ASK KLAUS!

Klaus Knopper is the creator of Knoppix and co-founder of the LinuxTag expo. He currently works as a teacher, programmer, and consultant. If you have a configuration problem, or if you just want to learn more about how Linux works, send your questions to:

klaus@linux-magazine.com

USB Details

Can you please help me understand the details of USB device plugging and also tell me how to connect to a Seagate external USB hard drive. My machine has two USB ports on the front and several in back. How do I know if it's safe to use the front ports? Is it OK to plug devices into USB ports in a random way? If I plug my USB printer into a particular port, should I always plug the printer into that same port?

I can use both front and back ports to successfully connect a USB flash drive – I'm using openSUSE 10.2; however, I can't get the Seagate external USB hard drive to connect properly. I plug it into a rear USB port, but nothing happens.

After following advice from a web search, I did the following.

YaST detected the drive and called it */dev/sdb*. As root, I added this line to */etc/fstab:*

```
/dev/sdb   /media/seagate ➥
usbfs      auto,hotplug,➥
defaults 1 2
```

I created the directory */media/seagate* in */etc/udev/rules.d* and created *rule 98-mount.rules* with the following lines:

```
# run mount -a everytime a ➥
block device is added/removed
SUBSYSTEM=="block", ➥
run+="/bin/mount -a"
```

Unfortunately this did not work. Have I made a mistake? Can you help me connect this USB hard drive?

The details of USB device plugging are mostly invisble and therefore somewhat mysterious. Usually, the process works like this:

1. You plug in the drive.
2. Hotplug/udev detects that this is a USB drive and loads the *usb-storage* kernel module.
3. The module turns over all detected partitions to the kernel, and they appear in */proc/partitions*.
4. For udev, devices in */dev/sd** are created for every partition on the drive, and the */etc/udev* scripts are run.

Steps 1 to 3 should cause some system messages to appear in *dmesg*. Also, after the few seconds required for the USB subsystem to settle down, the partitions contained on the hard drive should appear in */proc/partitions*. To see them, type:

```
cat /proc/partitions
```

Now, if this is a fresh disk, chances are it is not partitioned yet – unless it is a flash drive, which usually has a single FAT16/FAT32 partition. Although you can use an unpartitioned disk under Linux, partitioning it is somewhat safer because the kernel will then detect the correct partition table. To do this, you could use *fdisk* or *cfdisk /dev/sdb* (in case your disk appears as *sdb* in */proc/partitions*) and then format the first partition as a Linux filesystem with *mkreiserfs /dev/sdb1* or as FAT32 with *mkdosfs -F32 /dev/sdb1*.

If you put a script for udev, like your *98-mount.rules*, into */etc/udev/rules.d*, it will be executed for this device if the file is executable and readable. Make sure that this is the case:

```
chmod 775 /etc/udev/rules.d/➥
98-mount.rules
```

I would not use a *mount -a*, though, because it mounts everything that is automountable in */etc/fstab*. This udev script mounts only the detected partitions:

```
SUBSYSTEM=="block", ➥
ACTION=="add", ➥
RUN+="/bin/mount $DEVNAME"
SUBSYSTEM=="block", ➥
ACTION=="remove", ➥
RUN+="/bin/umount -l $DEVNAME"
```

This script will try to mount each newly detected partition with the options and destination directories listed in */etc/fstab*

and will *umount* the disk on removal (although a *umount* before unplugging is strongly recommended to save as yet unwritten data to disk).

The entry you added to */etc/fstab* would match a formatted, unpartitioned disk, but *usbfs* as a filesystem is most likely wrong because *usbfs* is a virtual filesystem for describing USB devices in general, not a filesystem useful for storing data.

If you partitioned your USB hard drive with one primary partition as a FAT32 filesystem, you could use an */etc/fstab* entry such as:

```
/dev/sdb1   /media/sdb1 ➥
vfat    noauto,users,umask=➥
000    0   0
```

instead, and don't forget to *mkdir* the */media/sdb1* directory.

The mount flags *users,umask=000* will make sure you can *mount* and *umount* the drive as a normal desktop user as well – not just as root – and access all files in read-write mode.

Manual *mount*/*umount* will also work without hotplug or udev – just use the */ec/fstab* entry – provided that the *usb-storage* module is present.

```
sudo modprobe usb-storage
# wait some time...
mount /media/sdb1
```

Now the partition should be mounted and usable.

For large partitions that are being used mainly under Linux, you should probably use ReiserFS or Ext3 instead of FAT32 (vfat) because the latter can only store files up to 4GB. If you want to use the entire disk instead of creating partitions, as in the example you mentioned, the command would be:

```
mkreiserfs -f /dev/sdb
```

and the entry in */etc/fstab* would look like this:

```
/dev/sdb   /media/sdb  ➥
reiserfs    noauto,users   0  0
```

After you mount the whole disk as */media/sdb* for the first time, do a:

```
chown USER /media/sdb
```

as root, where *USER* is your normal desktop user account name, so you don't have to be root to access the disk.

Webcams

I love your "Ask Klaus," as I find it gives lots of good information that is easy to understand.

I own two different models of older Logitech Quickcam digital web cameras. The first camera is a QuickCam Express, which is worse than the other camera, but it works fine on Windows.

The second camera is a Quickcam USB camera:

- Driver: v.0.6.6
- Vendor: 045D
- Product: 0870
- Sensor: HDCS-1000/1100

(I have no Windows install CD for this second camera.)

I finally got the second camera to work in Linux, but after an automatic update to my kernel (I use Debian Etch), the camera does not work anymore. I have tried to install a proper driver, but without any success. I get some compilation error about *gcc not compatible*.

Now I want to shift to another brand of webcam, which can work, but I do not know which one to choose.

I want a webcam with a stable Linux driver that is easy to install. In previous years, using Debian Sarge, I had lots of problems with these webcams.

My digital camera works perfectly when I plug it in to the USB port.

Do you have you any clues?

Most digital cameras (those for still pictures) have a "hard disk mode" that works with the *usb-storage* kernel module, so you can just mount the camera's flash memory and read or write pictures, which is why most digital cameras will work in Linux without the need for any additional drivers.

Webcams, instead, require a video module that fetches video frames and sound from the camera at a configurable frame rate. For this, chipset-specific kernel modules are required. Unfortunately, unlike graphics cards that have a common svga standard, there are about as many different drivers as different webcams because vendors can't agree on a standard.

The gspca (Generic Software Package for Camera Adapters) module and support tools created and maintained by Michel Xhaard serves as a driver for many different webcams (about 240).

From the list of supported devices, chances are good that at least the first of your mentioned webcams, the Quickcam Express, will work. But then, vendors sometimes change their chipset without changing the name of the product.

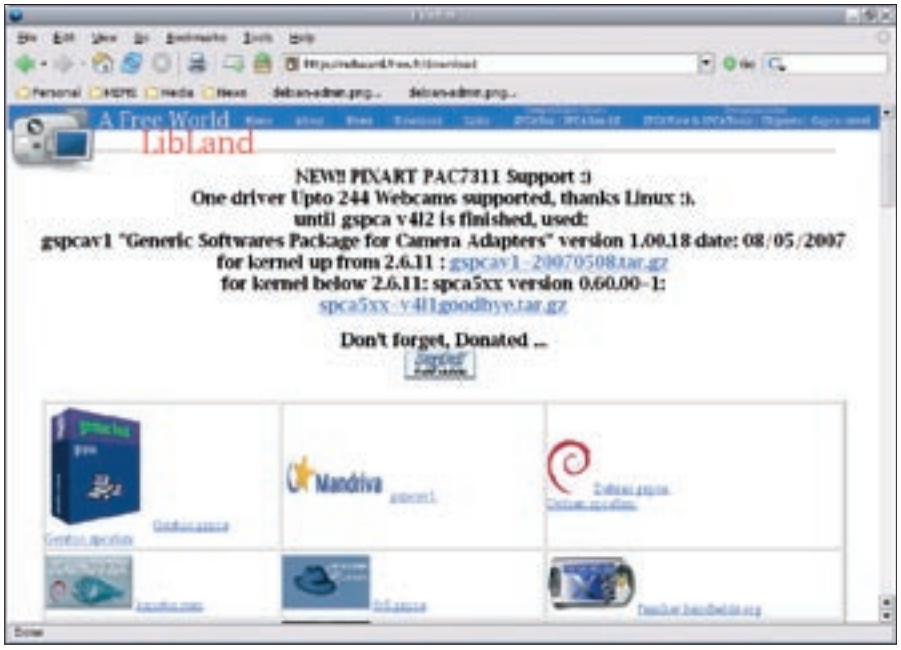

Figure 1: The gspca module supports over 240 webcams.

The gspca module is not included in the standard kernel, and it is probably also not present in most distributions, even though the license is GPL and therefore freely distributable. Hence, you will have to get the current source [1] and compile it using the kernel source of your running kernel; start with:

```
wget -c http://mxhaard.free.fr➥
/spca50x/Download/gspcav1➥
-20070508.tar.gz
```

then unpack:

```
tar zxf gspcav1-20070508.tar.gz
```

then change to the directory and complete the install:

```
cd gspcav1-20070508
make
sudo make install
```

The build process determines the kernel source location from the symlink in */lib/modules/`uname -a`/build*. If this is not the correct location, use:

```
make KERNELDIR=➥
/path/to/kernel/source
```

instead. Also, you will have to use the exact same C compiler that was used to build your kernel; you can add the *CC=/path/to/gcc-version* variable for this. If the matching C compiler is missing, it won't work, or you will have to rebuild and install the kernel with a different C compiler as well.

If your system runs hotplug, the gspca module will now be loaded automatically as soon as you plug in your camera. At least, this is the case if your camera uses a supported chipset; please check the compatibility list [2] .

To manually load the module, type (as root):

```
modprobe gspca
```

and look into the output of *dmesg* to see if anything happened that indicates the camera is detected.

Send your Linux questions to klaus@linux-magazine.com.

To get a picture, you can use gqcam, a video conferencing tool like Ekiga, or:

```
mplayer tv:// -tv ➥
driver=v4l:device=/dev/video0
```

If you have multiple video adapters, change */dev/video0* to the appropriate device and add *:width=352:height=288:* to the driver options to change the picture size.

Modems

I need help! I have installed Mandriva 2007 and later Mandriva 2007.1 (Spring), and I can't configure my modem. I've searched everywhere to find out how to run *slamr*, but it is impossible! The modem worked fine with Mandriva 2006.

This very much depends on which kind of modem you are using. Usually, a real modem consists of a controller part and a data pump, and it responds on its own to modem commands. Now the industry has introduced so-called winmodems, mostly onboard but also available as cards. These winmodems can't run with a plain serial driver. The reason is probably cost. (Winmodems are typically integrated with the onboard chipset, sometimes even as part of the sound card.)

A winmodem needs a driver or software that knows how to send and receive data from the chipset. For some reason, many hardware manufacturers seem to have "forgotten" to provide open specifications on how to write drivers for these devices, so you are either stuck with a non-working hardware component, or you have to buy a real modem, or you have to try to figure out how to make this "black box" do the same thing a real modem would do. Some winmodems from the more Linux-friendly vendors have a driver included in the ALSA sound drivers (really!) that successfully simulates a real modem with the use of userspace software that replaces missing hardware features (just as the Windows drivers would do).

Apart from the kernel-included *mwave* winmodem module and some winmodem drivers in ALSA, various chipset-specific projects handle winmodems. Please check out the Linux winmodem support page [3]. Some projects, like pctel-linux, offer modules that contain binary object files of unclear origin, so I would not necessarily call them open source. License issues could be the reason why many distributors don't offer drivers anymore and leave the installation details to the user.

By looking at the screenshots you sent, I see that your winmodem is a SiS chipset that worked with the *slamr/slmodemd* driver and userspace daemon, which is non-free (i.e., it uses an eventually non-distributable license). This driver is therefore not included in most GNU/Linux distributions, and probably no installable RPM packages are available that match the current kernel. But you can still download the driver, compile it, and install it on your own [4].

With the appropriate drivers and (in the *slmodemd* case) userspace daemon installed, the winmodem behaves (almost) like a regular modem and can be used with dialout tools like wvdial or kppp. If the newly built kernel module fails to create the device file */dev/slamr0* when loaded (*modprobe slamr*, check *dmesg*) via udev; you might have to manually create at least *slamr0*:

```
mknod -m 600 /dev/slamr0 c 212 0
```

After starting with:

```
slmodemd /dev/slamr0
```

slmodemd should automatically create a symlink */dev/ttySL0* in */dev* that is the right device for connecting to the Internet via kppp or wvdial.

Instructions are available on how to get other winmodems working [3]. Replacing your winmodem with a real modem is always an option. Regular modems just need a serial driver, and they should work with any operating system or distribution. ■

INFO

[1] Gspca source code: *http://mxhaard.free.fr/download.html*

[2] Gspca compatibility list: *http://mxhaard.free.fr/spca5xx.html*

[3] Linux winmodem support: *http://linmodems.org/*

[4] Smartlink packages: *http://linmodems.technion.ac.il/packages/smartlink/*