" ,&fops );`
- des affichages au travers de `/var/log/syslog` par
  `printk (KERN_ALERT "message formaté");` (*sans virgule !*)
- possibilité de création dynamique de l'entrée dev

```
int hello_start()  // init_module(void)
{jmfdev.name = "jmf";  // /dev/jmf: major = classe misc
 jmfdev.minor = MISC_DYNAMIC_MINOR;
 jmfdev.fops = &fops;
 misc_register(&jmfdev);  // creation dynamique /dev/jmf
}
```

Commandes associées à la gestion des modules

Les module (*kernel object*) d'extension `.ko`

- se lient au noyau par `insmod toto.ko`,
- se retirent du noyau par `rmmod toto`.
- La liste des modules attachés s'obtient par `lsmod`
- Les messages affichés par les modules (`printk`) s'obtiennent par `dmesg`
- Les modules "officiels" se trouvent dans `/lib/modules`
- Les modules "officiels" présentant des dépendances se chargent par `modprobe` [3]

3. voir `/lib/modules/version/modules.dep`

Répertoires asociées à la compilation des modules

Les sources du noyau en cours d'exécution sur PC se trouvent dans `/usr/src` :

- Debian/GNU Linux sépare la partie générale de la partie spécifique à une architecture
- `/usr/src/linux-headers-*-common` ne contient *pas* les spécificités du noyau en cours d'exécution
- `/usr/src/linux-headers-*-*arch` est le répertoire auquel nous ferons appel pour trouver la configuration du noyau.

Makefile particulier pour les modules : ajouter son nouveau module à `obj-m` et appeler `Makefile` du noyau

```
obj-m +=0mymod.o 1mymod.o 2mymod.o
all:
	make -C /usr/src/linux-headers-4.8.0-1-amd64 M=$(PWD) modules
```

Dans buildroot : `output/build/linux-*` pour les sources du noyau de l'A13

Adresses réelle-virtuelle

Équivalent de /dev/mem et mmap en espace utilisateur : ici aussi il faut demander à la MMU de nous fournir le bloc qui pointera vers l'adresse physique :

```
request_mem_region (IO_BASE, 36*9, "GPIO test" );// PG9
jmf_gpio = ( u32 )ioremap(IO_BASE, 36*9);      // 36 bytes/port, ports A-I
writel (0x10, jmf_gpio+36*6+0x04 );            // GPIO PG9 en sortie
```

30.2. Port Register List

Module Name	Base Address
PIO	0x01C20800

Register Name	Offset	Description
Pn_CFG0	n*0x24+0x00	Port n Configure Register 0 (n from 0 to 9)
Pn_CFG1	n*0x24+0x04	Port n Configure Register 1 (n from 0 to 9)
Pn_CFG2	n*0x24+0x08	Port n Configure Register 2 (n from 0 to 9)
Pn_CFG3	n*0x24+0x0C	Port n Configure Register 3 (n from 0 to 9)
Pn_DAT	n*0x24+0x10	Port n Data Register (n from 0 to 9)
Pn_DRV0	n*0x24+0x14	Port n Multi-Driving Register 0 (n from 0 to 9)
Pn_DRV1	n*0x24+0x18	Port n Multi-Driving Register 1 (n from 0 to 9)
Pn_PUL0	n*0x24+0x1C	Port n Pull Register 0 (n from 0 to

Base address, espacement des registres ... : A10 User Manual à https://dl.linux-sunxi.org/A10/

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

```
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; }
```

→ ISR est brève et appelle une tasklet

Module noyau : gestion des interruptions

Seul un module noyau peut accéder aux interruptions du processeur (exemple : interruption 7 est associée à la broche 10 du port parallèle).

```
jmfriedt@(none):~$ cat /proc/interrupts | grep parp
  7:          0    IO-APIC-edge  parport0
```

qui se gère par [4]

```
#define PARALLEL_PORT_INTERRUPT 7
#define BASEPORT                0x378
int s=0;
static int int_handler(void) {printk(">>> PARALLEL PORT INT\n");s=1;}
static int __init jmfpport_init_module (void) {
  ret = request_irq(PARALLEL_PORT_INTERRUPT, &int_handler,
                    SA_INTERRUPT, "parallelport", NULL);
  enable_irq(7);
  outb_p(0x10, BASEPORT + 2);
}
static ssize_t skeleton_read (struct file *file, char *buf,
                size_t count, loff_t *ppos) {
 [...]
 do {} while (sortir==0);    // interruptible_sleep_on(&skeleton_wait);
 err = copy_to_user(buf,string,counter);
 ...
}
```

4. www.captain.at/howto-linux-device-driver-template-skeleton.php

Mise en pratique

1. Module noyau sous GNU/Linux
2. Passage mémoire virtuelle-mémoire rélle en espace noyau
3. Création dynamique du point d'entrée dans `/dev`
4. La lecture de la datasheet (adresse matérielle) reste incontournable !
5. License GNU pour pouvoir exploiter les fonctionnalités d'autres modules noyau (abstraction du matériel)