LKN Chapitre 8: Personnalisation du noyau Linux

Table of Contents

Source: Linux Kernel in a Nutshell: 8 Customizing a Kernel

L'une des parties les plus difficiles de la construction d'une version du noyau Linux consiste à déterminer exactement les pilotes et les options de configuration nécessaires au bon fonctionnement de le machine. Ce chapitre guidera à travers ce processus de recherche et de sélection des bons pilotes.

Utilisation d'un noyau de distribution

L'une des façons les plus simples de déterminer les modules nécessaires est de commencer par la configuration du noyau fournie avec le package de noyau d'une distribution. Il est également beaucoup plus facile de déterminer quels pilotes sont nécessaires sur un système en cours d'exécution, où les pilotes appropriés sont déjà liés au matériel.

Si aucune distribution Linux n'est déjà installée sur la machine pour laquelle on construit le noyau, utiliser une version LiveCD d'une distribution. Cela permet de démarrer Linux sur la machine et de

déterminer les options de configuration du noyau nécessaires pour que le matériel fonctionne correctement.

Où est la configuration du noyau?

Presque toutes les distributions fournissent les fichiers de configuration du noyau dans le cadre du package du noyau de distribution. Lire la documentation spécifique à la distribution pour savoir comment trouver ces configurations. Il se trouve généralement quelque part sous l'arborescence du répertoire `/usr/src/linux/`.

Si la configuration du noyau est difficile à trouver, regarder dans le noyau lui-même. La plupart des noyaux de distribution sont construits pour inclure la configuration dans le système de fichiers `/proc`. Pour déterminer si cela est vrai pour un noyau en cours d'exécution, entrer:

```
ls /proc/config.gz

/proc/config.gz
```

Si le nom de fichier `/proc/config.gz` est présent, copier ce fichier dans le répertoire source du noyau et décompresser:

```
cp /proc/config.gz ~/linux/
cd ~/linux
gzip -dv config.gz

config.gz: 74,9% - remplacé par config
```

Copier ce fichier de configuration dans le répertoire noyau et renommer en .config. Ensuite, utiliser ce ficier comme base de la configuration du noyau pour construire le noyau.

L'utilisation de ce fichier de configuration devrait toujours générer une image de noyau fonctionnelle pour le machine. L'inconvénient de cette image du noyau est qu'on aurea construit presque tous les modules et pilotes du noyau présents dans l'arborescence des sources du noyau. Cela n'est presque jamais nécessaire pour une seule machine, on peut donc commencer à désactiver différents pilotes et options qui ne sont pas nécessaires. Il est recommandé de désactiver uniquement les options que l'on est sûr de ne pas avoir besoin, car certaines parties du système peuvent dépendre d'options spécifiques à activer.

Trouver quel module est nécessaire

La création d'un fichier de configuration provenant d'une distribution prend beaucoup de temps, en raison de la création de tous les pilotes. il est préférable de créer uniquement les pilotes pour le matériel dont on a besoins, ce qui fera gagner du temps lors de la construction du noyau et permettra de créer tout ou partie des pilotes dans le noyau lui-même, en économisant éventuellement un peu de mémoire et sur certaines architectures. , ce qui permet un système plus rapide. Pour réduire lzq

pilotes, il faut déterminer quels modules sont nécessaires pour piloter le matériel. On va prendre deux exemples de la façon de savoir quel pilote est nécessaire pour contrôler quel morceau de matériel.

Plusieurs emplacements sur le système stockent des informations utiles pour déterminer quels périphériques sont liés à quels pilotes dans un noyau en cours d'exécution. L'emplacement le plus important est un système de fichiers virtuel appelé **sysfs**. **sysfs** est toujours monté dans l'emplacement `/sys` du système de fichiers par les scripts d'initialisation de le distribution Linux. **sysfs** donne un aperçu de la façon dont les différentes parties du noyau sont liées, avec de nombreux liens symboliques différents pointant tout autour du système de fichiers.

Dans tous les exemples suivants, les chemins et les types de matériel **sysfs** réels sont affichés. le machine sera différente, mais les emplacements relatifs des informations seront les mêmes. Les noms de fichiers dans **sysfs** sont différents en fonction des machines.

De plus, la structure interne du système de fichiers **sysfs** change constamment, en raison de la réorganisation des périphériques et de la réflexion des développeurs du noyau sur la meilleure façon d'afficher les structures internes du noyau dans l'espace utilisateur. Pour cette raison, au fil du temps, certains des liens symboliques mentionnés précédemment dans ce chapitre peuvent ne pas être présents. Cependant, les informations sont toujours là, juste déplacées un peu.

Exemple: détermination du pilote réseau

L'un des périphériques les plus courants et les plus importants du système est la carte d'interface réseau. Il est impératif de déterminer quel pilote contrôle ce périphérique et de l'activer dans la configuration du noyau pour que le réseau fonctionne correctement.

Commencer par revenir en arrière à partir du nom de la connexion réseau pour savoir quel périphérique PCI le contrôle. Pour ce faire, regarder les différents noms de réseau:

```
ls /sys/classe/net/

eth0 eth1 eth2 lo
```

Le répertoire **lo** représente le périphérique de bouclage réseau et n'est attaché à aucun périphérique réseau réel. Les répertoires **eth0**, **eth1** et **eth2** sont ceux auxquels il faut faire attention, car ils représentent de véritables périphériques réseau.

Pour regarder plus en détail ces périphériques réseau afin de déterminer ceux qui vous intéressent, utiliser l'utilitaire **ifconfig**:

```


/sbin/ifconfig -a

eth0      Link encap:Ethernet  HWaddr 00:12:3F:65:7D:C2
          inet addr:192.168.0.13  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2720792 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1815488 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
```

```
          RX bytes:3103826486 (2960.0 Mb)  TX bytes:371424066 (354.2 Mb)
          Base address:0xdcc0 Memory:dfee0000-dff00000

eth1      Link encap:UNSPEC  HWaddr 80-65-00-12-7D-
C2-3F-00-00-00-00-00-00-00-00
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

eth2      Link encap:UNSPEC  HWaddr 00-02-3C-04-11-09-D2-
BA-00-00-00-00-00-00-00-00
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:60 errors:0 dropped:0 overruns:0 frame:0
          TX packets:60 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:13409 (13.0 Kb)  TX bytes:13409 (13.0 Kb)
```

Dans cette liste, on peut voir que le périphérique **eth0** est le périphérique réseau qui est actif et fonctionne, comme le montrent les lignes:

```
eth0      Link encap:Ethernet  HWaddr 00:12:3F:65:7D:C2
          inet addr:192.168.0.13  Bcast:192.168.0.255  Mask:255.255.255.0
```

Ceux-ci montrent qu'il s'agit d'un périphérique Ethernet avec une adresse IP (inet) valide qui lui est attribuée.

Maintenant qu'on a déterminé que l'on doit s'assurer que le périphérique **eth0** fonctionnera dans le nouveau noyau, il faut trouver quel pilote le contrôle. Il s'agit simplement de parcourir les différents liens dans le système de fichiers sysfs, ce qui peut être fait dans une commande sur une ligne:

```
basename `readlink /sys/class/net/eth0/device/driver/module`

e1000
```

Cela montre que le module nommé **e1000** contrôle le périphérique réseau **eth0**.

```
La commande "basename ..." compresse les étapes suivantes en une seule ligne
de commande:
```

```
  1. On suit le lien symbolique /sys/class/net/eth0/device dans le
répertoire de l'arborescence /sys/device/ qui contient les informations pour
le périphérique qui contrôle eth0. Noter que le répertoire
/sys/class/net/eth0 peut également être un lien symbolique sur les nouvelles
versions du noyau.

  2. Dans le répertoire qui décrit le périphérique dans sysfs, il existe un
lien symbolique vers le pilote lié à ce périphérique. Ce lien symbolique est
appelé pilote, nous suivons donc ce lien.

  3. Dans le répertoire qui décrit le pilote dans sysfs, il existe un lien
symbolique vers le module dans lequel ce pilote est contenu. Ce lien
symbolique est appelé module. On veut la cible de ce lien symbolique. Pour
obtenir la cible, utiliser la commande readlink, qui produit une sortie
telle que:

    readlink /sys/class/net/eth0/device/driver/module

    ../../../../module/e1000

  4. Comme on ne veut que le nom du module, on supprime le reste du chemin
de la sortie de la commande readlink en récupérant uniquement la partie la
plus à droite. C'est ce que fait la commande **basename**. Exécuté
directement sur un chemin d'accès, il produit:

    basename ../../../../module/e1000

    e1000
```

Maintenant qu'on a le nom du module, il faut trouver l'option de configuration du noyau qui le contrôle. On peut parcourir les différents menus de configuration des périphériques réseau ou rechercher le code source du noyau lui-même pour s'assurer qu'on a la bonne option:

```
cd ~/linux/linux-2.6.17.8
find -type f -name Makefile | xargs grep e1000

./drivers/net/Makefile:obj-$(CONFIG_E1000) += e1000/
./drivers/net/e1000/Makefile:obj-$(CONFIG_E1000) += e1000.o
./drivers/net/e1000/Makefile:e1000-objs := e1000_main.o e1000_hw.o
e1000_ethtool.o e1000_param.o
```

Ne pas oublier de remplacer **e1000** utilisé pour cet exemple par le nom du module que recherché.

La chose importante à rechercher dans la sortie de la commande find précédente est toute ligne contenant le terme **CONFIG_**. C'est l'option de configuration que le noyau doit avoir activée pour construire le module. Au dessus Par exemple, l'option **CONFIG_E1000** est l'option de configuration

que l'on recherche.

On dispose maintenant des informations dont on a besoin pour configurer le noyau.

Exécuter l'outil de menu de configuration :

```
make nuconfig
```

Appuyer ensuite sur la touche / (qui lance une recherche) et saisir l'option de configuration, moins la partie **CONFIG_** de la chaîne.

Le système de configuration du noyau indiquera alors exactement où sélectionner l'option pour activer ce module.

Le premier élément de l'affichage correspond exactement à ce qu'on recherche. Les informations d'emplacement indiquent que pour construire le module e1000 dans le noyau, l'option de configuration suivante doit être activée:

```
Device Drivers
    Network device support
        [*] Network device support
            Ethernet (1000 Mbit)
        [*] Intel(R) PRO/1000 Gigabit Ethernet support
```

Ces étapes fonctionneront pour tout type de périphérique actif dans le noyau.

Exemple: un périphérique USB

Comme autre exemple, examinons un convertisseur USB-série présent dans un exemple de système. Il est actuellement connecté au port `/dev/ttyUSB0`, il faut donc regarder dans la section **tty** de **sysfs**:

```
ls /sys/class/tty/ | grep USB

ttyUSB0
```

On peut suivre les **sysfs** de cet appareil pour trouver le module de contrôle, comme indiqué dans la section précédente:

```
basename `readlink /sys/class/tty/ttyUSB0/device/driver/module`

pl2303
```

Rechercher ensuite l'arborescence des sources du noyau pour trouver l'option de configuration qu'il faudra activer:

```
cd ~/linux/linux-2.6.17.8
find -type f -name Makefile | xargs grep pl2303

./drivers/usb/serial/Makefile:obj-$(CONFIG_USB_SERIAL_PL2303) += pl2303.o
```

Utiliser l'outil de configuration du noyau pour trouver l'option appropriée à activer afin de définir l'option **CONFIG_USB_SERIAL_PL2303**.

Dans notre exemple c'est l'option de pilote `USB Prolific 2303 Single Port Serial Driver` qui est nécessaire pour contrôler correctement cet appareil.

Synthèse de la découverte d'appareils

Recherche manuelle

En résumé, voici les étapes nécessaires pour rechercher le pilote d'un périphérique auquel un pilote fonctionnel est déjà lié:

1. Rechercher le périphérique de classe sysfs approprié auquel le périphérique est lié. Les périphériques réseau sont répertoriés dans `/sys/class/net` et les périphériques tty dans `/sys/class/tty`. D'autres types de périphériques sont répertoriés dans d'autres répertoires dans `/sys/class`, selon le type de périphérique.
2. Suivre l'arborescence **sysfs** pour trouver le nom du module qui contrôle ce périphérique. Il se trouve dans `/sys/class/class_name/device_name/device/driver/module`, et peut être affiché à l'aide des applications **readlink** et **basename**: `basename readlink /sys/class/class_name/device_name /device/driver/module`
3. Rechercher dans les Makefiles du noyau la règle **CONFIG_** qui construit ce nom de module en utilisant **find** et **grep**: `find -type f -name Makefile | xargs grep module_name`
4. Rechercher dans le système de configuration du noyau cette valeur de configuration et accéder à l'emplacement dans le menu qu'il spécifie pour permettre la construction de ce pilote.

Recherche par script

Maintenant qu'on a parcouru toutes les étapes pour fouiller dans **sysfs** et suivre les liens symboliques vers les noms de modules, voici un script très simple qui fera tout ce travail, d'une manière différente:

```
#!/bin/bash
#
# find_all_modules.sh
#
for i in `find /sys/ -name modalias -exec cat {} \;`; do
    /sbin/modprobe --config /dev/null --show-depends $i ;
done | rev | cut -f 1 -d '/' | rev | sort -u
```

Ce script parcoure **sysfs** et trouve tous les fichiers appelés **modalias**. Le fichier **modalias** contient l'alias de module qui indique à la commande **modprobe** quel module doit être chargé pour contrôler ce périphérique. L'alias de module est composé d'une combinaison de fabricant de périphérique, d'ID,

de type de classe et d'autres identificateurs uniques pour ce type spécifique de périphérique. Tous les modules de pilotes du noyau ont une liste interne de périphériques qu'ils prennent en charge qui est générée automatiquement par la liste des périphériques que le pilote indique au noyau qu'il prend en charge. **modprobe** examine cette liste de périphériques par tous les pilotes et essaie de le faire correspondre avec l'alias qu'il possède. S'il trouve une correspondance, il chargera alors le module (cette procédure explique comment fonctionne la fonction de chargement automatique du pilote sous Linux.)

Le script fait arrêter le programme **modprobe** avant de charger réellement le module, et affiche simplement les actions qu'il prendrait. Cela nous donne une liste de tous les modules nécessaires pour contrôler tous les périphériques du système. Un petit nettoyage de la liste, en la triant et en trouvant le champ approprié à afficher donne cette sortie:

```
find_all_modules.sh

8139cp.ko
8139too.ko
ehci-hcd.ko
firmware_class.ko
i2c-i801.ko
ieee80211.ko
ieee80211_crypt.ko
ipw2200.ko
mii.ko
mmc_core.ko
pcmcia_core.ko
rsrc_nonstatic.ko
sdhci.ko
snd-hda-codec.ko
snd-hda-intel.ko
snd-page-alloc.ko
snd-pcm.ko
snd-timer.ko
snd.ko
soundcore.ko
uhci-hcd.ko
usbcore.ko
yenta_socket.ko
```

Il s'agit d'une liste de tous les modules nécessaires pour contrôler le matériel de la machine.

Le script imprimera également probablement certains messages d'erreur qui ressemblent à:

```
FATAL: Module pci:v00008086d00002592sv000010CFsd000012E2bc03sc00i00 not
found.
FATAL: Module serio:ty01pr00id00ex00 not found.
```

Ce qui signifie qu'il n'a pas pu trouver de module capable de contrôler cet appareil. Il ne faut pas s'inuiéter à ce sujet, car certains périphériques n'ont pas de pilotes de noyau qui fonctionneront.

Déterminer le module correct à partir de zéro

Parfois, on a pas la possibilité de faire fonctionner un noyau de distribution sur une machine afin de déterminer quels modules de noyau sont nécessaires pour piloter le matériel. Ou on a ajouté du nouveau matériel à un système, et il faut déterminer quelle option de configuration du noyau doit être activée pour le faire fonctionner correctement. Cette section aidera à déterminer comment trouver cette option de configuration pour que le matériel soit opérationnel.

Le moyen le plus simple de déterminer quel pilote contrôle un nouveau périphérique est de créer tous les différents pilotes de ce type dans l'arborescence des sources du noyau en tant que modules et de laisser le processus de démarrage **udev** faire correspondre le pilote au périphérique. Une fois que cela se produit, on peut utiliser les étapes décrites ci-dessus pour déterminer le bon pilote nécessaire, puis revenir en arrière et activer uniquement ce pilote dans la configuration du noyau.

Mais si on ne veut pas créer tous les pilotes, ou si cela ne fonctionne pas pour une raison quelconque, il faudra un peu plus de travail pour déterminer le pilote approprié qui est nécessaire. Les étapes suivantes sont complexes et nécessitent parfois de creuser dans le code source du noyau. Il ne faut pas peur de cela, cela aidera qu'à mieux comprendre le matériel et la source du noyau.

Les étapes de la mise en correspondance du pilote avec le périphérique varient en fonction du type de périphérique avec lequel on travaille. On abordera les deux formes de périphériques les plus courants dans ce chapitre:

1. les périphériques PCI. Les méthodes décrites ici fonctionneront également avec d'autres types d'appareils.
2. les périphériques USB. Les méthodes décrites ici fonctionneront également avec d'autres types d'appareils.
3. les périphériques du Système de fichiers racine. Il est très important que le noyau puisse trouver tous les systèmes de fichiers du système, le plus important étant le système de fichiers racine.

Périphériques PCI

Les périphériques PCI se distinguent par l'ID du fournisseur et l'ID du périphérique; chaque combinaison d'ID de fournisseur et de périphérique peut nécessiter un pilote unique. C'est la base de la recherche que cette section montre comment faire.

Recherche pas à pas

Pour cet exemple, utilisons une carte réseau PCI qui ne fonctionne pas avec la version du noyau en cours d'exécution. Cet exemple sera différent de pour chaque situation, avec différentes valeurs de périphérique PCI et d'ID de bus, mais les étapes impliquées devraient être pertinentes pour tout type de périphérique PCI pour lequel vous souhaitez trouver un pilote fonctionnel.

Tout d'abord, rechercher le périphérique PCI dans le système qui ne fonctionne pas. Pour obtenir une

liste de tous les périphériques PCI, utiliser le programme **lspci**. Parce qu'on ne veut que des périphériques PCI Ethernet, on limite la recherche des périphériques PCI en recherchant uniquement les chaînes contenant le terme Ethernet (insensible à la casse):

```
/usr/sbin/lspci | grep -i ethernet

06:04.0 Ethernet controller: Realtek Semiconductor Co., Ltd.
RTL-8139/8139C/8139C+ (rev 10)
```

C'est l'appareil que l'on souhaite faire fonctionner. [1)]

Presque toutes les distributions placent le programme lspci dans le répertoire /usr/sbin/, mais certaines le placent dans d'autres emplacements. Pour savoir où il se trouve, entrer `which lspci`.

Si la distribution que l'on utilise la place ailleurs que dans `/usr/sbin/lspci`, il faudra indiquer ce chemin chaque fois qu'on utilisera de lspci

Les premiers bits de la sortie **lspci** indiquent l'ID de bus PCI pour ce périphérique, `06:04.0`. C'est la valeur qu'on utilisera en parcourant **sysfs** afin de trouver plus d'informations sur cet appareil.

Aller dans **sysfs** où tous les différents périphériques PCI sont répertoriés et regarder leurs noms:

```
cd /sys/bus/pci/devices/
ls

0000:00:00.0  0000:00:1d.0  0000:00:1e.0  0000:00:1f.3  0000:06:03.3
0000:00:02.0  0000:00:1d.1  0000:00:1f.0  0000:06:03.0  0000:06:03.4
0000:00:02.1  0000:00:1d.2  0000:00:1f.1  0000:06:03.1  0000:06:04.0
0000:00:1b.0  0000:00:1d.7  0000:00:1f.2  0000:06:03.2  0000:06:05.0
```

Le noyau numérote les périphériques PCI avec un premier `0000:` qui n'apparaît pas dans la sortie du programme **lspci**.[2)] Ajouter donc le premier `0000:` au numéro que l'on a trouvé en utilisant **lspci** et aller dans ce répertoire:

```
cd 0000:06:04.0
```

Dans ce répertoire, on souhaite connaître les valeurs des noms de fichiers du fournisseur et du périphérique:

```
cat vendor

0x10ec
```

```
cat device

0x8139
```

Il s'agit des ID de fournisseur et de périphérique pour ce périphérique PCI. Le noyau utilise ces valeurs pour faire correspondre correctement un pilote à un périphérique. Les pilotes PCI indiquent au noyau les ID de fournisseur et de périphérique qu'ils prendront en charge afin que le noyau sache comment lier le pilote au périphérique approprié. Nous nous y référerons plus tard.

Maintenant qu'on connait le fournisseur et l'ID du produit pour ce périphérique PCI, il faut trouver le pilote de noyau approprié qui annonce qu'il prend en charge ce périphérique. Revenir au répertoire source du noyau:

```
cd ~/linux/linux-2.6.17.8/
```

L'emplacement le plus commun pour les ID PCI dans l'arborescence des sources du noyau est `/linux/pci_ids.h`. Rechercher dans ce fichier le numéro de produit du fournisseur:

```
grep -i 0x10ec include/linux/pci_ids.h

#define PCI_VENDOR_ID_REALTEK           0x10ec
```

La valeur définie ici, **PCI_VENDOR_ID_REALTEK** est ce qui sera probablement utilisé dans tout pilote de noyau qui prétend prendre en charge les périphériques de ce fabricant.

Pour être sûr, regarder également dans ce fichier pour l'ID d'appareil, comme il est décrit ici:

```
grep -i 0x8139 include/linux/pci_ids.h

#define PCI_DEVICE_ID_REALTEK_8139      0x8139
```

Cette définition sera utile plus tard.

Rechercher maintenant les fichiers source du pilote faisant référence à cette définition de fournisseur:

```
grep -Rl PCI_VENDOR_ID_REALTEK *

include/linux/pci_ids.h
drivers/net/r8169.c
drivers/net/8139too.c
drivers/net/8139cp.c
```

Le premier fichier répertorié ici, **pci_ids.h** n'a pas besoin d'être examiné, car c'est là que l'on a trouvé la définition d'origine. Mais les fichiers **r8139.c**, **8139too.c** et **8169cp.c** dans le sous-répertoire `drivers/net/` doivent être examinés de plus près.

Ouvrir l'un de ces fichiers dans un éditeur et recherchez **PCI_VENDOR_ID_REALTEK**. Dans le fichier `drivers/net/r8169.c`, il apparaît dans cette section de code:

```
static struct pci_device_id rtl8169_pci_tbl[] = {
        { PCI_DEVICE(PCI_VENDOR_ID_REALTEK, 0x8169), },
        { PCI_DEVICE(PCI_VENDOR_ID_REALTEK, 0x8129), },
        { PCI_DEVICE(PCI_VENDOR_ID_DLINK,   0x4300), },
        { PCI_DEVICE(0x16ec,                0x0116), },
        { PCI_VENDOR_ID_LINKSYS,            0x1032, PCI_ANY_ID, 0x0024, },
        {0,},
};
```

Tous les pilotes PCI contiennent une liste des différents périphériques qu'ils prennent en charge. Cette liste est contenue dans une structure de valeurs **struct pci_device_id**, tout comme celle-ci. C'est ce que l'on doit examiner afin de déterminer si notre périphérique est pris en charge par ce pilote. La valeur du fournisseur correspond ici, mais la deuxième valeur après le fournisseur est la valeur du périphérique. Notre périphérique a la valeur 0x8139, tandis que ce pilote prend en charge les valeurs de périphérique de 0x8169 et 0x8129 pour les périphériques avec l'ID de fournisseur **PCI_VENDOR_ID_REALTEK**. Ce pilote ne prendra donc pas en charge notre appareil.

En passant au fichier `/drivers/8139too.c` suivant, on trouve la chaîne **PCI_VENDOR_ID_REALTEK** dans le bit de code suivant:

```
if (pdev->vendor == PCI_VENDOR_ID_REALTEK &&
    pdev->device == PCI_DEVICE_ID_REALTEK_8139 && pci_rev >= 0x20) {
    dev_info(&pdev->dev,
           "This (id %04x:%04x rev %02x) is an enhanced 8139C+ chip\n",
           pdev->vendor, pdev->device, pci_rev);
    dev_info(&pdev->dev,
           "Use the \"8139cp\" driver for improved performance and
stability.\n");
}
```

L'utilisation de la valeur **PCI_VENDOR_ID_REALTEK** ici correspond également au code qui vérifie si l'ID de périphérique PCI correspond à la valeur **PCI_VENDOR_ID_REALTEK_8139**. Si c'est le cas, le pilote doit imprimer un message qui dit: “Use the 8139cp driver for improved performance and stability”. Peut-être que nous devrions examiner ce pilote ensuite. Même si nous n'avions pas un indice aussi visible, le pilote **8139too.c** n'a pas la paire d'ID de fournisseur et d'appareil que l'on recherche dans une variable **struct pci_device_id**, ce qui nous donne l'indice qu'il ne prendra pas en charge notre appareil.

Enfin, regarder le fichier `drivers/net/8139cp.c`. Il utilise la définition **PCI_VENDOR_ID_REALTEK** dans le segment de code suivant:

```
static struct pci_device_id cp_pci_tbl[] = {
        { PCI_VENDOR_ID_REALTEK, PCI_DEVICE_ID_REALTEK_8139,
          PCI_ANY_ID, PCI_ANY_ID, 0, 0, },
        { PCI_VENDOR_ID_TTTECH, PCI_DEVICE_ID_TTTECH_MC322,
          PCI_ANY_ID, PCI_ANY_ID, 0, 0, },
        { },
};
```

```
MODULE_DEVICE_TABLE(pci, cp_pci_tbl);
```

Voici une utilisation de nos valeurs d'ID de fournisseur et de périphérique dans une variable **struct pci_device_id**. Ce pilote devrait prendre en charge notre appareil.

Maintenant qu'on a le nom du pilote, on peut revenir en arrière, comme indiqué dans la première section de ce chapitre pour trouver la valeur de configuration de noyau appropriée qui devrait être activée pour construire ce pilote.

Synthèse de la recherche

En résumé, voici les étapes nécessaires pour trouver quel pilote PCI peut contrôler un périphérique PCI spécifique:

1. Rechercher l'ID de bus PCI du périphérique pour lequel on souhaite rechercher le pilote, à l'aide de lspci.
2. Accéder au répertoire `/sys/bus/pci/devices/0000:bus_id`, où **bus_id** est l'ID de bus PCI trouvé à l'étape précédente.
3. Lire les valeurs des fichiers du fournisseur et du périphérique dans le répertoire du périphérique PCI.
4. Revenir à l'arborescence des sources du noyau et rechercher dans `include/linux/pci_ids.h` les ID de fournisseur et de périphérique PCI trouvés à l'étape précédente.
5. Rechercher dans l'arborescence des sources du noyau les références à ces valeurs dans les pilotes. Le fournisseur et l'ID de périphérique doivent tous deux être dans une définition **struct pci_device_id**.
6. Rechercher dans les Makefiles du noyau la règle **CONFIG_** qui construit ce pilote en utilisant **find** et **grep**: `find -type f -name Makefile | xargs grep DRIVER_NAME`
7. Rechercher dans le système de configuration du noyau cette valeur de configuration et accéder à l'emplacement dans le menu qu'il spécifie pour permettre la construction de ce pilote.

Périphériques USB

La recherche du pilote spécifique pour un périphérique USB ressemble beaucoup à la recherche du pilote pour un périphérique PCI comme décrit dans la section précédente, avec seulement des différences mineures dans la recherche des valeurs d'ID de bus.

Recherche pas à pas

Dans cet exemple, on recherche le pilote requis pour un périphérique sans fil USB. Comme avec l'exemple de périphérique PCI, les détails de cet exemple seront différents en fonstion des machines, mais les étapes impliquées doivent être pertinentes pour tout type du périphérique USB pour lequel on souhaite trouver un pilote fonctionnel.

Comme pour le périphérique PCI, l'ID de bus doit être trouvé pour le périphérique USB pour lequel on souhaite trouver le pilote. Pour ce faire, on peut utiliser le programme **lsusb** fourni dans le package **usbutils**.

Le programme **lsusb** affiche tous les périphériques USB connectés au système. Comme on ne sait pas comment s'appelle l'appareil spécifique que l'on recherche, on commence par regarder tous les appareils:

```
/usr/sbin/lsusb

Bus 002 Device 003: ID 045e:0023 Microsoft Corp. Trackball Optical
Bus 002 Device 001: ID 0000:0000
Bus 005 Device 003: ID 0409:0058 NEC Corp. HighSpeed Hub
Bus 005 Device 001: ID 0000:0000
Bus 004 Device 003: ID 157e:300d
Bus 004 Device 002: ID 045e:001c Microsoft Corp.
Bus 004 Device 001: ID 0000:0000
Bus 003 Device 001: ID 0000:0000
Bus 001 Device 001: ID 0000:0000
```

Les périphériques avec un ID de **0000:0000** peuvent être ignorés, car ce sont des contrôleurs hôtes USB qui pilotent le bus lui-même. Les filtrer laisse quatre appareils:

```
/usr/sbin/lsusb | grep -v 0000:0000

Bus 002 Device 003: ID 045e:0023 Microsoft Corp. Trackball Optical
Bus 005 Device 003: ID 0409:0058 NEC Corp. HighSpeed Hub
Bus 004 Device 003: ID 157e:300d
Bus 004 Device 002: ID 045e:001c Microsoft Corp.
```

Les périphériques USB étant faciles à retirer, débrancher le périphérique pour lequel on souhaite rechercher le pilote et réexécuter **lsusb**:

```
/usr/sbin/lsusb | grep -v 0000:0000

Bus 002 Device 003: ID 045e:0023 Microsoft Corp. Trackball Optical
Bus 005 Device 003: ID 0409:0058 NEC Corp. HighSpeed Hub
Bus 004 Device 002: ID 045e:001c Microsoft Corp.
```

Le troisième appareil est maintenant manquant, ce qui signifie que l'appareil est représenté comme suit:

```
Bus 004 Device 003: ID 157e:300d
```

est le périphérique pour lequel on souhaite rechercher le pilote.

Si on replace le périphérique et regarde à nouveau la sortie de **lsusb**, le numéro de périphérique aura changé:

```
/usr/sbin/lsusb | grep 157e
```

```
Bus 004 Device 004: ID 157e:300d
```

En effet, les numéros de périphérique USB ne sont pas uniques, mais changent chaque fois qu'ils sont branchés. Ce qui est stable, c'est l'ID du fournisseur et du produit, indiqué ici par lsusb sous la forme de deux valeurs à quatre chiffres avec un **:** entre eux. Pour cet appareil, l'ID fournisseur est **157e** et l'ID produit **300d**. Noter ces valeurs car on les utilisera dans les prochaines étapes.

Comme pour le périphérique PCI, on recherche dans le code source du noyau le fournisseur USB et les ID de produit afin de trouver le pilote approprié pour contrôler ce périphérique. Malheureusement, aucun fichier ne contient tous les ID de fournisseur USB, comme le fait PCI. Une recherche dans l'arborescence source du noyau est donc nécessaire:

```
grep -i -R  -l 157e drivers/*

drivers/atm/pca200e.data
drivers/atm/pca200e_ecd.data
drivers/atm/sba200e_ecd.data
drivers/net/wireless/zd1211rw/zd_usb.c
drivers/scsi/ql1040_fw.h
drivers/scsi/ql1280_fw.h
drivers/scsi/qlogicpti_asm.c
```

On sait qu'il s'agit d'un périphérique sans fil USB et non d'un périphérique ATM ou SCSI, on peut donc ignorer en toute sécurité les fichiers trouvés dans les répertoires **atm** et **scsi**. Cela laisse le nom de fichier `drivers/net/wireless/zd1211rw/zd_usb.c` à rechercher.

zd_usb.c montre la chaîne **157e** dans le bloc de code suivant:

```
static struct usb_device_id usb_ids[] = {
        /* ZD1211 */
        { USB_DEVICE(0x0ace, 0x1211), .driver_info = DEVICE_ZD1211 },
        { USB_DEVICE(0x07b8, 0x6001), .driver_info = DEVICE_ZD1211 },
        { USB_DEVICE(0x126f, 0xa006), .driver_info = DEVICE_ZD1211 },
        { USB_DEVICE(0x6891, 0xa727), .driver_info = DEVICE_ZD1211 },
        { USB_DEVICE(0x0df6, 0x9071), .driver_info = DEVICE_ZD1211 },
        { USB_DEVICE(0x157e, 0x300b), .driver_info = DEVICE_ZD1211 },
        /* ZD1211B */
        { USB_DEVICE(0x0ace, 0x1215), .driver_info = DEVICE_ZD1211B },
        { USB_DEVICE(0x157e, 0x300d), .driver_info = DEVICE_ZD1211B },
        {}
};
```

Comme les pilotes PCI, les pilotes USB indiquent au noyau quels périphériques ils prennent en charge afin que le noyau puisse lier le pilote au périphérique. Cela se fait en utilisant une variable **struct usb_device_id**, comme indiqué ici. Il s'agit d'une liste des différents ID de fournisseur et de produit pris en charge par ce pilote. La ligne:

```
        { USB_DEVICE(0x157e, 0x300b), .driver_info = DEVICE_ZD1211 },
```

indique que nos ID de fournisseur et de produit sont pris en charge par ce pilote.

Une fois que l'on a le nom du pilote nécessaire pour contrôler ce périphérique, parcourir les Makefiles du noyau, comme décrit plus haut dans le chapitre, pour déterminer comment activer correctement ce pilote.

Synthèse de la recherche

En résumé, les étapes nécessaires pour trouver le pilote USB qui contrôlera un périphérique USB spécifique sont les suivantes:

1. Rechercher le fournisseur USB et l'ID de produit du périphérique pour lequel on souhaite rechercher le pilote, en utilisant **lsusb** après avoir ajouté puis retiré le périphérique pour voir les changements dans la liste.
2. Rechercher dans l'arborescence des sources du noyau le fournisseur et l'ID produit du périphérique USB. Le fournisseur et l'ID de produit doivent être dans une définition **struct usb_device_id**.
3. Rechercher dans les Makefiles du noyau la règle **CONFIG_** qui construit ce pilote en utilisant **find** et **grep**: `find -type f -name Makefile | xargs grep DRIVER_NAME`
4. Rechercher dans le système de configuration du noyau cette valeur de configuration et accéder à l'emplacement dans le menu qui le spécifie pour permettre la construction de ce pilote.

Système de fichiers racine

Le système de fichiers racine est le système de fichiers dans lequel le noyau démarre la partie principale du système en cours d'exécution. Il contient tous les programmes initiaux qui démarrent la distribution et contient également généralement la configuration système complète de la machine. En bref, il est très important et doit pouvoir être trouvé par le noyau au démarrage pour que les choses fonctionnent correctement.

Si le noyau nouvellement configuré meurt au démarrage avec une erreur telle que:

```
VFS: Cannot open root device hda2 (03:02)
Please append a correct "root=" boot option
Kernal panic: VFS: Unable to mount root fs on 03:02
```

Le système de fichiers racine n'a pas été trouvé. Lorsqu'on on n'utilise pas d'image ramdiskau démarrage [3)], il est généralement recommandé de compiler dans le noyau à la fois le système de fichiers utilisé pour la partition racine et le contrôleur de disque pour ce disque, au lieu de l'avoir comme module. Si on utilise un disque virtuel au démarrage, il faut construire ces portions en tant que modules.

Les sous-sections suivantes montrent comment laisser le noyau trouver le système de fichiers racine lors du démarrage.

Type de système de fichiers

Tout d'abord, le type de système de fichiers utilisé par la partition racine doit être déterminé. Pour ce faire, regarder dans la sortie de la commande mount:

```
mount | grep " / "

/dev/sda2 on / type ext3 (rw,noatime)
```

Le type de système de fichiers, apparaît après le mot `type`. Dans cet exemple, il s'agit de `ext3`. Il s'agit du type de système de fichiers utilisé par la partition racine. Dans le système de configuration du noyau activer ce type de système de fichiers dans la section intitulée `Filesystems`.

Contrôleur de disque

Dans la sortie de la commande de montage montrée précédemment, la première partie de la ligne montre sur quel périphérique de bloc le système de fichiers racine est monté. Dans cet exemple, c'est `/dev/sda2`. Maintenant que le système de fichiers est correctement configuré dans le noyau, il faut également s'assurer que ce périphérique de bloc fonctionnera également correctement. Pour savoir quels pilotes sont nécessaires pour cela, il faut à nouveau consulter `sysfs`.

Tous les périphériques de bloc apparaissent dans `sysfs` dans `/sys/block/` ou dans `/sys/class/block/`, selon la version du noyau utilisé. Dans les deux emplacements, les périphériques de bloc sont une arborescence, les différentes partitions étant des enfants du périphérique principal:

```
tree -d /sys/block/ | egrep "hd|sd"

|-- hdc
|-- hdd
`-- sda
    |-- sda1
    |-- sda2
    |-- sda3
```

Compte tenu des informations contenues dans la commande `mount`, il faut s'assurer que le périphérique `sda2` est correctement configuré. Comme il s'agit d'une partition (les partitions de disque sont numérotées, tandis que les périphériques de bloc principal ne le sont pas), l'ensemble du périphérique `sda` ??doit être configuré. (Sans le périphérique bloc principal, il n'y a aucun moyen d'accéder aux partitions individuelles sur ce périphérique.)

Le périphérique de bloc `sda` est représenté par un lien symbolique dans le répertoire du périphérique appelé `device` qui pointe vers le périphérique logique qui contrôle ce périphérique de bloc:

```
ls -l /sys/block/sda

  ...
```

```
device -> ../../devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0
  ...
```

Il faut maintenant remonter la chaîne de périphériques dans `sysfs` pour savoir quel pilote contrôle ce périphérique:

```
ls -l /sys/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0

  ...
driver -> ../../../../../../bus/scsi/drivers/sd
  ...
```

Ici, on voit que le pilote du contrôleur de disque SCSI est responsable du fonctionnement de ce périphérique. On sait donc que l'on doit configurer la prise en charge des disques SCSI dans la configuration du noyau.

En poursuivant la chaîne de répertoires dans `sysfs`, essayer de trouver où se trouve le pilote qui contrôle le matériel:

```
ls -l /sys/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0

  ...
```

Il n'y a pas de lien appelé driver dans ce répertoire, donc remonter d'un niveau supplémentaire:

```
ls -l /sys/devices/pci0000:00/0000:00:1f.2/host0

  ...
```

Encore une fois, aucun pilote ici. Poursuivre sur un autre niveau:

```
ls -l /sys/devices/pci0000:00/0000:00:1f.2

  ...
driver -> ../../../bus/pci/drivers/ata_piix
  ...
```

Ici il s'agit du contrôleur de disque dont on doit s'assurer qu'il se trouve dans notre configuration de noyau.

Donc, pour ce système de fichiers racine, nous il faut activer les pilotes ext3, sd et ata_piix dans la configuration du noyau afin que l'on puisse démarrer avec succès le noyau sur ce matériel.

Script de recherche

Comme mentionné au début de ce chapitre, les fichiers et répertoires dans `sysfs` changent d'une

version du noyau à une autre. Voici un script qui est pratique pour déterminer le pilote de noyau et le nom du module requis pour n'importe quel nœud de périphérique dans le système. Il a été développé avec les développeurs du noyau responsables de `sysfs` et devrait fonctionner avec succès sur toutes les futures versions du noyau.

Par exemple, en reprenant l'exemple précédent, pour obtenir tous les pilotes appropriés pour le périphérique de bloc sda:

```
get-driver.sh sda


looking at sysfs device:
/sys/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0
found driver: sd
found driver: ata_piix
```

On peux également trouver tous les pilotes appropriés nécessaires pour les objets complexes tels que les périphériques USB vers série:

```
get-driver.sh ttyUSB0

looking at sysfs device:
/sys/devices/pci0000:00/0000:00:1d.3/usb4/4-2/4-2.3/4-2.3:1.0/ttyUSB0
found driver: pl2303 from module: pl2303
found driver: pl2303 from module: pl2303
found driver: usb from module: usbcore
found driver: usb from module: usbcore
found driver: usb from module: usbcore
found driver: uhci_hcd from module: uhci_hcd
```

Le script get-driver.sh:

```
#!/bin/sh
#
# Find all modules and drivers for a given class device.
#

if [ $# != "1" ] ; then
    echo
    echo "Script to display the drivers and modules for a specified sysfs
class device"
    echo "usage: $0 <CLASS_NAME>"
    echo
    echo "example usage:"
    echo "      $0 sda"
    echo "Will show all drivers and modules for the sda block device."
    echo
    exit 1
```

```
fi

DEV=$1

if test -e "$1"; then
    DEVPATH=$1
else
    # find sysfs device directory for device
    DEVPATH=$(find /sys/class -name "$1" | head -1)
    test -z "$DEVPATH" && DEVPATH=$(find /sys/block -name "$1" | head -1)
    test -z "$DEVPATH" && DEVPATH=$(find /sys/bus -name "$1" | head -1)
    if ! test -e "$DEVPATH"; then
        echo "no device found"
        exit 1
    fi
fi

echo "looking at sysfs device: $DEVPATH"

if test -L "$DEVPATH"; then
    # resolve class device link to device directory
    DEVPATH=$(readlink -f $DEVPATH)
    echo "resolve link to: $DEVPATH"
fi

if test -d "$DEVPATH"; then
    # resolve old-style "device" link to the parent device
    PARENT="$DEVPATH";
    while test "$PARENT" != "/"; do
        if test -L "$PARENT/device"; then
            DEVPATH=$(readlink -f $PARENT/device)
            echo "follow 'device' link to parent: $DEVPATH"
            break
        fi
        PARENT=$(dirname $PARENT)
    done
fi

while test "$DEVPATH" != "/"; do
    DRIVERPATH=
    DRIVER=
    MODULEPATH=
    MODULE=
    if test -e $DEVPATH/driver; then
        DRIVERPATH=$(readlink -f $DEVPATH/driver)
        DRIVER=$(basename $DRIVERPATH)
        echo -n "found driver: $DRIVER"
        if test -e $DRIVERPATH/module; then
            MODULEPATH=$(readlink -f $DRIVERPATH/module)
            MODULE=$(basename $MODULEPATH)
```

```
            echo -n " from module: $MODULE"
        fi
        echo
    fi

    DEVPATH=$(dirname $DEVPATH)
done
```

1)

On peut simplement essayer de rechercher dans la configuration du noyau un périphérique qui correspond à la chaîne décrite ici, un périphérique de Realtek Semiconductor avec un nom de produit RTL-8139/8139C/8139C +, mais cela ne fonctionne pas toujours. C'est pourquoi on préconise la méthode plus longue.

2)

Certains processeurs 64 bits afficheront le numéro de bus principal pour les périphériques PCI dans la sortie de lspci, mais pour la majorité des machines Linux courantes, il n'apparaîtra pas par défaut.

3)

Comment savoir si on utilise un disque virtuel au démarrage ? Lorsqu'on utilise le script d'installation fournit par une distribution pour installer le noyau, on utilise probablement un disque virtuel.