Module Peri 41109

2017fev

Drivers Introduction

PLAN

- Concept de mémoire virtuelle pour les utilisateurs
- Niveau de privilège dans les systèmes d'exploitation
- Appels système : de l'utilisateur au noyau
- Pilote de périphérique dans Linux
- Pilote de GPIO

Mémoire Virtuelle

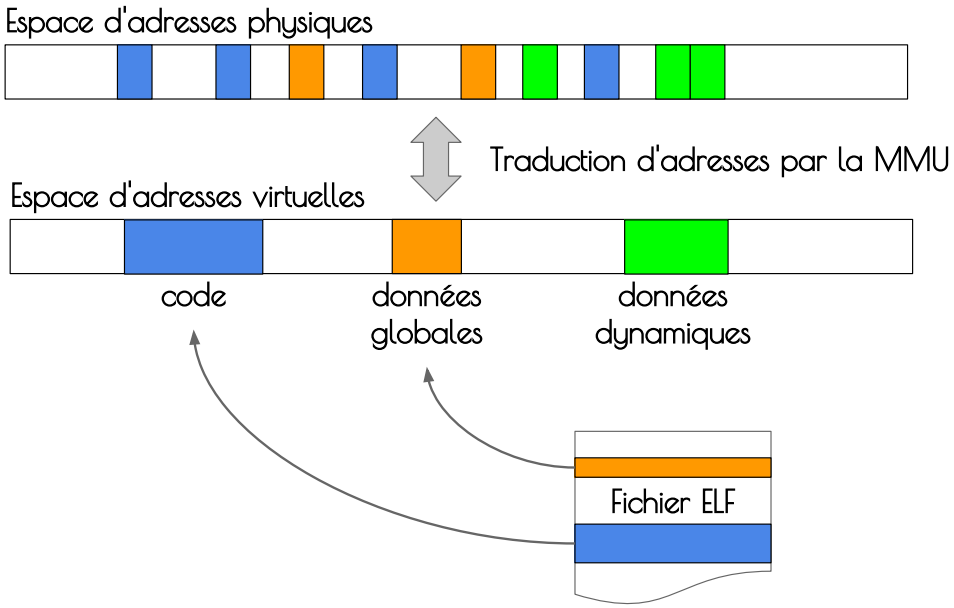
- 2^{32} adresses octets mais pas toutes de la mémoire de données ou d'instructions
  → Certaines adresses permettent d'accéder aux périphériques
- Plusieurs applications simultanées
  → Toutes les applications utilisent les mêmes adresses mais chacune doit avoir son contexte
- Nécessité de protéger les applications entre elles
  → Chaque application a des zones d'adresse protégées
- Chaque application voit un espace d'adresses virtuelles
  → L'espace virtuel est composé de pages de 4ko
  → Une page virtuelle est placée (mapper) dans l'espace physique

Structure d'un programme

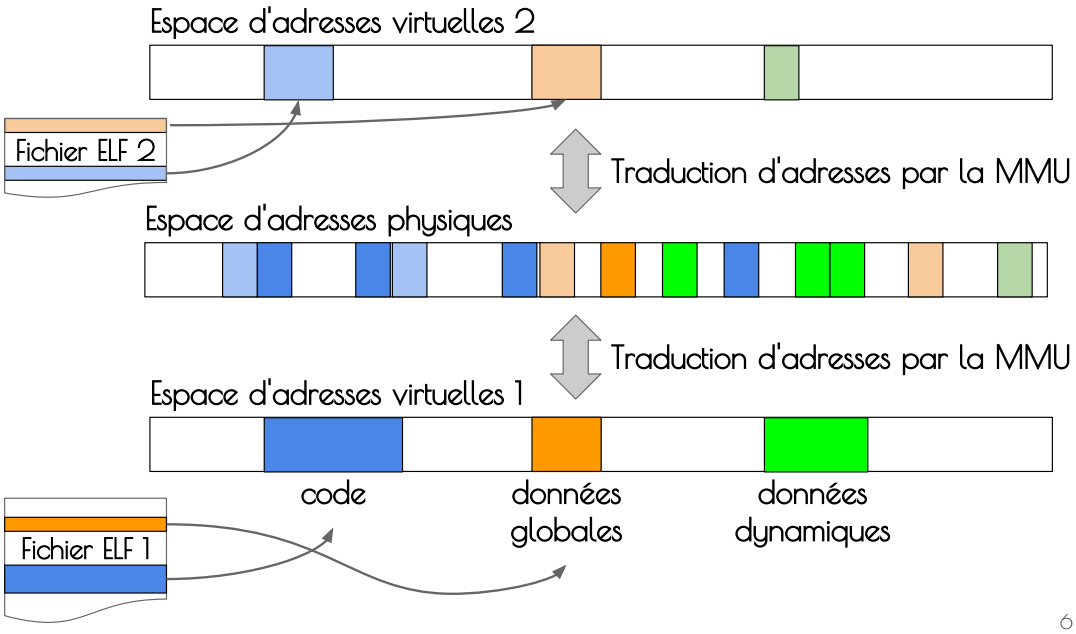
Un programme est contenu dans un fichier binaire ELF

- Segment de code
  - les instructions du programme
  - Toutes les instructions d'un programme ne sont pas dans le programme grâce au concept de bibliothèque dynamique
- Segment de données
  - Données globales (variables globales statiques)
  - Données dynamiques (créées à l'exécution : malloc / mmap)

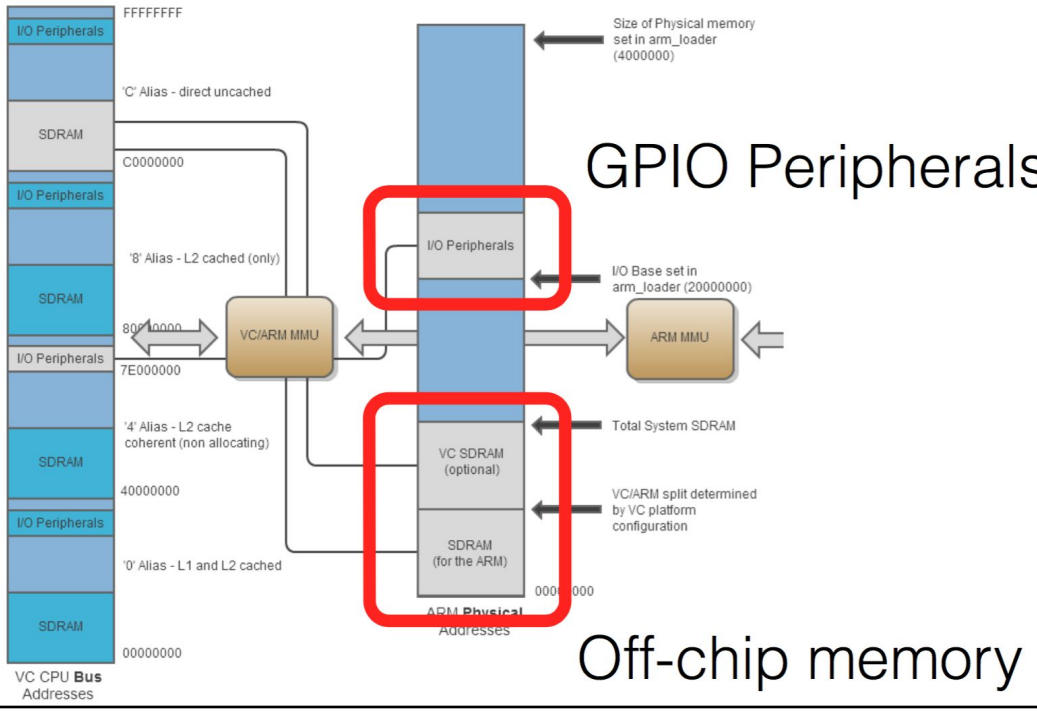
Chargement d'un programme 1/2



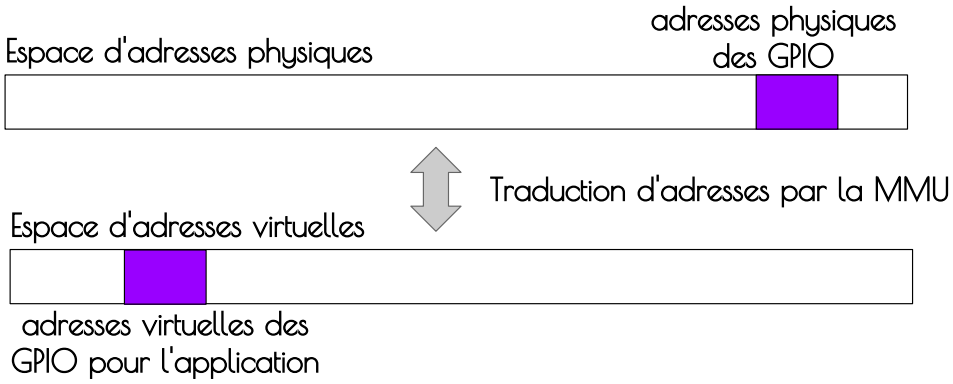
Chargement d'un programme 2/2



Cartographie Mémoire (Mapping) RPi



Mapping des GPIO dans l'espace utilisateur



Le mapping est réalisé par un appel au système

Espace Noyau – Espace Utilisateur

Kernel Land vs User Land

- Deux espaces d'exécution
- Chaque espace a ses droits

Pourquoi cette architecture ?

Pourquoi distinguer les espaces

Chaque espace a un rôle spécifique

- L'espace noyau
  → gère le matériel
  → gère les applications
  multi-applications et multi-utilisateurs
  → gère la virtualisation
    - du processeur (commutation de contexte)
    - de la mémoire (grâce à la MMU)
    - des périphériques (grâce à la MMU et aux fichiers)
- L'espace utilisateur
  → permet l'exécution d'un programme
  avec les ressources (virtuelles) attribuées par le noyau

Niveau de privilèges

Il est nécessaire d'avoir au moins deux niveaux de privilèges

- kernel (supervisor) pour l'espace noyau
- user pour l'espace utilisateur

Sur les processeurs modernes on peut en avoir plus

- Pour virtualiser la machine entière → hypervisor
- Pour limiter les droits de fonctions noyau
  contre les bugs ou les codes malveillants

Niveau de privilèges avec ARM

ARM définit jusqu'à 7 niveaux de privilèges

Mode	Description
User (usr)	Normal ARM execution state
FIQ (fiq)	Designed to support data transfer or channel process
IRQ (irq)	Used for general purpose interrupt handling
Supervisor (svc)	Protected mode for the operating system
Abort mode (abt)	Entered after data or instruction prefetch abort
System (sys)	A privileged user mode for the operating system
Undefined (und)	Entered when an undefined instruction is executed

As copied from the ARM7TDI datasheet (3.6 Operating Modes):

Appel Système (syscall) vu par l'utilisateur

- Un appel système permet à un utilisateur d'exécuter des fonctions avec les privilèges du noyau.
- Fonctionnement :
  - l'utilisateur demande un service avec
    - un numéro prédéfini
    - une liste d'arguments
  - Le noyau vérifie
    - que le service existe
    - que l'utilisateur a le droit de le demander
    - que les arguments sont corrects
  - Cette demande est réalisée en préparant les registres du CPU et en exéctant une instruction spécifique
    - pour ARM scv
    - pour MIPS syscall

open – close syscalls

- Ouverture ou fermeture d'un fichier
  - Dans Linux, les fichiers ne sont pas seulement sur le disque
  - Les périphériques sont aussi des "fichiers"
- Ouvrir un fichier consiste a demander un numéro identifiant le fichier pour l'utilisateur

```
int open (const char * pathname, int flags, mode_t mode);
```

→ retourne un "file descriptor" identifiant le fichier

```
int close (int fd);
```

→ ferme le fichier qui ne sera plus accessible par l'application

read – write Syscalls

- Permet de lire et d'écrire dans un fichier ouvert : fd
- La lecture et l'écriture utilise des tampons : buf
- Les deux fonctions rendent le nombre d'octets lus ou écrits et -1 en cas d'erreur

```
ssize_t read(int fd, void *buf, size_t count);

ssize_t write(int fd, const void *buf, size_t count);
```

nombre signé — rendu par open — buffer d'octets — nombre d'octets

mmap – munmap Syscalls

- Permet de mapper dans une région virtuelle de l'application un fichier ou une zone du fichier
- La mémoire physique est aussi un fichier : `/dev/mem`
  → permet de mapper les registres du GPIO dans l'espace d'une application utilisateur

```
void *mmap(
    void *addr,         // adresse de page dans EV application
    size_t length,      // nombre d'octets voulus
    int prot,           // type d'accès demandé (exec/read/write)
    int flags,          // type de partage
    int fd,             // identifiant du fichier
    off_t offset);      // décalage par rapport au début
int munmap(void *addr, size_t length);
    // adresse allouées et longueur demandée à mmap
```

ioctl Syscall

- Envoie une commande vers le pilote du périphérique
  - lire de donnée depuis le pilote
  - écrire des données dans le pilote
  - modifier le comportement ou configurer le périphérique

- Fonction avec un nombre d'arguments variable
  L'interprétation de la requête et le nombre d'argument dépend du pilote et du périphérique

```
int ioctl(int fd, unsigned long request, ...);
```


Exemple avec les GPIO 1/6

- L'objectif est d'accéder aux GPIO en utilisant le mapping des registres de contrôle dans l'espace virtuel de l'application.

- On doit pouvoir
  - permettre l'accès aux registres par mmap
  - lire et écrire dans les registres

Exemple avec les GPIO 2/6

Address	Field Name	Description	Size	Read/ Write
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x 7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x 7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x 7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W
0x 7E20 0014	GPFSEL5	GPIO Function Select 5	32	R/W
0x 7E20 0018	-	Reserved	-	-
0x 7E20 001C	GPSET0	GPIO Pin Output Set 0	32	W
0x 7E20 0020	GPSET1	GPIO Pin Output Set 1	32	W
0x 7E20 0024	-	Reserved	-	-
0x 7E20 0028	GPCLR0	GPIO Pin Output Clear 0	32	W
0x 7E20 002C	GPCLR1	GPIO Pin Output Clear 1	32	W
0x 7E20 0030	-	Reserved	-	-
0x 7E20 0034	GPLEV0	GPIO Pin Level 0	32	R
0x 7E20 0038	GPLEV1	GPIO Pin Level 1	32	R
0x 7E20 003C	-	Reserved	-	-
0x 7E20 0040	GPEDS0	GPIO Pin Event Detect Status 0	32	R/W
0x 7E20 0044	GPEDS1	GPIO Pin Event Detect Status 1	32	R/W
0x 7E20 0048	-	Reserved	-	-
0x 7E20 004C	GPREN0	GPIO Pin Rising Edge Detect Enable 0	32	R/W
0x 7E20 0050	GPREN1	GPIO Pin Rising Edge Detect Enable 1	32	R/W
0x 7E20 0054	-	Reserved	-	-
0x 7E20 0058	GPFEN0	GPIO Pin Falling Edge Detect Enable 0	32	R/W
0x 7E20 005C	GPFEN1	GPIO Pin Falling Edge Detect Enable 1	32	R/W

Address	Field Name	Description	Size	Read/ Write
0x 7E20 0060	-	Reserved	-	-
0x 7E20 0064	GPHEN0	GPIO Pin High Detect Enable 0	32	R/W
0x 7E20 0068	GPHEN1	GPIO Pin High Detect Enable 1	32	R/W
0x 7E20 006C	-	Reserved	-	-
0x 7E20 0070	GPLEN0	GPIO Pin Low Detect Enable 0	32	R/W
0x 7E20 0074	GPLEN1	GPIO Pin Low Detect Enable 1	32	R/W
0x 7E20 0078	-	Reserved	-	-
0x 7E20 007C	GPAREN0	GPIO Pin Async. Rising Edge Detect 0	32	R/W
0x 7E20 0080	GPAREN1	GPIO Pin Async. Rising Edge Detect 1	32	R/W
0x 7E20 0084	-	Reserved	-	-
0x 7E20 0088	GPAFEN0	GPIO Pin Async. Falling Edge Detect 0	32	R/W
0x 7E20 008C	GPAFEN1	GPIO Pin Async. Falling Edge Detect 1	32	R/W
0x 7E20 0090	-	Reserved	-	-
0x 7E20 0094	GPPUD	GPIO Pin Pull-up/down Enable	32	R/W
0x 7E20 0098	GPPUDCLK0	GPIO Pin Pull-up/down Enable Clock 0	32	R/W
0x 7E20 009C	GPPUDCLK1	GPIO Pin Pull-up/down Enable Clock 1	32	R/W
0x 7E20 00A0	-	Reserved	-	-
0x 7E20 00B0	-	Test	4	R/W

41 registres permettent de définir les fonctions et les comportement des GPIO

Source: BCM2835-ARM-Peripherals.pdf, page 90

Exemple avec les GPIO 3/6

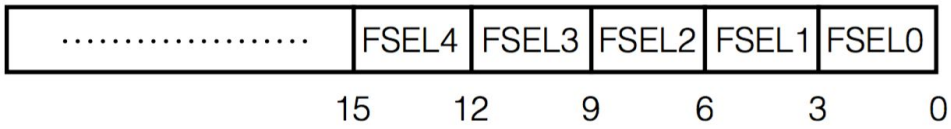
- accès aux registres

```
void * gpio_base;

mmap_fd = open( "/dev/mem",
        O_RDWR | O_SYNC );

if ( mmap_fd < 0 ) {
    return -1;
}

gpio_base = mmap (
    NULL,
    RPI_BLOCK_SIZE,
    PROT_READ | PROT_WRITE,
    MAP_SHARED,
    mmap_fd,
    BCM2835_GPIO_BASE );
```

```
if ( gpio_base == MAP_FAILED ) {
    close ( mmap_fd );
    return -1;
}
```

Exemple avec les GPIO 4/6

- Ecriture dans les registres de configurations des GPIO




```
*( gpio_base + 0x0 ) =
   ( *( gpio_base + 0x0 ) & ~( 0x7 << ( 2 * 3 ) ) )
   | ( 0x1 << ( 2 * 3 ) );
```

→ Ecrit 1 dans FSEL2

Exemple avec les GPIO 5/6

- Pour lire l'état d'une broche

```
uint32_t reg_value =
       ( *( gpio_base + ( 0x34 / 4 ) ) >> 2 ) & 0x1;
```

adresse du registre à partir de la base

Exemple avec les GPIO 6/6

- Pour finir, il faut demappé l'espace virtuel des GPIO

```
munmap ( gpio_base, RPI_BLOCK_SIZE);
```


Pilote de périphérique : utilisateur ou noyau

- Pilote dans l'espace utilisateur
  - profite de la libc, du débugger, de la protection de la MMU
  - mais pas de gestionnaire d'interruptions
  - obligation de passer par mmap
- Pilote dans l'espace noyau
  - exactement le contraire

Pilote de périphérique utilisateur

- Implémenté comme une bibliothèque de fonction
- Possibilité de définir une abstraction du périphérique
- Possibilité de faire des vérifications des arguments
- mais pas de protections problème de sûreté

Ecriture d'un pilote utilisateur

- Il faut spécifier l'API du pilote
  - Que veut-on exposer à l'utilisateur ?
  - Fournir seulement l'accès au périphérique
    mais pas de comportement
- Les comportements sont programmés dans
  - de deamons
  - des processus spécifiques

Pilote de périphérique noyau

- Linux dispose d'un noyau
  → monolithique
  → modulaire
  → orienté objet
- Les pilotes de périphériques sont des extensions du noyau
- Les pilotes contiennent du code spécifique au matériel
- Tous les pilotes implémentent la même API
  → mais chaque pilote propose une implémentation différente

Modèle de pilote Linux

- Un pilote dans le noyau peut être statique
  → inclus dans le noyau Linux à la compilation
- Un pilote peut être dynamique dans un module
  → compilé comme du code relogeable
  → chargé et déchargé à l'exécution
  → plusieurs pilotes peuvent avoir des comportements différents pour la même API
- Les pilotes sont orientés objets dans Linux
- Un sous-système doit
  → permettre la réutilisation du code
  → le code doit être simple, propre, lisible et sans superflu

Sous-systèmes dans Linux

- Linux est composé d'un ensemble de sous-système
  - → file system
  - → network system
  - → memory manager
  - → drivers
  - → etc.
- Il y a plusieurs types de drivers
  - → character ⟵ *c'est ce type qui nous intéresse*
  - → block
  - → network

Pilote de type caractère

- Utilisé pour les périphériques pour lesquels on échange caractère par caractère
- Ils sont accessibles dans le répertoire /dev
  - → chaque pilote est associé à un fichier avec 2 numéros d'identifications : major & minor

Char dev *Major* *minor*

```
crw-rw-rw-   1 root     root      1,   3 Apr 11  2002 null
crw-------   1 root     root     10,   1 Apr 11  2002 psaux
crw-------   1 root     root      4,   1 Oct 28 03:04 tty1
crw-rw-rw-   1 root     tty       4,  64 Apr 11  2002 ttyS0
crw-rw----   1 root     uucp      4,  65 Apr 11  2002 ttyS1
crw--w----   1 vcsa     tty       7,   1 Apr 11  2002 vcs1
crw--w----   1 vcsa     tty       7, 129 Apr 11  2002 vcsa1
crw-rw-rw-   1 root     root      1,   5 Apr 11  2002 zero
```

Concevoir un pilote noyau

- Définir les services
  → uniquement les accès et pas les comportements
  → les comportements seront proposés dans la bibliothèque système (exécuté en mode utilisateur)
- Doit réutiliser le code des autres sous-systèmes
  → USB, interruptions, etc
- Implémenter le comportement du syscall
- Ecrire du code réentrant

TP driver utilisateur dans une bibliothèque

- Définir une API utilisateur permettant d'accéder à deux LEDs et un bouton poussoir (BP)
- Ecrire un programme qui utilise les LEDs et le BP
- Mais aussi
  - utiliser le cross-compilateur
  - écrire un Makefile
  - manipuler le temps
  - etc.