Électronique numérique

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

Machine virtuelle GNU/Linux à
`http://jmfriedt.sequanux.org/stretch.ova`
(VirtualBox 5.0.24)

2 mars 2018

Plan des interventions :

7 cours/TP (séances de 4 h) :

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

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

Introduction

- Pourquoi un système d'exploitation (OS) sur circuit embarqué ?
- Impact (mémoire, CPU)
- Méthode de travail (les programmes développés sur PC tournent sur le système embarqué)
- gcc : environnement unifié de travail

Au dela du C : un système d'exploitation

- ajout d'une couche d'abstraction supplémentaire (assembleur - C - noyau)
- Pourquoi un système d'exploitation : un scheduler, un gestionnaire de mémoire (multitâche), une couche d'abstraction supposée cacher le matériel au programmeur, gestion des **systèmes de fichier** (> rawrite), **communication** (IP, TCP ...), console pour l'utilisateur.
- Par contre une contrainte additionnelle : maîtriser un nouvel ensemble de protocoles et méthodes de programmation ...
- ... afin de gérer le partage de ressources entre plusieurs programmes utilisateurs.

Généralités sur les systèmes d'exploitation ?

- Qu'est-ce que GNU/Linux : système d'exploitation, clone d'unix, compatible posix, **multiplateforme**.
- **Linux** est un noyau autour duquel fonctionnent des outils libres (**GNU**).
- Diverses bibliothèques C disponibles, avec différents impacts mémoire : glibc, uClibc, newlib ... et différentes fonctionnalités.
- Une distribution n'est qu'un emballage pour ces outils.
- uClinux pour les systèmes sans MMU [1], Linux pour les systèmes avec ; OpenWRT pour les routeurs.
- autres OS propriétaires : ~~MS-Windows~~ [2], LynxOS, QNX, vxWorks, iOS
- autres OS libres : *BSD (Free, Net, Open), Plan9, Inferno, Hurd
- Android : surcouche (bibliothèques, couche applicative) de Google à Linux, exécutable Java

1. *Memory Management Unit*, gestionnaire de mémoire chargé des contrôles d'accès et de la conversion de mémoire virtuelle en mémoire physique
2. `http://www.bbc.com/news/technology-41551546`

Méthodes de développement

OS sur plateforme cible :

- Développement : le code est **validé sur PC** puis transféré sur **plateforme embarquée** (OS commun aux deux platformes).
- Respect des appels système POSIX ⇒ code exploitable sur toute plateforme *sous réserve* de séparer accès matériel et *endianness*
- pour deux systèmes avec **gestionnaire de mémoire**, le code PC est directement exploitable
- en l'absence de gestionnaire de mémoire, il manque quelques fonctionnalités qu'il faut éviter (`fork`, lourdeur de `malloc`)
- NFS (*Network File System*) pour rapidement tester les applications

```
mount -o nolock 192.168.1.2:/home /tmp/nfs
```

- cible est rarement architecture x86 ⇒ cross-compiler
- `output/host/usr/bin/` pour la toolchain (host = PC – y faire pointer `$PATH`)

Rappel : bibliothèques

Une implémentation de `libc` fournissant les appels systèmes de Linux.

Sur plateforme cible : impact d'une chaîne de compilation incohérente

```
# ldd gpio_sleep
        libc.so.0 => /lib/libc.so.0 (0xb6f56000)
        ld-uClibc.so.0 => /lib/ld-uClibc.so.0 (0xb6fac000)
```

- Ici, uClibc fournit ces fonctionnalités.
- En cas d'oubli de ces bibliothèques dynamiques, le message d'erreur cryptique

  ```
  # ./gpio_sleep
  -sh: ./gpio_sleep: not found
  ```

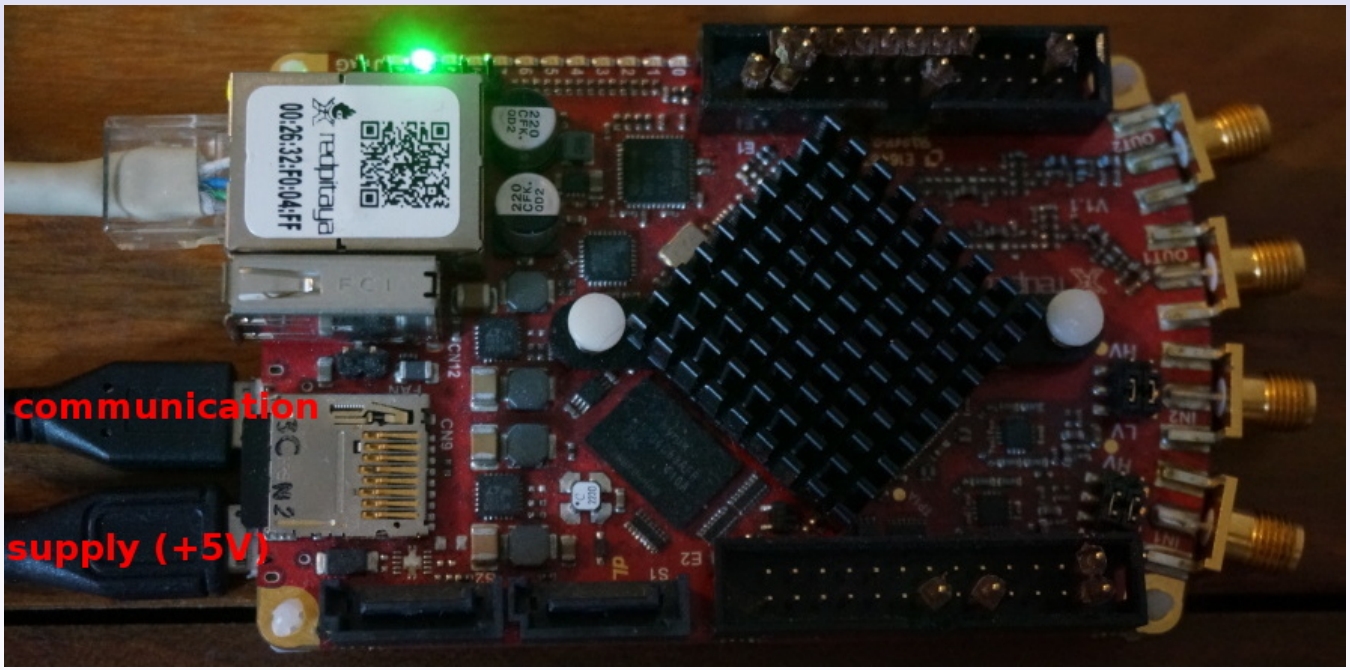
  ici dû à l'utilisation du mauvais compilateur (absence des bibliothèques dynamiques)

  ```
  # ldd gpio_sleep
  checking sub-depends for 'not found'
          libc.so.6 => not found (0x00000000)
          /lib/ld-linux.so.3 => /lib/ld-linux.so.3 (0x00000000)
  # ls -l /lib | grep ld-l
  #
  ```

Environnement de développement

Environnement de développement complexe car doit fournir des outils cohérents pour

- compiler le noyau Linux
- compiler les bibliothèques nécessaires aux outils en espace utilisateur
- compiler les exécutables ("paquets")
- compiler le *bootloader* et fichiers de configuration de la plateforme



Redpitaya : processeur Zynq (ARM Cortex A9), OS et bootloader sur carte SD (+FPGA)

Objectifs

Un environnement cohérent de développement pour générer

1. la toolchain de cross-compilation
2. le bootloader (uboot : initialisation CPU + chargement noyau)
3. le noyau du système d'exploitation (Linux)
4. le rootfs (programmes en espace utilisateur)

qui se trouveront en divers emplacements de la carte SD.

La très grande majorité des dispositifs embaqués sont non-x86 [3] $\Rightarrow$ cross-compilation des programmes de l'hôte vers la cible

3. http://iqjar.com/jar/an-overview-and-comparison-of-todays-single-board-micro-computers/

Environnements de développement

Méthode cohérente de travail pour de nombreuses cibles :

1. Openembedded
2. Yocto
3. **Buildroot**

Pour Olinuxino A13-micro (35 euros [4]) :

- `https://github.com/trabucayre/buildroot`
- 6 GB d'espace disque pour une images de 200+ MB dans `output/images/a13_olinuxino.sdimg`
- Voir dans configs les plateformes supportées : `a13_olinuxino_micro_defconfig` donc [5] : `make a13_olinuxino_micro_defconfig && make`

Pour Redpitaya :

- `https://github.com/trabucayre/redpitaya.git` en complément de la version officielle de buildroot
- `make redpitaya_defconfig && make`

4. `https://www.olimex.com/Products/OLinuXino/A13/A13-OLinuXino-MICRO/open-source-hardware`
5. `http://jmfriedt.free.fr/A13_v2.pdf`

Buildroot

`make menuconfig` pour configurer buildroot (outils espace utilisateur)
`make linux-menuconfig` pour configurer le noyau (support USB p.ex – le noyau se trouve dans `output/build/linux*`)
Organisation de l'arborescence :

1. `configs/*defconfig` : configuration de buildroot
2. `board/a13*/*defconfig` : configuration du noyau Linux, ou `redpitaya/redpitaya/*defconfig`
3. `output` contient tous les résultats de la compilation
4. `output/target/` contient les fichiers pour la cible ARM
5. `output/target/lib` contient les bibliothèques dynamiques du système embarqué
6. `output/build/linux-*` : source du noyau Linux
7. `output/build/linux-*/arch/arm/boot` : noyau Linux compilé
8. `output/images/` : image à flasher sur support non-volatil (`dd`)

Gestion de paquets : ajout dans `packages` [6]

6. `http://www.linuxembedded.fr/2011/03/ajouter-un-package-dans-buildroot-en-5-minutes/`

Accès au port série (matériel)

```
jmfriedt@dhcp-221:~$ ls -l /dev/ttyS*
crw-rw---- 1 root dialout 4, 64 Oct  8 18:43 /dev/ttyS0
crw-rw---- 1 root dialout 4, 65 Oct  8 18:43 /dev/ttyS1
crw-rw---- 1 root dialout 4, 66 Oct  8 18:43 /dev/ttyS2
crw-rw---- 1 root dialout 4, 67 Oct  8 18:43 /dev/ttyS3
```

Exemple de l'accès au port série :

```
int fd;
struct termios oldtio,newtio;
fd=open("/dev/ttyS0", O_RDWR | O_NOCTTY );
tcgetattr(fd,&oldtio); /* save current serial port settings */
newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;  /* _no_ CRTSCTS */
tcsetattr(fd,TCSANOW,&newtio);
```

puis

```
unsigned char cmd;
read(fd,&cmd,1);
printf"(%x) ",(cmd&0xff));fflush(stdout);
```

Outil pour transférer des données sur le port série sous GNU/Linux :

`stty -F /dev/ttyS0 9600 && cat < /dev/ttyS0` ou `minicom`

Accès au matériel depuis l'espace utilisateur

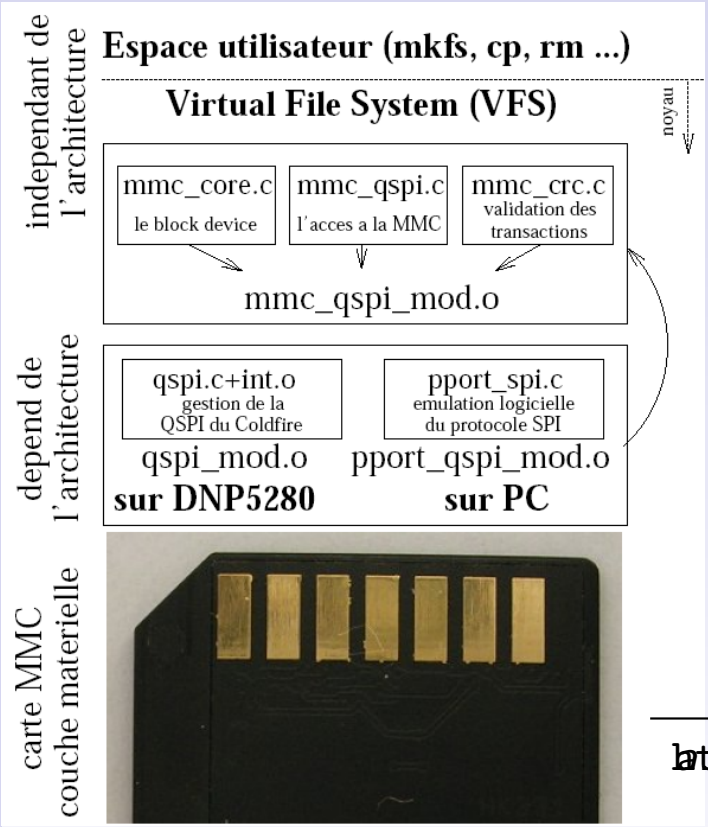
Au travers de `/dev`

- écriture, lecture ou contrôle (`ioctl`)
- passage au travers du module noyau qui implémente les diverses méthodes
- chaque périphérique est identifié par sa classe (*major number*) et son indice (*minor number*)

```
brw-rw----  1 root     disk      8,   0 Feb 28 06:21 sda
brw-rw----  1 root     disk      8,   1 Feb 28 06:21 sda1
brw-rw----  1 root     disk      8,   2 Feb 28 06:21 sda2
brw-rw----  1 root     disk      8,   3 Feb 28 06:21 sda3
[...]
crw-rw----  1 root     dialout   4,  64 Feb 28 07:21 ttyS0
crw-rw----  1 root     dialout   4,  65 Feb 28 07:21 ttyS1
crw-rw----  1 root     dialout   4,  66 Feb 28 07:21 ttyS2
crw-rw----  1 root     dialout   4,  67 Feb 28 07:21 ttyS3
```

En l'absence d'une entrée (par `udev`) dans `/dev` : `mknod`

Accès au matériel depuis l'espace utilisateur [7] [8]



- `block` (b) device ("fichier") se comporte *du point de vue utilisateur* comme `char` (c) device (*pipe*) [a]
- les échanges avec le matériel se font par blocs de données (tampon, cache) au lieu d'octet par octet
- remplacer `file_operation` par `block_operation`
- les transferts sont régis selon une "géométrie" de disque définie dans une structure `gendisk`

a `http://www.makelinux.net/ldd3/chp-16-sect-1`

7. P. Ficheux, *Programmation noyau sous Linux : les pilotes en mode bloc*, GNU/Linux Magazine France, 109, pp.4-10 (Octobre 2008)
8. S. Guinot, J.-M Friedt, *Stockage de masse non-volatile : un block device pour MultiMediaCard*, GNU/Linux Magazine France, Hors Série 25 (Avril 2006)

Accès au matériel depuis l'espace utilisateur

Au travers de `/sys/class`

- échange d'informations en trames ASCII
- configuration au travers du fichier approprié (pas de `ioctl`)
- particulièrement approprié pour une interaction avec l'utilisateur (programmation *shell*)

```
$ cat /sys/class/rtc/rtc0/time
07:37:02
# cat /sys/class/backlight/intel_backlight/max_brightness
4500
# echo "1000" >  /sys/class/backlight/intel_backlight/brightness
```

Attention en C : `fseek` **à l'offset 0 en** `SEEK_SET` **pour plusieurs accès sans refaire** `fopen` **et** `fclose`

LEDs sur A13 (bibliothèque GPIO du noyau) :

```
# echo "1" > /sys/class/gpio/gpio12_pg9/value
```

Mise en pratique

Découverte de la carte Redpitaya [9] (Xilinx Zynq 7010)
Accès au matériel depuis l'espace utilisateur [10]

- configuration réseau TCP/IP (`ifconfig`, `route`)
- `cat < /dev/ttyUSB0`
- depuis le shell (`/sys/class/gpio`) [11] : `Device Drivers` → `GPIO Support`
- ⇒ html/web serveur/CGI (*Common Gateway Interface*)

L'OS n'est pas toujours bien

- un OS doit booter : prend du temps et donc de l'énergie
- un OS nécessite de maîtriser des API
- un OS nécessite de la mémoire et de la puissance de calcul

⇒ bien peser les avantages (bibliothèques, contributions externes, stabilité des outils) et les contres avant de faire un choix.

9. `http://redpitaya.readthedocs.io/`
10. `http://jmfriedt.free.fr/redpitaya.pdf`
11. `https://www.kernel.org/doc/Documentation/gpio/sysfs.txt`