Dr. Linux

EXORCISE YOUR DAEMONS

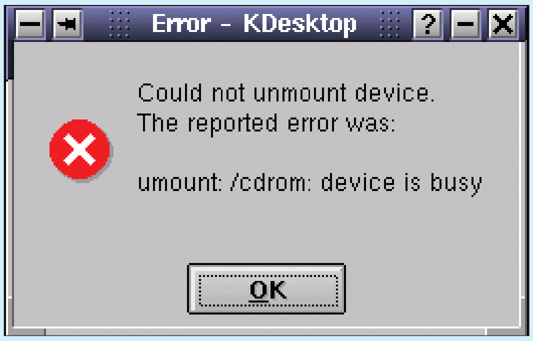
Unix systems are not for the faint-hearted, as the world of processes is swarming with zombies and daemons. Marianne Wacholz takes us on a trip into the crypts of Linux

Which program is tying up the drive?

Q After I had looked at a CD with graphics on my Linux computer, I had the following problem: when I tried to *unmount* the CD drive, this error message appeared:

```
unmount: /media/cdrom: The device is busy.
```

How can I find out which program is preventing an **unmount?**



```
Error - KDesktop
Could not unmount device.
The reported error was:

umount: /cdrom: device is busy
```

Figure 1: Which program is holding onto the drive?

Dr. Linux: *You will get this error message, or its graphical equivalent as shown in Figure 1, if there is an open file tying up the device which you are trying to close.*

You can find which **process** *has opened this file quite simply with the command* lsof *("list open files"). This command can be used in so many ways that the associated manpage contains over 2,000 lines. It is usually found under* /usr/sbin *– should your search path not include this directory, you must instead call up the command with the full path included,* lsof *would be* /usr/sbin/lsof.

As an argument, give it the device name and, where applicable, the path of the file which you want to know about. In the case of a blocked CD drive, this would be the path of the corresponding device file. Without specifying some object for lsof *to look at, you will find yourself presented with a list of all open files on your system, and in all but exceptional cases this will prove to be very long.*

If you have yet to acclimatise yourself to the way in which a Unix system describes its devices, having spent years learning those Windows drive letters, you can quickly look it up with the command mount. mount *without further details lists all currently mounted drives, obviously including the CD which refuses to allow itself to be unmounted:*

```
perle@maxi:~> mount
/dev/hda7 on / type ext3 (rw)
[...]
/dev/hdb on /media/cdrom type iso9660
(ro,nosuid,nodev,user=perle)
```

In the example the CD-ROM drive is mounted as slave on the primary IDE controller (/dev/hdb*), and the data contained on the CD can be accessed under the directory* /media/cdrom *in the Linux file hierarchy. You can tell from the filesystem type* iso9660 *that this is a data CD.*

mount and unmount

Data devices are integrated in the Linux file tree with the root-reserved *mount* command. Before you can remove a mounted CD or diskette from the drive, an *unmount* command is vital. In the file */etc/fstab* the system administrator can stipulate that unprivileged users have the right to *mount* and *unmount* certain data devices such as CD-ROMs, or, more importantly, refusing them access. The same is true for hard disk partitions, which can escape access under Linux.

Dr. Linux

Complicated organisms, which is just what Linux systems are, have some little complaints all of their own. Dr. Linux observes the patients in Linux newsgroups, issues prescriptions here for the latest kernel problems and proposes alternative healing methods.

An lsof /dev/hdb *now comes up with an output as in Listing 1. To find out which tasks are accessing data on the CD, let's take a close look at the following output columns:*

Listing 1: Example outputs from lsof

```
perle<\@>maxi:~> lsof /dev/hdb
COMMAND  PID      USER     FD       TYPE     DEVICE   SIZE     NODE     NAME
gs       1252     perle    3r       REG      3,64     593581   47726     /media/cdrom/autoren.pfd
```

- *The* COMMAND *column outputs the accessing command, possibly abbreviated to nine letters.*
- PID *contains the* Process Identification *number, which will be required, for example, should you want to send the corresponding task to its grave with the* kill *command.*
- *In the third column, overwritten with* USER*,* lsof *gives us the name of the user who started the ball rolling with the task, though sometimes this will just be his or her user-ID (UID).*

If, under COMMAND*, you find a program which you did not even start and want to get right to the bottom of this matter, it's worth using the classic* pstree *command. With this command you get a display of tasks in tree form, so that you quickly get an overview of which tasks have started which other tasks. With the option* –p *the* pstree *output also includes the PIDs (Figure 2).*

There are many graphical user interfaces and programs that allow you to administer tasks, which you are welcome to use once you have found one that suits your needs. However, remember that pstree *will be available on all systems, even an old one, so keep a healthy respect for it.*

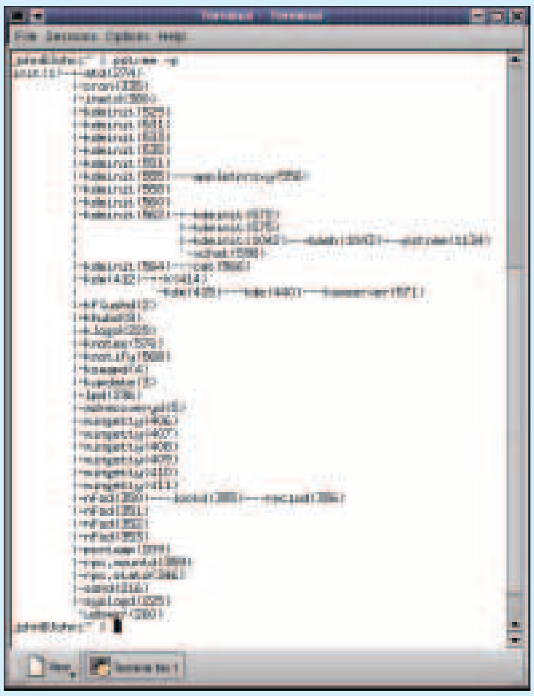


Figure 2: pstree displays tasks in tree form

If there are tasks preventing the unmounting of the CD which you started under your User-ID, you can shoot them down with the command kill *and the respective PID as argument. Sometimes a task will not react to this and may need a firmer hand by sending progressively more threatening signals until it realises who the boss is. Starting with* kill –15 <PID> *will, hopefully, get the task to close in a clean way, but if this fails you will need the heavy-handed* kill –9 <PID>*.*

```
perle@maxi:~> kill -9 1252
```

The tasks of other users can only be shot down by root; it goes without saying that when you are equipped with root powers you must proceed with great caution. As soon as the task in question has breathed its last, there is no longer an obstacle to an unmount of the CD.

Is it running or isn't it?

Q When configuring utilities which are controlled by **daemons** (for example *cron*), I often find in the documentation the demand to check whether the respective daemon or the program is running. How do I ascertain this quickly and easily?

Dr. Linux: *In the* /var *sector of the Linux directory tree you will find data which can be changed quickly, ergo is variable. This also includes the information as to whether a certain daemon is running. If it is started, it also receives (like every other task) a task-ID, but one which, unlike "normal" tasks, is recorded in a file named "*name.pid*" in* /var/run*. This prevents one of the daemons being started twice, and when the system prepares itself to be powered down it is immediately apparent which utilities must be shut down first, which is why you don't just turn a Linux box off.*

If you find an entry matching a daemon in /var/run*, you can normally assume that the corresponding utility has been started (Listing 2). Obviously it may be that it is doing nothing at all at this precise moment and is just passing the time until its next appointment; this depends on its task and the respective configuration. If you need the task-ID, it is best to look at the content of the file with* cat *– the output is limited to just one number, so that the use*

Process The operating system kernel has direct access to the resources of the computer, for example memory and computing time. If a command is invoked or a program started, it loads the necessary program code into the main memory. Once started thus, this program is now referred to as a task. Each task has a unique task number (PID), which the system keeps in a task table. Tasks have no access to resources; they request these as required from the kernel. If the same program, the command *gimp* for example, is started twice then this usually involves two different tasks, although the same program is executed. The operating system kernel allocates the necessary computing time and the memory so quickly that it gives the impression that programs can run simultaneously. Tasks can multiply themselves by creating child tasks by means of duplication; when this happens they themselves are referred to as parent tasks. Parent tasks can wait for their children tasks to end or die; this does not work the other way around.

Daemons An abbreviation for *Disk and Execution Monitor*. Daemons are not an integral part of the system kernel, but programs which run in the background and make their services available to other programs or computers. Some are started when booting and remain active throughout the running time of a system; others are active only as long as their services are required.

Listing 2

The .pid-files in /var/run contain the PIDs of started daemons

```
perle@maxi:/var/run> ls -l
total 112@l =  -rw-r--r--    1 root    root       4 Mar    6 09:26 atd.pid
               -rw-r--r--    1 root    root       4 Mar    6 09:26 cron.pid
               -rw-r-----    1 root    root       4 Mar    6 09:26 gpm.pid
               -rw-r--r--    1 root    root       4 Mar    6 09:26 inetd.pid
               -rw-r--r--    1 root    root       4 Mar    6 09:26 klogd.pid
               -rw-r--r--    1 lp      lp         4 Mar    6 09:26 lpd.printer
               -rw-r--r--    1 root    root       4 Mar    6 09:26 nscd.pid
               drwxr-x--T    2 root    root       4096 Mar 6 10:18 sendmail
               -rw-r--r--    1 root    root       38 Mar   6 10:18 sendmail.pid
               drwxr-x---    2 root    dialout    4096 Mar 6 09:26 smpppd
               -rw-r--r--    1 root    root       4 Mar    6 09:26 sshd.pid
               -rw-r--r--    1 root    root       4 Mar    6 09:26 syslogd.pid
               -rw-rw-r--    1 root    tty        3456 Mar 6 10:29 utmp
               -rw-r--r--    1 root    root       4 Mar    6 09:26 xfstt.pid[...]
```

of a pager such as less *would be an extravagance:*

```
perle@maxi:/var/run> cat cron.pid
599The cron daemon currently running thus bears
the PID 599.
```

To further your conviction that a program really is alive and kicking, there is a link to the program killall5 *with the command* pidof *(under SuSE this is in* /sbin*, rather than in the usual user path). Give* pidof *the name of the wanted program, which in this case does not even have to be a daemon. If it is running, you will receive as output its PID; if it has been started more than once,* pidof *outputs all task identification numbers.*

```
perle@maxi:~>/sbin/pidof /sbin/syslogd
337
```

If pidof *says nothing after your input, you are dealing with a script (or with something which* pidof *believes to be one, though this belief does not have to be correct). In this case specify the option* –x *too:*

```
perle@maxi:~> /sbin/pidof -x kdeinit
1125 929 927 923 921 919 909 895 893 89
```

Hollywood’s nightmares in the system?

Q When I start *top*, in order to check that the system is running, I sometimes find one or more zombies in the system (Listing 3). What does this mean?

Dr. Linux: *The Film “Dawn of the Dead” defines the term Zombie thus: “When there’s no more room in Hell, then the dead come back to earth.”. Anyone who is now shuddering at his or her Linux system can sit back and relax, because there is also death in the Unix world. This applies to tasks – specifically when they stop on their own or are shut down. A zombie is a dead task, whose* **Exit–Status** *continues to be kept by the kernel in the task table and waits for its parent process to read it. Only then can it be deleted in peace from the task table. If the parent process dies before it has read this register value, the zombie is also deleted.*

Exit–Status A register value, defined by the programmer, which defines how a program departs this life – either successfully, on the grounds of a bug or otherwise.

Background or foreground?

Q I know that I can start programs in the background on a command line, if I put an *&* after the command, but which command do I use to get one of several background tasks back into the foreground, so that I can shut it down using Ctrl+C for example?

Dr. Linux: *Programs sent into the background by a command line, so as not to block the console, are referred to as* Jobs*. When a job is started, the command is also given, in addition to its task number, a figure in brackets, the so-called* Job Number*.*

```
perle@maxi:~> emacs &
[1] 1650
```

The background tasks can be listed for each console (and each X terminal) with the command jobs*:*

```
perle@maxi:~> jobs
[1]   Running emacs &
[2]-  Running gimp &
[3]+  Running xtetris &
```

The active background programs are marked with Running*. Other possibilities are* Stopped *for*

Listing 3: top sees a zombie

```
1:01am  up 13:29,  1 users,  load average: 0.25, 0.19, 0.12
88 processes: 86 sleeping, 2 running, 1 zombie, 0 stopped
CPU states:  7.2% user,  8.0% system,  0.0% nice, 84.6% idle
Mem:   320072K av,  293952K used,   26120K free,       0K shrd,
16496K buff
Swap:  305152K av,    3276K used,  301876K free
140224K cached

PID     USER     PRI     NI     SIZE     RSS     SHARE     STAT     %CPU     %MEM     TIME     COMMAND

1719    perle    19      0      9224     8860    7164      R        8.0      2.7      0:25     kdeinit
1286    root     16      0      22264    13M     1720      S        5.7      4.3      9:40     X
4808    perle    14      0      1096     1096    792       R        1.3      0.3      0:09     top
1       root     9       0      208      208     176       S        0.0      0.0      0:04     init
[...]
1445    root     9       0      0        0       0         Z        0.0      0.0      0:00     cron[...]
```

interrupted programs, Terminated *or* Done*. You will also come across minus and plus signs. + indicates the most recently started background process, – the previous ones. These signs can be handed over as arguments to the command* fg *with a preceding percentage sign (%), which brings the background process to the foreground. Other possible arguments include:*

- %n*, where* n *must be replaced by the job number, i.e. the figure placed in brackets.*
- %e*, which keeps the job whose command line commences with the character string* e*, in the foreground. If more than one background command fits the character string, you will get the error message:*

```
bash: fg: ambiguous job spec: e
```

- %?s*, which brings forward the background process, whose command line contains the character string* s *(or complains about an ambiguous job specification).*
- %% *or* %+*, which both mean the current job.*
- %–*, which stands for the previous job.*
- fg *is not the only command which helps in job administration in bash. The following commands also accept the arguments just described above:*
- bg *sends a job into the background. So that* bg *cannot enter into a shell and block a foreground process, this must be temporarily be stopped with* Ctrl+Z*.*
- kill *ends the job specified as argument. If the pattern matches several tasks, however, there will be an error message.*
- *The listing of the jobs with* jobs *can be limited with the aforementioned arguments to certain tasks. In practice this looks something like this:*

```
perle@maxi:~> jobs
[1]   Running emacs &
[2]-  Running gimp &
[3]+  Running xtetris &
perle@maxi:~> kill %1
perle@maxi:~> jobs
[1]   Stops emacs
[2]-  Running gimp &
[3]+  Running xtetris &
```

Before you shoot down file processing programs such as Emacs, you should however pause for a moment. With kill *you are in fact also moving the files you've not backed up into oblivion. If you leave the console from which the program was started, with* exit*, though, the editor will be kept open for you and you can still back up your data.*

If, in the terminal to be closed down with exit*, there are still some interrupted jobs waiting, you will be gently reminded of this fact:*

```
There are stopped jobs.
```

Only after a second exit *will the console take its leave of you.*