S. Lavirotte
Polytech'Nice – Sophia
SI5 – Master 2 IFI
2018-2019

TD n° 5 et 6
Pilotes et Périphériques Matériels

Le but de ce TD est de développer un module qui prenne en compte un matériel de type USB. Dans un premier temps, nous nous contenterons de détecter la connexion et la déconnexion de ce matériel. Ensuite, nous afficherons les informations relatives à sa description suivant le protocole USB. Enfin, nous complèterons le code développé pour rendre le pilote complètement fonctionnel et accéder aux valeurs d'un capteur connecté à la plateforme.

1 Ecriture d'un squelette de pilote pour dispositif USB

De nombreux périphériques USB sont disponibles sur le marché et il peut être intéressant de voir comment ceux-ci sont gérés dans le noyau Linux. Nous allons donc découvrir les premières étapes de l'écriture d'un pilote pour un périphérique USB.

Nous allons utiliser comme périphérique USB de test une plateforme « Phidget Interface Kit 8/8/8 »[1]. Ce périphérique USB permet d'activer des sorties digitales (Output `0` à `8`) ou de récupérer les informations sur des entrées digitales (Input `0` à `7`) ou issues de capteurs analogiques (que l'on peut brancher sur les connecteurs noirs).

Nous commencerons par mettre en place toute la structure d'un pilote USB qui permet la détection du périphérique. Pour vous aider à débuter l'écriture de ce driver, vous trouverez dans l'archive ci-dessous un `Makefile` adapté ainsi qu'un peu de code spécifique pour décoder les données que fournies la plateforme.

```
http://trolen.polytech.unice.fr/cours/sea/td05-06/td05-06.zip
```

Pour vous faciliter la vie avec la lecture des messages du noyau postés dans /var/log/messages, il est intéressant d'ouvrir un nouveau terminal et d'y lancer la commande :

```
tail -f /var/log/message
```

Ceci aura pour conséquence d'afficher les dernières lignes du fichier en question et de rester en attente des prochaines qui y seront ajoutées.

1.1 Définition d'un pilote USB : détection de l'ajout et du retrait de périphérique

Vous commencerez par créer un fichier `usbdriver1.c` pour y définir les structures nécessaires à la déclaration d'un nouveau pilote USB. Cela permettra au module `usb-core` d'être informé de l'enregistrement d'un nouveau pilote USB pour une classe particulière de dispositif (`vendor` et `product`).

Pour réaliser ceci, vous initialiserez une structure de données de type `usb_driver` et spécifierez la table de type `usb_device_id`. Vous implémenterez ensuite les fonctions `probe` et `disconnect` qui sont appelées par `usb-core` à chaque branchement ou retrait d'un périphérique USB du bus. Vous vous contenterez d'un simple affichage avec un `printk` pour tester la détection de la connexion et de la déconnexion d'un périphérique USB.

1.2 Recherche d'un endpoint du périphérique dans sa description

La phase suivante est de parcourir la description des informations USB du périphérique pour trouver le (ou les) *endpoint* qui permet(tent) d'échanger des données avec le périphérique. Dans notre cas, nous avons un seul *endpoint* de type `IN` qui communique par interruption.

1.3 Enregistrement du dispositif et connexion d'un fichier dans /dev

L'étape suivante est de connecter l'interface que vous récupérez à l'appel de la fonction `probe` avec la structure qui permettra de rendre l'interface accessible côté espace utilisateur (via une entrée dans `/dev`). Vous créerez pour cela une structure `usb_class_driver` en l'initialisant correctement et en particulier en lui fournissant la structure de

[1] https://www.phidgets.com/?tier=3&catid=2&pcid=1&prodid=18

données `file_operations` pour définir les fonctions appelées lors de l'accès au fichier dans `/dev`. Vous définirez le nom dans `usb_classe_driver` à `"skel%d"` ce qui aura pour conséquence de créer une entre `/dev/skel0` lors de l'enregistrement via la fonction `usb_register_dev`. Ainsi, si on a plusieurs plateformes connectées, la suivante prendra le numéro `/dev/skel1` et ainsi de suite.

Vous n'oublierez pas de désenregistrer cette structure à la déconnexion du périphérique du bus USB.

1.4 Fournir les informations de description du *endpoint*

La dernière étape de ce TD est de fournir à la lecture du fichier `/dev/skel0` les informations sur le *endpoint*. Voici les informations que vous devriez obtenir :

```
Endpoint Descriptor:
  bLength:          7
  bDescriptorType:  5
  bEndpointAddress: 0x81
  bmAttributes:     3
  wMaxPacketSize:   64
  bInterval:        8
```

Vous initialiserez pour cela une structure de données pour stocker les informations nécessaires lors de la connexion du dispositif au port USB. Cette structure pourra être du type :

```
struct skel_device {
 struct usb_interface *interface;
 char *buf;
 int used;
};
```

Sur l'accès en lecture du fichier `/dev/skel0`, vous restituerez les informations fournies. Nous sommes là dans la gestion de l'accès à un fichier de `/dev` qui est géré par un `file_operations`. Vous devez donc pouvoir rapidement mettre en place la lecture des données comme nous l'avons fait dans les TD précédents.

2 Pour un pilote complet et fonctionnel … accéder aux données du dispositif

Normalement, le fichier `/dev/skel0` est là pour échanger les données avec la plateforme suivant un standard qui lui est propre (protocole de commande et d'envoie d'information spécifique, défini par le constructeur). Nous allons donc remplacer le comportement précédent pour permettre d'accéder aux données fournies par la plateforme via le fichier `/dev/skel0`. Pour récupérer les données du dispositif en mettant en place les structures de gestion de communication : URB (USB Request Block).

Pour vous faciliter la vie, nous vous fournissons un code dans le fichier `phidget-parse.c` (et l'entête `phidget.h`) permettant d'analyser les données codées dans les 64 octets suivant le protocole utilisé par le matériel. Vous n'aurez donc qu'à utiliser les fonctions `phidget_ifkit_init`, `phidget_ifkit_free` pour initialiser (respectivement libérer) une structure représentant l'interface kit et `phidget_parse_packet` pour décoder le contenu d'un message.

2.1 Création et initialisation d'un URB pour la communication

Vous allez remplacer la fonction `read` que vous aviez associée au fichier `/dev/skel0` pour vous permettre maintenant de lire les données du capteur en provenance du port `0` de la plateforme Phidget InterfaceKit. Nous nous limiterons à la lecture de la donnée d'un capteur pour ne pas trop charger le code et se concentrer sur l'écriture même du pilote et non sur du code fonctionnel.

Il va donc falloir pour cela créer un URB (et stocker les informations dans la structure de donnée `skel_device`) pour communiquer. Pour cela, vous modifierez votre fonction `probe` pour lui ajouter la création d'un URB et son initialisation. Pour mémoire, nous travaillons avec un *endpoint* de type *interrupt*. Vous n'oublierez pas de libérer le URB à la déconnection du périphérique.

2.2 Lecture des données en provenance du périphérique

Vous allez maintenant devoir modifier votre fonction `read` de lecture de votre pilote afin de fournir les données reçues de la plateforme au lieu des informations que vous avez mises en place à la section `1.4`. Pour réaliser cette opération, vous commencerez par compléter votre structure de données `skel_device` en ajoutant au moins les champs suivants :

```
#include <linux/wait.h>
#include <linux/mutex.h>

struct skel_device {
    ...
    // lock until data is read
    wait_queue_head_t wait_q;
    bool reading;
}
```

Dans la fonction `probe`, vous initialiserez cette structure de données grâce à la fonction `init_waitqueue_head`.

Dans la fonction read, vous allez en effet devoir mettre une pause jusqu'à ce que les données soient disponibles. Pour réaliser cela, vous utiliserez la fonction `wait_event_interruptible_timeout`[2] (dont vous analyserez bien sûr la valeur de retour pour savoir si tout s'est bien déroulé). Enfin dans la fonction `skel_completion` que vous ajouterez à votre code pour traiter la mise à disposition des données par le URB, vous ferez un appel à la fonction `wake_up_interruptible`. Dans cette fonction, vous serez aussi amené à mettre en place une barrière mémoire à l'aide de la fonction `smp_wmb`[3] avant d'accéder à la variable `reading` de votre structure `skel_device`.

3 Aide à la réalisation de ce TD

Pour vous aider dans la mise en place de votre premier pilote USB, vous pourrez bien entendu vous aider des notions qui nous ont été expliquées en cours et donc des slides du cours, mais aussi de l'exemple d'un pilote USB donc le code est assez compact :

- https://elixir.bootlin.com/linux/latest/source/drivers/usb/misc/chaoskey.c

Bonne implémentation à vous !

4 Résolution de problèmes

4.1 Pas de dispositifs USB visibles dans le menu « Périphériques/USB »

Ce problème est possible sur une machine hôte Linux ; vous obtenez une liste vide dans le menu Périphériques/USB. Le problème vient d'un problème de droit en lecture pour l'utilisateur qui a lancé VirtualBox. Pour confirmer ce problème, lancer la commande suivante dans un terminal :

```
VBoxManage list usbhost
```

[2] https://static.lwn.net/images/pdf/LDD3/ch07.pdf, page 193 (timeouts)
[3] https://static.lwn.net/images/pdf/LDD3/ch09.pdf, page 215 et suivantes

Si vous obtenez une liste vide, vous êtes victime de ce problème. Pour le corriger, il suffit d'ajouter l'utilisateur que vous êtes aux utilisateurs du groupe `vboxusers`. Voici la procédure à suivre.

- `sudo usermod –a –G vboxusers $(whoami)`
- Se déconnecter de la session
- Se reconnecter sur la session (ce qui a pour effet de prendre en compte la modification de groupe pour votre utilisateur)
- `VBoxManage list usbhost` //doit maintenant vous afficher la liste des dispositifs connecté
- Relancer VirtualBox

4.2 Problème de sélection d'un dispositif dans la liste « Périphériques / USB »

Il est possible que VirtualBox ait un souci avec la capture des dispositifs USB (ce n'est pas normal, mais cela arrive). Cela se produit dans le cas où le pilote USB de VirtualBox s'est mal installé où bien qu'un autre logiciel a installé un pilote de capture des informations USB qui gêne VirtualBox dans son bon fonctionnement.

Si vous rencontré le problème suivant : Pas de case qui se coche et se décoche quand vous sélectionnez un périphérique USB dans la liste des périphériques pour l'attacher à votre machine virtuelle. La solution[4] est la suivante :

1. Editer le registre Windows
   - Lancer l'application d'édition du registre Windows en tapant `regedit` dans la zone de lancement des commandes
   - Naviguer à la clé `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Class{36FC9E60-C465-11CF-8056-444553540000}`
   - Sur la droite si la clé `UpperFilters` existe, la supprimer
2. Installer les pilotes USB de VirtualBox manuellement
   - Aller dans le dossier `C:\Program Files\Oracle\VirtualBox\drivers\USB\filter`
   - Faire un clic-droit sur le fichier `VboxUSBMon.inf` et sélectionner l'option `Installer`
   - Déconnecter tous les dispositifs USB possible et surtout celui que vous souhaitez attacher à votre VM
3. Redémarrer votre machine hôte
4. Lancer VirtualBox et le refermer (cela permet de retirer le dispositif USB de la liste des dispositifs capturés)
5. Connecter votre dispositif USB
6. Relancer VirtualBox et démarrez la machine virtuelle
7. Aller dans le menu Périphérique (ou sur l'icône USB en bas de fenêtre) et sélectionnez le dispositif USB que vous souhaitez attacher à votre VM.

[4] http://superuser.com/questions/1136607/virtualbox-usb-capture-issue-windows-7-host-guest