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

Links deleted on reboot

Several multimedia programs I use rely on the existence of "devices" such as */dev/dvd* or */dev/cdrom*. In the past, I have created a soft link as root *[user # ln -s /dev/hdc /dev/hdd; ln -s /dev/hdd /dev/cdrom]* and that was that.

Now each time I reboot, my Suse 10.1 system deletes my links, and I have to remember to set them up again (or otherwise the software "reminds me" by failing to perform). As a workaround, I have simply written a tiny script that is automagically run on reboot, but I doubt that's the best solution. So my questions for you are:

- Why are my links being deleted on reboot?
- Is there a preferred way to address this problem?

I know what you are talking about, and I agree that this problem is very annoying.

The reason for device files being magically deleted, renamed, or simply missing is that most distributions are switching from "static" device files to dynamic creation and deletion of these files, depending on what drivers are present.

In the old times, */dev* was simply a directory that contained device files (*char* devices, which showed up with a "c" in the file permissions when you entered the command *ls -l*, and block devices, which showed up with a "b"). If you tried to access a device that had no kernel module loaded to handle it, the kernel automatically loaded the driver with *modprobe*.

The philosophy of Unix has always been "everything is a file" (well, except for network cards). That philosophy still exists, but the technical basis changed when udev appeared.

With udev, */dev* is a ramdisk partition; all files in it are created at boot time by the udev daemon udevd and discarded when Linux shuts down. This explains why your changes, such as creating a device node for floppy disks with 1.7 MB capacity with

```
mknod /dev/fd0u1722 b 2 60
```

will just "disappear" on shutdown and also sometimes when using suspend-to-disk after wakeup-from-disk.

udev will create devices in */dev* when a kernel module is loaded that uses a syscall for registering a *block* or *char* device with a specific name. So the sequence now is: load a kernel module, and get a device file with a specific name.

Kernel modules to load at boot time are listed, in most distributions, in */etc/modules*. If you just need a working device file with the default name, it's sufficient to add the matching module name to */etc/modules*. But sometimes, you would also like to have the symlinks or

Listing 1: Overformatted Floppy Drive Device File

```
01 # This file does not exist.
   Please do not ask the debian
   maintainer about it.
02 # You may use it to do
   strange and wonderful things,
   at your risk.
03 ...
04 L core          /proc/kcore
05 L sndstat       /proc/asound/
   oss/sndstat
06 ...
07
08 # Hic sunt leones.
09 M ppp           c 108 0
10 D loop
11 M loop/0        b   7 0
12
13 # My own rules, create a
   cdrom symlink to hdc,
14 # and a floppy with extended
   capacity.
15 L cdrom         hdc
16 M fd0u1722      b   2 60
```

"additional feature" devices that you used prior to udev.

Rather than writing an init script that creates those files "manually," you could use a more or less "unofficial" feature. The file */etc/udev/links.conf* (as seen in Debian GNU/Linux) contains lines that are interpreted when udev is started with its own init script.

In Listing 1, the command *L cdrom hdc* will cause a *ln -s hdc* in the */dev* directory, while *M fd0u1722 b 2 60* will do a *mknod /dev/fd0u1722 b 2 60*, thus creating the "overformated floppy drive" device file.

Note that the funny comments about this file not existing, and the lions, are from the maintainer of the udev package. Of course the file exists, and it is interpreted when udev is started by its own init script */etc/init.d/udev.* The *links.conf* file may not be present in all distributions (though most will have a similar mechanism for adding your own stuff), and maybe this is even subject to change in future Debian releases, but so far, it has proven very helpful.

If you have a distribution that does not implement such a file, you can still use udev's own mechanism for creating symlinks and devices. Check out */etc/udev/cd-aliases.rules* (if you have an older version of udev) or */etc/udev/cd-aliases-generator.rules* (for udev versions > = *0.098*). You may have to get familiar with the udev config syntax (which is not too difficult) to create the desired symlinks, but it's worth a try.

When you called suspend-to-disk, and you notice that some symlinks and devices are gone after resume, you will have to call

- *udevsynthesize* (for "older" versions of udev), or
- *udevtrigger* (for current versions of udev)

from within your suspend/resume script. This will cause udev to rescan for devices and execute the */etc/udev/rules.d/** scripts, as it does on a regular bootup.

You may also be wondering, why does udev have to be located on a tmpfs ramdisk, thus recreating all the device files on every reboot? Using a regular directory on the hard disk seems to make persistent device naming a lot easier. But then, udev tries to keep track of devices internally; it may get confused if device files or symlinks are already there and create duplicate files with different names. I haven't experimented too much with this yet.

More Fedora Core 5

When I changed from an AMD 32-bit to an AMD 64-bit processor, I was unable to access the files on the main hard drive that contain my personal information. Knoppix 5.0 was a lifesaver, enabling me to copy these files and rescue them. I would like to install it (like other distros) but I haven't figured out how yet.

I have installed Fedora Core 5, and it is mostly working, but several annoyances remain. It seems impossible to use kppp to activate a modem without supplying the root password. Everything in sight has been set and reset for user access, setuid, etc. (*/usr/bin/... /usr/sbin/... consolehelper*), but nothing suppresses the request for the root password.

Also I can't change the kppp icon.

This is a single user machine, but I can't find any way to avoid a login with password.

Finally, there are lots of pieces of emacs installed on the system, but I haven't found a way to actually run emacs.

Maybe you can comment on some of these problems.

(ed. note: The reader writes back with the following update.)

I found a solution to the first problem on the Fedora support pages: I just changed the */usr/bin/kppp* link to *consolehelper*. Other problems persist.

Your remarks remind me that Knoppix 5.1 should have been out already, but there are still some things to be fixed related to Kernel 2.6.19-rc*. By the time this article is published, it should be available.

About 64-bit vs. 32-bit processors: most (all?) 64-bit processors have a compatibility mode that allows you to run programs written for 32-bit processors of the same type. But the binary code (the machine instruction set) is different, which means, you CAN run 64-bit and 32-bit programs in parallel, but you cannot mix 32-bit with 64-bit code. And here is the problem: when you are using a program with 64-bit code, the supporting libraries must all be 64-bit as well. If the "bitness" is not consistent, you may experience all kinds of funny effects, ranging from parts of the program not working correctly (hence the failure to load files that are otherwise OK to read) to sudden crashes with "illegal instruction" errors.

Since Knoppix (the main branch) is plain 32-bit code in all programs, it will work fine on i*86 compatible 64-bit CPUs; not as fast as native 64-bit environment, but probably not noticeably slower, unless you do a lot of number-crunching. This may explain why you were able to access your personal files.

So, if you run a 64-bit operating system with applications, make sure all the applications run in 64-bit, and that there are no libraries, modules, or plugins that still use 32-bit code. For example, a 32-bit compiled OpenOffice will start in 64-bit mode, but as soon as it starts using plugins and libraries from your 64-bit environment, things are likely to get weird.

If you have to run 32-bit applications, it is wise to use a chroot into a 32-bit GNU/Linux installation, so these applications run only with 32-bit libraries. Though I would not recommend this, if you still would like to install a 32-bit Knoppix CD or DVD as Debian OS on 64-bit hardware, you can use the knoppix-

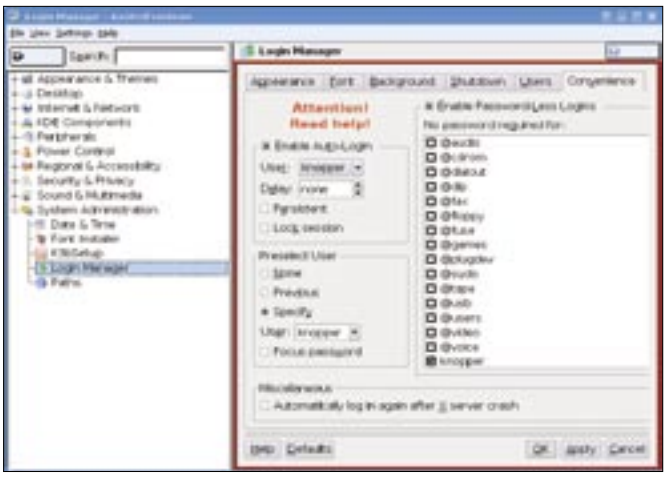

Figure 1: Configuring passwordless login for KDM.

installer. (We are working on a new installer, by the way.) But it's probably better and more efficient for your new hardware to use a native 64-bit Debian installation and migrate all 32-bit dependent data to the new system. You will probably want to use one or the other 64-bit-based program some day, and you would eventually stumble over the same problems you have seen before when using 32-bit code in a 64-bit environment.

About kppp (and some other KDE programs) always asking for a root password: some KDE (and Gnome) programs are designed not to run with the set-user-id file attribute. Therefore, they have to be started with su or sudo. Knoppix uses sudo to switch to the root user before starting kppp. If you want to be able to do this, passwordless, you can use this entry in */etc/sudoers*:

```
your_login ALL=NOPASSWD: ALL
```

You should be aware that this setting introduces all kinds of security problems related to the execution of malicious scripts your browser or mail client may be enticed into starting automatically as root. These scripts will now be a greater threat, whereas they would only have destroyed your user files if they were forced to start under your login id. But then, it is also a securiy risk to have to type your root password so frequently that it ends up one day via cut&paste in your chat window…

When enabling passwordless sudo, you can change the "command" part in the .desktop setting for a program to *sudo [-H] command [options]* to start it as root, rather than using kdesu or gksu.

But even with the graphical kdesu or gksu, you have a (clickable) option to "remember password," so you don't have to type the root password again in the next attempt. Accidentially clicking this option may have been the reason why kppp did not ask you for a password some times during your experiments. I'm not convinced yet that this "remember" option is more secure than the sudo method. It is never good to store passwords on disk, even with (sometimes too weak) encryption.

For passwordless login, you can use a Knoppix-like method with

```
su -c xinit - your_login
```

instead of launching xdm, gdm, or kdm as display and session manager inside an init script.

But kdm also allows you to auto-login with a dedicated user. Use the KDE control panel for configuring this option under *System Management* | *Login Manager* (Figure 1).

About your emacs question: maybe you have several emacs versions installed – maybe even xemacs and emacs in parallel (which is possible, though the location and naming of configuration files and extensions can be confusing). Rather than finding out how many pieces of emacs are currently installed, the easiest solution is to just completely remove everything emacs-related with *rpm -qa | grep emacs* and reinstall emacs (+plugin) RPMs with the same version number.

SATA Drive

I have a PC with a SATA hard disk. I have installed Linux before, but only on computers with ATA hard disks. Now when I try to install on a computer with a SATA disk, I get the message "No hard drives found" during the installation.

I don't know what to do. I have posted a lot of requests on the Internet, but so far I haven't received any useful suggestions. Can you please help me?

If you use a search engine for the question "Does my SATA controller work with Linux?", you will probably find a lot of recommendations on which Linux distribution to use. But what you really want to know is if your SATA controller is recognized and working at all. If your SATA controller is working, you can then choose a distribution that you like and get the necessary support for your controller.

The place to look for supported hardware is the device list of supported hardware in the current Linux kernel. Most SATA controllers are indeed well supported by the 2.6.x kernel series; have a look here to find out if yours is listed: *http://linux-ata.org/driver-status.html*

However, depending on your specific GNU/Linux distribution, the latest kernel may not be available in the installer process. Therefore, you may get the error message you described about missing hard drives.

If this happens, you still have some options. Rather than trying to modify the installer CD or DVD by yourself by adding a newer kernel, you could boot a different distribution with support for your controller in rescue mode, do a chroot to the mounted installer medium, and run the installer inside the chroot environment. This is still kind of an advanced installation hack, since you will still need to configure and install a different kernel after the main installation to match your hardware.

Another possibility would be using an old-style IDE setup for installation, then (afterwards) upgrading the kernel and copying the entire installation to a SATA disk, and changing the device names in */etc/fstab* accordingly.

Some computer BIOSes allow you to enable "legacy IDE" emulation for SATA disks, so they will look like "normal" IDE disks. This legacy IDE emulation is not a common option, but you may want to check for it. ■

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