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 1

**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

# La notion de module sous Linux

Dans les documents de ce laboratoire, nous vous avons souvent parlé de *module* sous Linux sans jamais en avoir donné une description formelle. Je vais donc profiter de ce moment pour clarifier un peu le sujet. Par la suite, une fois les capacités d'un module connues, les commandes de base utilisées sous Linux pour travailler avec les modules vous seront présentées.

## *Les modules*

De façon générale, lorsqu'un système d'exploitation veut effectuer un certain contrôle sur un périphérique quel qu'il soit (USB, carte de type PCI, carte de type ISA, etc…), il a besoin d'un interface entre ses appels systèmes standards et le matériel qu'il désire contrôler. Cet interface est introduit dans le système d'exploitation à l'aide d'un code informatique chargé au démarrage du système informatique. Vous aurez deviné que ce code informatique est en fait un pilote (driver en anglais ou gestionnaire de périphérique). Ces pilotes sont généralement inclus à même le noyau du système informatique et sont activés au démarrage de ce dernier.

Par contre, dans le cas entre autre de Linux, la notion de module a été utilisée. Ces derniers permettent exactement la même chose qu'un pilote en plus d'ajouter la possibilité d'être activés uniquement lorsque l'usager le désire et ce sans même redémarrer le système informatique. De plus les modules peuvent être retirés lorsque l'utilisateur le désire. **Un module est donc un morceau de code permettant d'ajouter des fonctionnalités au noyau de façon dynamiquement sans avoir besoin de recompiler le noyau ou de redémarrer le système**. Il y a, par contre, une légère baisse dans l'efficacité du système. De façon générale, cette baisse n'est pas suffisamment grande pour être remarquée par l'usager.

## *Les commandes pour contrôler les modules*

Pour permettre d'ajouter ou de retirer les modules ou encore pour obtenir de l'information sur les modules, Linux utilise entre autre deux commandes que vous devez bien comprendre afin d'être en mesure d'effectuer votre laboratoire #2. Ces deux commandes (`modprobe` et `lsmod`) ont été inclues par `Busybox` dans votre environnement de travail que vous avez développé dans le cadre du laboratoire #1.

### La commande `modprobe`

La commande `modprobe` charge et décharge un module dans un noyau Linux. Par exemple, si vous voulez ajouter le module `uvcvideo` (inclus dans votre environnement de travail), il suffit de taper la commande suivante :

```
user@ELE784Head:~$ modprobe uvcvideo
```

Il cherchera alors le module désiré dans le répertoire `/lib/modules/2.6.21.6/…` et ses sous répertoires et une fois qu'il l'aura trouvé, il le chargera en mémoire ainsi que toutes ces dépendances (certains modules dépendent sur d'autres modules et `modprobe` se chargera de les installer sans que vous n'ayez quoi que se soit à faire). Par exemple, dans le cas du module `uvcvideo`, ce dernier est dans le répertoire `/lib/modules/2.6.21.6/usb/media` et il ne dépend d'aucun autre module. La commande `modprobe` chargera donc uniquement le module `uvcvideo` en mémoire.

De la même manière, si vous voulez retirer un module du noyau, vous pouvez utiliser la commande `modprobe` avec l'argument `-r` et le nom de module à retirer comme suit :

```
user@ELE784Head:~$ modprobe -r uvcvideo
```

Ceci retirera le module du noyau ainsi que toutes ces dépendances.

## La commande `lsmod`

Si vous voulez connaître la liste des modules qui sont présentement installés dans le système, vous pouvez utiliser la commande `lsmod`. Cette dernière permet en effet d'obtenir la liste des modules présentement installés, leur taille, le compteur d'usage ainsi que toutes les dépendances entre chaque module. Par exemple, si je tape cette commande sur mon ordinateur personnel, j'obtiens ce qui suit :

```
user@host:~$ lsmod
Module                  Size    Used by
snd_hda_intel           251096  1
snd_pcm                 77700   1 snd_hda_intel
snd_page_alloc          10184   2 snd_hda_intel,snd_pcm
asix                    17472   0
usbnet                  18888   1 asix
i2c_i801                8784    0
iwl3945                 169384  0
mii                     5696    2 asix,usbnet
uvcvideo                48452   0
```

Nous pouvons voir que, pour le cas du module `snd_page_alloc`, il fait 10184 bytes et il est utilisé par les modules `snd_ha_intel` et `snd_pcm`.

Finalement, il est important de savoir que la commande `lsmod` ne fait que présenter de façon claire, les informations contenues dans le fichier `/proc/modules`. Vous pouvez aller voir ce fichier en tapant la commande suivante :

```
user@host:~$ cat /proc/modules
```

Dans le cas de mon ordinateur personnel, j'obtiens ce qui suit:

```
user@host:~$ cat /proc/modules
snd_hda_intel 251096 1 - Live 0xf8b81000
snd_pcm 77700 1 snd_hda_intel, Live 0xf8908000
snd_page_alloc 10184 2 snd_hda_intel,snd_pcm, Live 0xf8901000
asix 17472 0 - Live 0xf8962000
usbnet 18888 1 asix, Live 0xf895c000
i2c_i801 8784 0 - Live 0xf8a43000
iwl3945 169384 0 - Live 0xf8931000
mii 5696 2 asix,usbnet, Live 0xf891c000
uvcvideo 48452 0 - Live 0xf8871000
```

# Étapes du laboratoire

La deuxième partie du laboratoire est séparée en trois parties distinctes. Chaque partie devrait vous prendre, en principe, une séance de 4 heures. Il n'y aura pas de livrable à chaque deux semaines, il importe donc à vous de respecter les échéances. Par contre, au bout des 12 heures allouées au laboratoire #2, votre module devra être fonctionnel ainsi que le programme de test nécessaire pour valider son fonctionnement.

## *Étape 1*

Jusqu'à présent, vous n'avez pas encore vue en classe la théorie sur les pilotes USB. Nous devrons donc pour cette semaine, nous concentrer sur le squelette de notre module. Vous aurez donc à créer la fonction d'initialisation (`module_init(…)`), la fonction pour le point de sortie ou de *cleanup* (`module_exit(…)`) , la fonction `open(…)` , la fonction `release(…)`, la fonction `read(…)et` la fonction `ioctl(…)`. Le prototype de ces fonctions est donné dans ce qui suit (n'oubliez pas d'ajouter les attributs pour les fonctions d'initialisation et de point de sortie):

```
static ssize_t ele784_read(struct file * file, char __user * buffer, size_t count, loff_t
*ppos)
static int ele784_open(struct inode *inode, struct file *file)
static int ele784_release(struct inode * inode, struct file * file)
static int ele784_ioctl(struct inode *inode, struct file *file, unsigned int cmd,
unsigned long arg)
static int ele784_init(void);
static void ele784_cleanup(void)
```

Voici ce que vous devez ajouter aux fonctions durant la première séance:

- Dans la fonction `ele784_read(…)`, afficher, à l'aide de la fonction `printk(…)`, le message suivant : « `ELE784 -> read \n\r` »
- Dans la fonction `ele784_open(…)`, afficher, à l'aide de la fonction `printk(…)`, le message suivant : « `ELE784 -> open \n\r` »
- Dans la fonction `ele784_release(…)`, afficher, à l'aide de la fonction `printk(…)`, le message suivant : « `ELE784 -> release \n\r` »
- Dans la fonction `ele784_ioctl(…)`, ajouter la structure switch/case nécessaire pour traiter les commandes suivantes :

```
IOCTL_GET               0x10
IOCTL_SET               0x20
IOCTL_STREAMON          0x30
IOCTL_STREAMOFF         0x40
IOCTL_GRAB              0x50
IOCTL_PANTILT           0x60
IOCTL_PANTILT_RESEST    0x70
```

  Dans chaque « `case` » afficher, à l'aide de la fonction printk(…), le message suivant : « `ELE784 -> ioctl (##) \n\r` ». Remplacer « ## » par le code de la commande. Par exemple, pour la commande `IOCTL_GET`, le message sera : « `ELE784 -> ioctl (0x10) \n\r`»

- Dans la fonction `ele784_init(…)`, vous devez ajouter le code nécessaire pour initialiser un module de type « `char` » avec un « *MAJOR number* » égale à 250 et un

« *MINOR number* » égale à 0. Dans cette fonction, on devrait retrouver au moins les fonctions suivantes :

- `register_chrdev_region(…)`
- `cdev_init(…)`
- `cdev_add(…)`

Je vous réfère aux diapositives du cours #3 et au livre de référence « *Linux Device Drivers, Third Edition*[2] » pour connaître les étapes d'initialisation de périphérique de type « `char` ».

- Dans la fonction `ele784_cleanup(…)`, il faut ajouter le nécessaire pour supprimer l'enregistrement du périphérique de type « *`char`* ». Pour ce faire, il faut utiliser la fonction suivante :
  - `unregister_chrdev_region(…)`

Une fois que vous aurez terminé cette partie, vous aurez entre les mains, les bases d'un module pour un périphérique de type « *`char`* ». Vous pouvez tenter de l'insérer dans votre noyau pour voir le résultat.

Pour pouvoir l'utiliser, vous devez créer votre fichier de périphérique (« *node file* ») avec le « MAJOR number » égal à 250 et le « MINOR number » égal à 0 avec la commande suivante (cette commande doit être exécutée sur votre système informatique et non pas sur le serveur de compilation) :

```
user@192.168.0.2:~$ mknod /dev/ele784 c 250 0
```

Une fois ce fichier créé, vous pouvez, comme premier test, exécuter la commande suivante (cette commande doit être exécutée sur votre système informatique et non pas sur le serveur de compilation):

```
user@192.168.0.2:~$ cat /dev/ele784
```

Si tout est bien initialisé, vous devriez voir les messages que vous avez codés dans certaines de vos fonctions.

Par la suite, en vous inspirant du programme `pingWatchdog.c` situé dans le répertoire `~/ELE784/myWatchdog/`, vous devez créer un programme qui testera chacun des `IOCTL` que vous avez inclus dans votre module.

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