

# Le lecteur vu par Linux

Avant de parler au lecteur, il faut d'abord savoir comment Linux le voit. Et la bonne nouvelle, c'est que Linux, c'est trop bien et nous donne déjà plein d'indices, il suffit de les lire.

## Ce que dit le noyau au branchement

Quand on branche l'Accu-Chek Guide, le noyau l'annonce tout de suite dans ses journaux (`dmesg` ou `journalctl -k`) :

```
usb 3-1: new full-speed USB device number 59 using xhci_hcd
usb 3-1: New USB device found, idVendor=173a, idProduct=21d5, bcdDevice= 1.00
usb 3-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 3-1: Product: ACCU-CHEK Guide
usb 3-1: Manufacturer: Roche
```

Deux numéros sont importants ici :

- **idVendor = 173a** : c'est l'identifiant du fabricant, Roche ;
- **idProduct = 21d5** : c'est le modèle, ici l'Accu-Chek Guide.

Ce couple `173a:21d5`, c'est un peu la plaque d'immatriculation du lecteur. accucheck s'en sert pour savoir à qui il a affaire, et il ne touche à **aucun autre** périphérique USB.

## Regarder sous le capot avec lsusb

Pour avoir plus de détails, la commande `lsusb -v` décrit tout ce que le lecteur déclare. Voici les parties qui nous intéressent :

```
Bus 003 Device 118: ID 173a:21d5 Roche
bNumConfigurations 1
Configuration Descriptor:
bNumInterfaces 1
bConfigurationValue 1
Interface Descriptor:
bInterfaceNumber 0
bAlternateSetting 0
```

```
bNumEndpoints 2
bInterfaceClass 15
iInterface 4 Personal Healthcare Device Class
Endpoint Descriptor:
bEndpointAddress 0x81 EP 1 IN
Transfer Type Bulk
wMaxPacketSize 0x0040 1x 64 bytes
Endpoint Descriptor:
bEndpointAddress 0x01 EP 1 OUT
Transfer Type Bulk
wMaxPacketSize 0x0040 1x 64 bytes
```

Il y a une ligne qui m'a mis sur la piste : **bInterfaceClass 15, "Personal Healthcare Device Class"** (PHDC). Ce n'est pas un protocole maison bricolé par Roche (ce qui nous arrange pas mal), c'est une classe USB standard faite pour les appareils de santé personnels : tensiomètres, balances, glucomètres... On y reviendra au chapitre suivant.

Le reste se lit assez facilement :

- **une seule configuration, une seule interface** : le lecteur ne sait faire qu'une chose ;
- **deux "endpoints"** (des tuyaux) de type **Bulk** : `0x81` pour recevoir (du lecteur vers le PC, "IN") et `0x01` pour envoyer (du PC vers le lecteur, "OUT") ;
- **64 octets** maximum par paquet.

Petite astuce pour s'y retrouver : en USB, le sens est toujours vu depuis le PC. "IN" veut dire "qui rentre dans le PC", donc ce que le lecteur nous envoie. Le bit de poids fort de l'adresse (le `8` de `0x81`) indique justement ce sens.

## Comment accuchekek reconnaît le lecteur

Dans `main.cpp`, la fonction `addDeviceIfAccuChek` passe en revue tous les périphériques USB et applique une série de filtres. Un périphérique n'est retenu que s'il coche toutes les cases :



1.

son couple vendeur:produit est dans la liste des lecteurs connus (sinon on ne l'ouvre même pas) ;

2. il a exactement 1 configuration, 1 interface et 1 "alt setting" ;

- il a exactement 2 endpoints de 64 octets, qui ne sont pas de type "interrupt" ;
- il y en a un dans chaque sens (un IN et un OUT) ;
- on arrive à l'ouvrir et à lire son nom de fabricant et de produit.

La liste des lecteurs connus est intégrée au programme :

| Identifiant USB | Lecteur                             |
|-----------------|-------------------------------------|
| 173a:21d5       | Accu-Chek Guide (modèle 925), testé |
| 173a:21d7       | modèle Roche similaire, non testé   |
| 173a:21d8       | Relion Platinum (modèle 982)        |

On peut en ajouter ou en désactiver avec un petit fichier de configuration passé via `--config`, une ligne par modèle :

```
vendor_0x173a_device_0x21d5 1 # Roche Accu-Chek Guide (reports model 925)
vendor_0x173a_device_0x21d8 0 # désactivé
```

## Prendre la main sur le lecteur

Une fois le lecteur trouvé, `operateDevice` enchaîne les appels libusb classiques :

```
libusb_open(dev, &devHandle); // ouvrir le périphérique
libusb_detach_kernel_driver(devHandle, interface); // décrocher un éventuel pilote du noyau
libusb_set_configuration(devHandle, configValue); // choisir la configuration
libusb_claim_interface(devHandle, interface); // réserver l'interface pour nous
libusb_set_interface_alt_setting(devHandle, interface, altSetting);
```

Ensuite, toute la discussion avec le lecteur passe par trois opérations seulement, regroupées dans une petite classe `LibusbTransport` :

- `controlStatus` : une requête USB standard `GET_STATUS`, le "toc toc" du début ;
- `bulkOut` : envoyer un message au lecteur sur l'endpoint OUT ;
- `bulkIn` : lire un message du lecteur sur l'endpoint IN.

Chaque transfert a un délai maximum de **5 secondes**. Si le lecteur ne répond pas (débranché, écran éteint...), on n'attend pas indéfiniment.

Ce découpage en trois opérations a l'air anodin, mais c'est ce qui m'a permis plus tard de rejouer des échanges enregistrés sans lecteur branché. J'en parle au chapitre 6.

# Le problème du root, et la règle udev



Par défaut, sous Linux, un simple utilisateur n'a pas le droit d'ouvrir un périphérique USB "brut". La version d'origine devait donc tourner avec `sudo`. Ça marche, mais ce n'est pas terrible : un programme qui décode des paquets venant de l'extérieur n'a vraiment aucune raison d'avoir tous les droits sur la machine.

La solution propre, c'est une règle **udev**. udev est le service qui réagit au branchement des périphériques. Voici la règle installée dans `/etc/udev/rules.d/70-accuchek.rules` :

```
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", ATTR{idVendor}=="173a",  
ATTR{idProduct}=="21d5|21d7|21d8", TAG+="uaccess"
```

En français : "quand un périphérique USB Roche de l'un de ces trois modèles arrive, colle-lui l'étiquette **uaccess**". Cette étiquette demande à systemd-logind de donner l'accès au périphérique à l'utilisateur de la **session locale active**, et à lui seul. La personne assise devant le PC peut lire son lecteur, un utilisateur connecté à distance en SSH non.

Le préfixe `70-` du fichier n'est pas là par hasard : la règle doit passer avant `73-seat-late.rules`, celle qui applique réellement les étiquettes `uaccess`. Avec un numéro plus grand, la règle serait lue trop tard et ne servirait à rien.

Après l'installation de la règle, on recharge udev :

```
sudo udevadm control --reload  
sudo udevadm trigger --subsystem-match=usb
```

Puis on débranche et on rebranche le lecteur. À partir de là, accuchek tourne avec les droits normaux de l'utilisateur.

## Savoir pourquoi ça ne marche pas

J'ai aussi fait en sorte qu'accuchek distingue bien les cas "pas de lecteur" et "lecteur trouvé mais accès refusé". Si un lecteur connu est sur le bus mais que `libusb_open` répond `LIBUSB_ERROR_ACCESS`, le programme le retient et le dit clairement :

```
$ accuchek
```

```
accuchek: permission denied on USB meter 173a:21d5 on bus 003 device 118
```

```
$ echo $?
```

```
3
```

Le code de sortie **3** veut dire "la règle udev manque ou n'est pas chargée", alors que le code **2** veut dire "aucun lecteur connu sur le bus". Ça a l'air d'un détail, mais c'est ce qui permet à Glucofi d'afficher le bon message ("rebranchez le lecteur" ou "relancez l'installation") au lieu d'une erreur qu'on ne comprendrait pas forcément.

---

Révision #3

Créé 2026-10-02 20:00:46 UTC par Renard

Mis à jour 2026-10-02 20:51:10 UTC par Renard