Université du Québec
École de technologie supérieure
Département de génie électrique

ELE784 - Ordinateurs et programmation système

Laboratoire #2

Développement d'un pilote pour une caméra USB sous Linux

Partie 2

Description sommaire :

Dans ce laboratoire, séparé en trois parties, il vous sera demandez de coder un pilote pour une caméra USB répondant au standard UVC. Dans un premier temps, le squelette du module sera mis en place. Par la suite, certaines fonctions types d'un module USB seront ajoutées et finalement le cœur du module sera codé dans la troisième partie. Le résultat final sera un module capable d'envoyer des commandes de base à une caméra et un programme écrit en C utilisé pour communiquer avec ce module. Vous serez donc en mesure d'obtenir des images de la caméra et ces images seront utilisées dans le cadre du laboratoire #3 afin de mettre en évidence l'interaction matériel-logiciel.

Professeur : Bruno De Kelper

Chargé de laboratoire : Louis-Bernard Lagueux

Objectif

Le but ultime de la série de laboratoire de ce cours est de vous faire configurer un système informatique avec un noyau Linux, d'y charger un module (pilote) que vous aurez développé pour contrôler une caméra USB et d'utiliser les images générées par cette dernière afin d'effectuer certains tests sur le processeur. De cette manière, il vous sera possible d'étudier la structure fonctionnelle d'un ordinateur et ses différentes composantes avec un intérêt majeur sur l'interaction matériel-logiciel[1] (ceci est l'un des objectifs principales du cours ELE784).

L'ensemble du laboratoire sera divisé en trois parties:

1. Développement des composantes logicielles de base d'un système informatique. C'est dans cette partie que vous allez configurer le système informatique avec le noyau Linux et avec certains outils couramment utilisés.

2. Développement d'un pilote pour contrôler une caméra USB sous Linux

3. Traitement des données obtenues avec la caméra pour démontrer l'importance de l'interaction matériel-logiciel dans un système informatique.

Les objectifs du laboratoire #2 sont les suivants :

- Se familiariser avec la notion de module et de pilote sous Linux
- Se familiariser avec les différentes commandes utilisées pour travailler avec les modules sous Linux
- Se familiariser avec les différentes sections dans le code d'un module
- Se familiariser avec la notion de synchronisation dans un pilote
- Se familiariser avec le transfert de données entre le « *user space* » et le « *kernel space* »

[1] Adaptation du sommaire du cours que l'ont trouve sur le site du département de génie électrique

Introduction

Dans la seconde partie du deuxième laboratoire vous devrez ajouter certaines fonctions typiques d'un pilote USB nécessaires pour le bon fonctionnement du périphérique. Vous devrez, dans un premier temps, modifier les fonctions d'initialisation et de sortie du module que vous avez créées dans la première partie. Par la suite, Les fonctions *probe* et *disconnect* seront ajoutées. Finalement, nous modifierons la fonction *open* et la fonction *IOCTL* afin de pouvoir faire bouger l'objectif de la caméra.

La fonction d'initialisation

Dans la fonction d'initialisation, vous devrez enregistrer votre pilote USB dans le système avec la fonction `usb_register(…)`. N'oubliez pas de vérifier la valeur de retour de cette fonction et d'afficher un message en fonction du résultat. Pour ce faire, vous aurez, avant toute chose, à créer la structure `usb_driver` (voir diapo #20 du cours #5 et l'exemple dans le livre de référence[2] chapitre #13).

Votre structure `usb_device_id` est la suivante[3] :

```
static struct usb_device_id unnom_id [] = {
        {USB_DEVICE(0x046d, 0x08cc)},
        {},
};
MODULE_DEVICE_TABLE(usb, unnom_id);
```

Les étapes de la fonction d'initialisation seront donc :

1. Afficher le message « `ELE784 -> Init \n\r` »
2. `usb_register(…)`
3. Afficher un message en fonction de la valeur de retour de l'étape #2
4. Quitter la fonction avec la valeur de retour de l'étape #2

La fonction de sortie

Dans cette fonction vous aurez tout simplement à afficher le message suivant :

«`ELE784 -> Cleanup \n\r` »

Par la suite, vous devrez utiliser la fonction `usb_deregister(…)` pour retirer (*dé enregistrer*) votre pilote du système USB. C'est tout !!

La fonction « *probe* »

Avant de pouvoir coder cette fonction, vous devez définir la structure de type `usb_class_driver` ainsi que la structure de type `file_operations` pour votre pilote (voir diapo #29 du cours #5 et l'exemple dans le livre de référence chapitre #13).

[2] Source : http://lwn.net/Kernel/LDD3/
[3] Source : fichier *uvc_driver.c* du module UVC

Ensuite, pour la deuxième partie de ce laboratoire, nous travaillons uniquement avec le *EndPoint* de type contrôle #0. Le standard USB oblige que chaque périphérique incorpore ce *EndPoint*, nous n'aurons donc pas à vérifier s'il est disponible. Nous sauterons donc directement aux étapes nécessaires pour conserver l'information sur le périphérique, pour attacher la structure locale du pilote à l'interface de l'unité-USB et pour inscrire l'unité-USB dans le système (voir diapo #26 du cours #5 et l'exemple dans le livre de référence chapitre #13).

Pour ce faire, vous aurez besoin des fonctions suivantes :

```
interface_to_usbdev(…)
usb_get_dev(…)
usb_set_intfdata(…)
usb_register_dev(…)
```

Jeter un coup d'œil aux diapositives #27 et #28 du cours #5 pour avoir un exemple.

La structure *intf* passée en argument à la fonction *probe* contient plusieurs informations sur le périphérique qui a été inséré dans le système. Vous trouverez notamment la description de tous les *Endpoint* présent sur le périphérique pour chaque configuration. Il pourrait donc être intéressant pour vous d'afficher à l'écran cette information (cette étape est optionnelle. Vous pouvez la faire si vous avez le temps) Pour comprendre cette structure vous pouvez jeter un coup d'œil au chapitre 13 du livre de référence[4] (pages 329 à 332).

La fonction « *disconnect* »

Dans cette fonction vous aurez tout simplement à afficher le message suivant :

```
«ELE784 -> Disconnect \n\r »
```

Par la suite, vous devrez utiliser la fonction `usb_deregister_dev(…)` pour signifier au système USB que votre pilote n'est plus associé au périphérique qui a été retiré.

La fonction « *open* »

Pour cette fonction, vous n'aurez rien à faire, car le code vous est donné dans ce qui suit (petit cadeau gratuit). Prenez quand même le temps de comprendre les étapes qui sont effectuées.

```
static int ele784_open(struct inode *inode, struct file *file)
{
        struct usb_interface *intf;
        int subminor;

        printk(KERN_WARNING "ELE784 -> Open \n\r");

        subminor = iminor(inode);

        intf = usb_find_interface(&mon_driver, subminor);
        if (!intf) {
                printk(KERN_WARNING "ELE784 -> Open: Ne peux ouvrir le peripherique");
                return -ENODEV;
        }

        file->private_data = intf;
        return 0;
```

[4] Source : http://lwn.net/Kernel/LDD3/

```
}
```

La fonction « *IOCTL* »

Dans la deuxième partie du laboratoire, il vous est demandé, pour la fonction IOCTL, d'ajouter le code pour les 4 commandes suivantes :

```
IOCTL_STREAMON            0x30
IOCTL_STREAMOFF           0x40
IOCTL_PANTILT             0x60
IOCTL_PANTILT_RESEST      0x70
```

Dans les sections qui suivent, vous trouverez une description de ce que vous devez faire pour chacune d'entre elles. Par contre, avant toutes choses vous devrez récupérer la référence à la structure *intf*. Pour ce faire, vous aurez à récupérer la « donnée privée » de la structure *file* passée en argument à la fonction *IOCTL*. Cette donnée privée contient une référence à la structure *intf* (grâce à la fonction *open*). Pour ce faire, utiliser les fonctions suivantes :

```
struct usb_interface *intf = file->private_data;
struct usb_device *dev = usb_get_intfdata(intf);
```

Ainsi vous aurez la référence à votre *usb_device* nécessaire pour envoyer une commande au périphérique. Finalement, il est important de savoir que pour certaines commandes, il vous sera nécessaire de passer des données entre votre programme de test et votre pilote. N'oubliez pas d'utiliser les bonnes fonctions pour valider ce que vous transférez (Voir les diapos du cours #4 en plus des chapitres #3 et #6 du livre de référence[5]).

IOCTL_STREAMON

Cette commande est utilisée pour démarrer l'acquisition d'une image par la caméra. Cette commande est envoyée à un *EndPoint* de type contrôle. Voici les informations nécessaires pour acheminer cette commande avec la fonction `usb_control_msg`:

Argument	Information
usb_device	Votre device USB
pipe	Endpoint #0 de type SND
request	0x0B
requestType	USB_DIR_OUT | USB_TYPE_STANDARD | USB_RECIP_INTERFACE
value	**0x0004**
index	0x0001
data	Null
size	0
timeout	0

IOCTL_STREAMOFF

Cette commande est utilisée pour arrêter l'acquisition d'une image par la caméra. Cette commande est envoyée à un *EndPoint* de type contrôle. Voici les informations nécessaires pour acheminer cette commande avec la fonction `usb_control_msg`:

[5] http://lwn.net/Kernel/LDD3/

Argument	Information
usb_device	Votre device USB
pipe	Endpoint #0 de type SND
request	0x0B
requestType	USB_DIR_OUT | USB_TYPE_STANDARD | USB_RECIP_INTERFACE
value	**0x0000**
index	0x0001
data	Null
size	0
timeout	0

IOCTL_PANTILT

Cette commande est utilisée pour modifier la position de l'objectif de la caméra. Les degrés de liberté sont gauche/droite et haut/bas. Cette commande est envoyée à un *EndPoint* de type contrôle. Voici les informations nécessaires pour acheminer cette commande avec la fonction `usb_control_msg`:

Argument	Information
usb_device	Votre device USB
pipe	Endpoint #0 de type SND
request	0x01
requestType	USB_DIR_OUT | USB_TYPE_CLASS | USB_RECIP_INTERFACE
value	0x0100
index	0x0900
data	Voir tableau plus bas
size	4
timeout	0

Direction	Buf[0]	Buf[1]	Buf[2]	Buf[3]
Haut	0x00	0x00	0x80	0xFF
Bas	0x00	0x00	0x80	0x00
Gauche	0x80	0x00	0x00	0x00
Droite	0x80	0xFF	0x00	0x00

IOCTL_PANTILT_RESET

Cette commande est utilisée pour initialiser la position de l'objectif de la caméra (replacer au centre de la caméra). Cette commande est envoyée à un *EndPoint* de type contrôle. Voici les informations nécessaires pour acheminer cette commande avec la fonction `usb_control_msg`:

Argument	Information
usb_device	Votre device USB
pipe	Endpoint #0 de type SND
request	0x01
requestType	USB_DIR_OUT | USB_TYPE_CLASS | USB_RECIP_INTERFACE
value	0x0200
index	0x0900
data	0x03
size	1

timeout	0

IOCTL_GET

Cette commande est utilisée pour récupérer une valeur sur la caméra. Cette commande est envoyée à un *EndPoint* de type contrôle. Voici les informations nécessaires pour acheminer cette commande avec la fonction `usb_control_msg`:

Argument	Information
usb_device	Votre device USB
Pipe	Endpoint #0 de type RCV
Request	GET_CUR (0x81) GET_MIN (0x82) GET_MAX (0x83) GET_RES(0x84)
requestType	USB_DIR_IN | USB_TYPE_CLASS | USB_RECIP_INTERFACE
Value	Processing Unit Control Selectors << 8
Index	0x0200
Data	
Size	2 (rcv)
Timeout	0

IOCTL_SET

Cette commande est utilisée pour affecter une valeur sur la caméra. Cette commande est envoyée à un *EndPoint* de type contrôle. Voici les informations nécessaires pour acheminer cette commande avec la fonction `usb_control_msg`:

Argument	Information
usb_device	Votre device USB
pipe	Endpoint #0 de type SND
request	SET_CUR (0x01)
requestType	USB_DIR_OUT | USB_TYPE_CLASS | USB_RECIP_INTERFACE
value	Processing Unit Control Selectors << 8
index	0x0200
data	2 bytes
size	2
timeout	0

Pour ces deux dernières commandes, les valeurs pour les champs *value*, *index* et *data* devront être passé au pilote par votre programme de test. N'oubliez pas, lorsque vous transférez des données entre votre programme de test et votre pilote (quelque soit la direction) vous devez utiliser des fonctions spéciales.

Un bon test serait de récupérer la valeur minimum, maximum et par défaut de certaines options tel que la luminance, le contraste et le gain. Par la suite, vous pouvez, avec votre programme de test, configurer ces options à votre guise. La section 4.2.2.3 du document `USB_Video_Class_1.1.pdf` peut être utile pour ces deux commandes. De plus la table `A-13` de ce même document vous sera aussi utile.