Aide à la mise en oeuvre d'un périphérique USB sous Linux

Thierry Vaira <tvaira@free.fr>

Table des matières

Aide à la mise en oeuvre d'un périphérique USB sous GNU/Linux **1**
Le noyau Linux 1
Les modules chargeables 2
Les fichiers de périphériques 3
Les périphériques USB 4
Le port série virtuel 4

Site : tvaira.free.fr

Aide à la mise en oeuvre d'un périphérique USB sous GNU/Linux

Le noyau Linux

Le noyau (*kernel*) Linux est un noyau de système d'exploitation de type UNIX. Le noyau Linux est un logiciel libre développé essentiellement en langage C par des milliers de bénévoles et salariés communiquant par Internet.

Le noyau est le coeur du système, c'est lui qui s'occupe de **fournir aux logiciels une interface pour utiliser le matériel**. Le noyau Linux a été créé en 1991 par Linus Torvalds pour les compatibles PC construits sur l'architecture processeur x86 (INTEL PC). Depuis, il a été porté sur nombre d'architectures dont m68k, PowerPC, StrongARM, Alpha, SPARC, MIPS, etc. Il s'utilise dans une très large gamme de matériel, des systèmes embarqués aux superordinateurs, en passant par les ordinateurs personnels.

Ses caractéristiques principales sont d'être multitâche et multi-utilisateur tout en respectant les normes POSIX (UNIX).

Au départ, le noyau a été conçu pour être **monolithique**. Depuis sa version 2.0, le noyau est **modulaire**, c'est-à-dire que certaines fonctionnalités peuvent être ajoutées ou enlevées du noyau à la volée (cad en cours d'utilisation).

Le code source du noyau Linux est disponible sur le site kernel.org, mais les distributions GNU/Linux fournissent également des sources empaquetées sur leurs dépôts.

L'étape la plus importante de la compilation d'un noyau personnalisée est la configuration du noyau. Les options de configuration sont déclarées dans le fichier `.config`, chacun correspond à une fonctionnalité du noyau, qu'on décide d'utiliser ou non. Trois choix sont généralement possibles :

- Y : la fonctionnalité est compilée et implantée dans l'image du noyau
- M : la fonctionnalité est compilée comme module
- N : la fonctionnalité est ignorée

Il existe plusieurs outils (config, menuconfig, gconfig ou xconfig) pour régler la configuration :

```
make config
```

La compilation du noyau et des modules se fait par la commande `make`. Cette opération peut être assez longue. L'installation est automatisée, les commandes `make install` et `make modules_install` permettent respectivement d'installer l'image du noyau et ses modules.

Sur les systèmes Linux, `vmlinux` (ou `vmlinuz` pour sa forme compressé) est le nom du fichier exécutable qui contient le noyau Linux dans l'un des formats de fichiers objet supporté par Linux.

```
$ sudo find / -name vmlinu[xz]* -print
/boot/vmlinuz-3.8.0-44-generic
/boot/vmlinuz-3.8.0-29-generic
/boot/vmlinuz-3.8.0-34-generic
/boot/vmlinuz-3.8.0-33-generic
```

```
$ uname -r
3.8.0-44-generic

$ ls -l /boot/vmlinuz-3.8.0-44-generic
-rw------- 1 root root 5461488 juil. 15  2014 /boot/vmlinuz-3.8.0-44-generic

$ sudo file /boot/vmlinuz-3.8.0-44-generic
/boot/vmlinuz-3.8.0-44-generic: Linux kernel x86 boot executable bzImage, version 3.8.0-44-generic
 (buildd@tipua) #66~precise1-Ubuntu SMP Tue Jul, RO-rootFS, swap_dev 0x5, Normal VGA
```

Les modules chargeables

La modularité du noyau Linux est assuré par des **modules chargeables** ou LKM (*Loadable Kernel Module*).

Un module chargeable du noyau est un **fichier objet** qui contient le code qui permet d'étendre les fonctionnalités du noyau d'un système d'exploitation en cours d'exécution. Ces fichiers portent maintenant l'extension `.ko` (*kernel objet*) :

```
$ sudo find / -name "*.ko" -print
/lib/modules/3.8.0-44-generic/kernel/drivers/usb/class/usblp.ko
/lib/modules/3.8.0-44-generic/kernel/drivers/usb/serial/usbserial.ko
/lib/modules/3.8.0-44-generic/kernel/drivers/usb/storage/usb-storage.ko
...
```

Ils sont généralement utilisés pour ajouter le support d'un nouveau matériel (notion de pilote de périphérique ou *driver*), la prise en charge d'autres systèmes de fichiers (FAT, NTFS, ...), ou pour ajouter des appels système.

Lorsque la fonctionnalité fournie par un LKM est plus nécessaire, il peut être déchargé afin de se libérer de la mémoire et d'autres ressources.

Remarque : La plupart des systèmes d'exploitation actuels supportent les modules chargeables, bien qu'ils puissent utiliser un nom différent pour eux.

Il existe des commandes dédiées à la gestion des modules chargeables :

- `lsmod` : affiche l'état des modules chargés actuellement dans le noyau
- `insmod` : charge un module dans le noyau
- `rmmod` : décharge un module dans le noyau
- `modprobe` : charge et décharge un module dans le noyau
- `modinfo` : affiche des informations sur un module chargeable

La commande `dmesg` est aussi utile pour afficher le tampon des messages du noyau.

```
$ sudo lsmod
Module                  Size  Used by
option                 46396  0
usb_wwan               19467  1 option
usbserial              37161  2 option,usb_wwan
nfsv3                  39946  2
usb_storage            61749  1
usblp                  18314  0
...

$ sudo lsmod | grep usb
usb_wwan               19467  1 option
usbserial              37161  2 option,usb_wwan
usb_storage            61749  0
usblp                  18314  0
usbhid                 47346  0
hid                   105826  2 hid_generic,usbhid

$ sudo rmmod usblp

$ sudo lsmod | grep usb
usb_wwan               19467  1 option
usbserial              37161  2 option,usb_wwan
usb_storage            61749  0
```

 © 2016 tv <tvaira@free.fr>

```
usbhid                 47346  0
hid                   105826  2 hid_generic,usbhid

$ dmesg
...
[178824.184949] usbcore: deregistering interface driver usblp
[178824.185090] usblp0: removed

$ sudo modprobe usblp

$ dmesg | tail -4
...
usblp 1-1.1:1.0: usblp0: USB Bidirectional printer dev 3 if 0 alt 0 proto 2 vid 0x03F0 pid 0x6104
usbcore: registered new interface driver usblp
usblp0: removed
usblp 1-1.1:1.0: usblp0: USB Bidirectional printer dev 3 if 0 alt 0 proto 2 vid 0x03F0 pid 0x6104

$ sudo modinfo usblp
filename:       /lib/modules/3.8.0-44-generic/kernel/drivers/usb/class/usblp.ko
license:        GPL
description:    USB Printer Device Class driver
author:         Michael Gee, Pavel Machek, Vojtech Pavlik, Randy Dunlap, Pete Zaitcev, David Paschal
srcversion:     BD8A318D5C286F1E78768B9
alias:          usb:v04B8p0202d*dc*dsc*dp*ic*isc*ip*in*
alias:          usb:v*p*d*dc*dsc*dp*ic07isc01ip03in*
alias:          usb:v*p*d*dc*dsc*dp*ic07isc01ip02in*
alias:          usb:v*p*d*dc*dsc*dp*ic07isc01ip01in*
alias:          usb:v*p*d*dc07dsc01dp03ic*isc*ip*in*
alias:          usb:v*p*d*dc07dsc01dp02ic*isc*ip*in*
alias:          usb:v*p*d*dc07dsc01dp01ic*isc*ip*in*
depends:
intree:         Y
vermagic:       3.8.0-44-generic SMP mod_unload modversions
parm:           proto_bias:Favourite protocol number (int)
```

Les fichiers de périphériques

Rappel : UNIX/Linux gère l'ensemble de ses ressources sous forme de fichier ce qui permet d'avoir une interface commune (les appels systèmes) pour y accéder (les opérations ouvrir, lire, écrire et fermer).

Les fichiers de périphérique sont des **fichiers spéciaux** situés dans le répertoire `/dev`. Ces fichiers forment un "lien" vers le pilote de périphérique (*driver*) qui a la charge de gérer ce matériel dans le noyau.

Si on liste le contenu de `/dev` :

```
$ sudo ls -l /dev | more
```

On s'aperçoit que certains périphériques sont de type **c** (*character*) dans ce cas ils communiquent octet par octet (par exemple le port série). Alors que d'autres sont de types **b** (*blocks*) ils communiquent par blocs de données (par exemple le disque dur).

Par ailleurs le noyau identifie chaque périphérique au moyen de deux numéros : le **majeur** (ici 8) et le **mineur** (ici de 0, 1, 2, ...)

```
$ ls -l /dev/sd*
brw-rw---- 1 root disk 8,  0 sept. 16 07:35 /dev/sda
brw-rw---- 1 root disk 8,  1 sept. 16 07:35 /dev/sda1
brw-rw---- 1 root disk 8,  2 sept. 16 07:35 /dev/sda2
brw-rw---- 1 root disk 8,  3 sept. 16 07:35 /dev/sda3
brw-rw---- 1 root disk 8,  4 sept. 16 07:35 /dev/sda4
brw-rw---- 1 root disk 8,  5 sept. 16 07:35 /dev/sda5
brw-rw---- 1 root disk 8,  6 sept. 16 07:35 /dev/sda6
brw-rw---- 1 root disk 8, 16 sept. 16 07:35 /dev/sdb
```

Les partitions `sda1` à `sda4` ont le même majeur. Le majeur correspond au premier disque dur SCSI ou SATA. Le noyau identifie ensuite chaque partition grâce au numéro mineur.

 © 2016 tv <tvaira@free.fr>

Les majeurs et les mineurs sont définit dans la documentation accompagnant le noyau Linux dans le fichier `/usr/src/linux/Documentation/devices.txt`.

La commande `mknod` sert à créer de nouveaux périphériques :

```
$ sudo mknod /dev/bidon b 42 0
```

Cf. devices.txt

Les périphériques USB

USB (*Universal Serial Bus*) est une norme relative à un **bus informatique en transmission série** qui sert à connecter des périphériques informatiques à un ordinateur. Le bus USB permet de connecter des périphériques à chaud (quand l'ordinateur est en marche) et en bénéficiant du *Plug and Play* (le système reconnaît automatiquement le périphérique). Il peut alimenter les périphériques peu gourmands en énergie.

Lors de la connexion du périphérique à l'hôte, ce dernier détecte l'ajout du nouvel élément grâce au changement de la tension entre les fils D+ et D-. À ce moment, l'ordinateur envoie un signal d'initialisation au périphérique puis à lui fournir une adresse définitive et à obtenir sa description : c'est la procédure d'énumération.

En effet, après avoir reçu son adresse, le périphérique transmet à l'hôte une liste de caractéristiques qui permettent à ce dernier de l'identifier (type, constructeur, nom, version). L'hôte, disposant de toutes les caractéristiques nécessaires est alors en mesure de charger le pilote approprié.

Les périphériques sont regroupés en types ou classes dans la terminologie USB. Tous les dispositifs d'une classe donnée reconnaissent le même protocole normalisé. Il existe par exemple une classe pour les périphériques de stockage de masse (*mass storage class*), implémentée par la quasi-totalité des clés USB, disques durs externes, appareils photo et par certains baladeurs. La plupart des systèmes d'exploitation possèdent des pilotes génériques, pour chaque type de périphérique. Ces pilotes génériques donnent accès aux fonctions de base, mais des fonctions avancées peuvent manquer.

```
$ lsusb
Bus 001 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 03f0:6104 Hewlett-Packard DeskJet 5650c
Bus 001 Device 004: ID 0461:0010 Primax Electronics, Ltd
Bus 001 Device 005: ID 046d:c05a Logitech, Inc. Optical Mouse M90
Bus 002 Device 003: ID 058f:6366 Alcor Micro Corp. Multi Flash Reader

$ sudo lsusb -v | grep -E '\<(Bus|iProduct|bDeviceClass|bDeviceProtocol)' 2> /dev/null
...

$ dmesg | grep -i usb
...
```

Les périphériques sont généralement identifiés par une paire de nombres hexadécimaux, comme ceci : 03f0 :6104.

- `vendorId` : les 4 premiers chiffres représentent l'ID(entifiant) du vendeur (03f0 = Hewlett-Packard).
- `productId` : les 4 derniers chiffres représentent l'ID du périphérique (6104 = DeskJet 5650c).

Le port série virtuel

Un port série virtuel est une **solution logicielle qui émule un port série standard**.

Cela permet généralement :

- d'augmenter le nombre de ports série (sans installation de matériel supplémentaire mais dans les limites des ressources disponibles)
- de partager les données en provenance d'un périphérique entre plusieurs applications
- de raccorder un périphérique série standard (RS232, ...) sur un port USB avec un adaptateur (manque ou absence de ports série physiques disponibles)

Pour établir une communication effective via un port série physique (RS-232) ou virtuel, il est nécessaire de définir le protocole utilisé : notamment, le **débit de la transmission (en bits/s)**, le codage utilisé, le découpage en trame, etc. La norme RS-232 laisse ces points libres, mais en pratique on utilise souvent des trames d'un caractère ainsi constituées :

 © 2016 tv <tvaira@free.fr>

– 1 bit de départ (START)
– **7 à 8 bit de données**
– **1 bit de parité optionnel (aucune, paire ou impaire)**
– **1 ou plusieurs bits d'arrêt (STOP)**

Retour au sommaire

 © 2016 tv <tvaira@free.fr>