3.3 Grundkonfiguration/Systemzustand ermitteln

3.3.1 Hardware der Zentraleinheit

Die Zentraleinheit eines Computers umfasst im allgemeinen die CPU, das RAM (Hauptspeicher), das ROM (z. B. BIOS oder UEFI) und einige Ein-/Ausgabe-Einheiten für Tastatur, Maus und Bildschirm. Mit den folgenden Befehlen kann man Informationen dazu erhalten:

Terminal

```
root@debian964:~# cat /proc/cpuinfo | less # CPU
root@debian964:~# lspcu|less               # CPU
root@debian964:~# cat /proc/meminfo | less # RAM
root@debian964:~# memfree | less           # RAM
root@debian964:~# biosdecode | less        # ROM
```

3.3.2 Ein-/Ausgabegeräte

Den weitaus größten Teil eines Rechnersystems nehmen traditionell die Ein-/Ausgabe-Einheiten ein. Sie dienen dazu, den Rechner mit den Geräten für die Benutzer-Ein- und Ausgabe zu verbinden (Tastatur, Maus, Mikrofon, Scanner, Kamera; Bildschirm, Drucker, Lautsprecher usw.) sowie mit Massenspeichern und dem Netzwerk.

Die CPU kann nun auf bis zu vier Arten mit den Ein-/Ausgabe-Einheiten kommunizieren:

a) I-/O-Port — Jede E-/A-Einheit braucht eine eigene Adresse, unter der sie ansprechbar ist, den sogenannten *I-/O-Port*. Diese Adresse liegt bei PC-Prozessoren in einem eigenen Speicherbereich, der über eine eigene Steuerleitung angewählt wird. Sobald die CPU den Befehl `in` oder den Befehl `out` ausführt, setzt sie diese Steuerleitung auf aktiv.

b) RAM-Bereich — Grafikkarten schleusen in jeder Sekunde große Datenmengen aus dem System. Hier ist das Konzept der I-/O-Ports überfordert. Stattdessen stiehlt man in diesem Fall dem System einen Teil der möglichen RAM-Adressen und benutzt ihn für eine direkte Ausgabe mit beliebigen CPU-Befehlen.

c) IRQs — Das laufende Programm soll auf Tastendruck unterbrochen werden: Dazu sendet die Tastatur einen Interrupt an die CPU. Die CPU unterbricht ihr laufendes Programm und bearbeitet den Tastendruck. Danach läuft das Programm weiter. Damit das funktioniert, hat die CPU mehrere Eingänge, die für eine Unterbrechung sorgen. Sie heißen IRQ-Eingänge (*interrupt request*). Im Idealfall besitzt jede E-/A-Einheit eine eigene IRQ-Leitung, die zu einem IRQ-Eingang der CPU führen. Mit Hilfe eines IRQ-Controllers können sich mehrere Geräte einen gemeinsamen IRQ-Eingang an der CPU teilen.

d) DMAs — DMA (*direct memory access*) ist eine Technik, mit der E-/A-Einheiten direkt mit dem RAM kommunizieren können. Sie umgehen damit die CPU. Für solch eine Kommunikation gibt es im PC mehrere Kanäle. Aufgrund der Leistungsfähigkeit heutiger CPUs hat diese Technik an Bedeutung verloren.

Bei diesen bis zu vier Kommunikationsmöglichkeiten ist zu beachten, dass jedes Gerät eindeutig identifizierbar sein muss. Wenn die Tastatur über den I-/O-Port 0x1234 ansprechbar sein soll, kann die Maus nicht ebenfalls denselben I-/O-Port verwenden. Jedes Rechnersystem hat nur eine begrenzte Auswahl an I-/O-Ports, RAM-Bereich, IRQs und DMA-Kanälen. Es handelt sich um begrenzte Ressourcen.

Traditionell war es so, dass ein Gerät entweder feste oder von Hand einstellbare Ressourcen besaß. Der Administrator musste diese Ressourcen (in der Regel den I-/O-Port und die IRQ-Nummer) seiner Geräte wissen. Dann musste er dem Treiber mitteilen, über welche Ressourcen er (der Treiber) auf das Gerät zugreifen sollte. Dazu bekam der Treiber beim Aufruf die Ressourcen mitgeteilt – zumindest dann, wenn sie nicht der Standard-Konfiguration entsprachen.

Heute ist das oft einfacher: Viele Geräte liefern auf Anfrage selbst alle möglichen Informationen über sich, etwa über den PCI-Systembus, über den USB oder über die Serial-ATA-Schnittstelle. Die Treiber finden die Geräte dann (hoffentlich) anhand dieser Informationen.

Wie kommt man nun an die Informationen heran? An die traditionellen Geräte, die sich nicht dem System vorstellen, kommt man gar nicht heran[1]. Es gibt einige traditionelle Geräte, die seit Urzeiten des PCs an bestimmten I-/O-Ports angeschlossen sind (heutzutage im Chipsatz versteckt); sie sind unproblematisch.

Für alle modernen Geräte bietet es sich an, die PCI- und USB-Controller abzufragen:

Terminal
```
root@debian964:~# lspci -vvv|less
root@debian964:~# lsusb -v|less
```

Für den Serial-ATA-Controller gibt es leider keinen solchen Befehl.

Eine andere Möglichkeit besteht darin, die Systemtabellen abzufragen. Das sogenannte `/proc`-Dateisystem bietet dazu die Möglichkeit:

Terminal
```
root@debian964:~# cat /proc/ioports
root@debian964:~# cat /proc/iomem
root@debian964:~# cat /proc/interrupts
root@debian964:~# cat /proc/irq/*
root@debian964:~# cat /proc/dma
```

Eine weitere umfangreiche Quelle für Informationen ist das modernere `/sys`-Dateisystem (*sysfs*):

Terminal
```
root@debian964:~# ls -l /sys/dev/block # Massenspeicher
root@debian964:~# ls -l /sys/dev/char  # Sonstige Geräte
```

3.3.3 Geräte-Dateien

Wenn der Normalbenutzer auf einen Massenspeicher zugreifen will, macht er das mit Hilfe einer Geräte-Datei. Über diese Geräte-Datei spricht er mit dem Treiber. Der wiederum spricht über die oben genannten Ressourcen mit dem Gerät.

Die Geräte-Datei liegt wie jede andere Datei auch im Dateisystem. Das Verzeichnis `/dev` (für *devices*) ist dafür vorgesehen. Traditionell war das Verzeichnis `/dev` statisch. Mit dem Befehl `mknod` legte man hier zu einem neuen Gerät die entsprechende(n) Datei(en) an. Heutzutage, da Linux auch mit wechselnden Datenträgern betrieben werden kann, muss das automatisch erfolgen. Dazu gibt es den Dienst `udev`. Er legt (mit Hilfe des `/sys`-Dateisystems) zu jedem neu erkannten Gerät und Treiber die entsprechenden Einträge unterhalb von `/dev` an. `udev` löscht die Einträge wieder, sobald das Gerät entfernt wurde.

Für Massenspeicher gibt es bestimmte Konventionen, was die Namen der Gerätedateien angeht:

Terminal
```
schueler@debian964:~$ ls -l /dev/
brw-rw---- 1 root disk  8,  0 Feb 1 11:28 sda  # 1.Festplatte
brw-rw---- 1 root disk  8, 16 Feb 1 11:28 sdb  # 2.Festplatte
brw-rw---- 1 root disk  8, 32 Feb 1 11:28 sdc  # 3.Festplatte
brw-rw---- 1 root disk  8, 48 Feb 1 11:28 sdd  # 4.Festplatte
brw-rw---- 1 root disk  8, 64 Feb 1 11:28 sde  # 5.Festplatte
brw-rw---- 1 root disk  8, 80 Feb 1 11:28 sdf  # 6.Festplatte
brw-rw---- 1 root disk  8, 81 Feb 1 11:28 sdf1 # 6.Festpl., 1.Part.
brw-rw---- 1 root disk  8, 82 Feb 1 11:28 sdf2 # 6.Festpl., 2.Part.
brw-rw---- 1 root disk  8, 85 Feb 1 11:28 sdf5 # 6.Festpl., 5.Part.
```

[1] Man könnte per Polling einfach alle I-/O-Ports abfragen und dabei nachsehen, an welchen Adressen verdächtige Werte anliegen.

```
brw-rw---- 1 root disk  8, 86 Feb 1 11:28 sdf6 # 6.Festpl., 6.Part.
brw-rw---- 1 root disk  8, 87 Feb 1 11:28 sdf7 # 6.Festpl., 7.Part.
```

Die alten IDE-Festplatten heißen entsprechend:
`/dev/hda` usw.,
optische Laufwerke:
`/dev/cdrom0` und `/dev/dvd0`.
Die aktuell eingebundenen Platten kann man so anzeigen:

Terminal
```
schueler@debian964:~$ df -h | less
schueler@debian964:~$ mount | less
```

3.3.4 Kernel und Kernel-Module (Treiber)

Bei Linux kann man wählen zwischen einem monolithischen Kernel und einem modularen Kernel. Beim monolithischen Kernel sind alle Treiber fest in den Kernel eincompiliert. Das bedeutet ein Plus an Sicherheit und Geschwindigkeit. Beim modularen Kernel können jederzeit neue Treiber eingebunden werden. Das bedeutet natürlich eine höhere Flexibilität.

Im folgenden gehen wir von einem modularen Kernel aus (weil der monolithische Kernel eher problemlos ist). Zunächst ist es sinnvoll, nachzusehen, welcher Kernel gerade läuft:

Terminal
```
schueler@debian964:~$ uname -a # _alle_ Info
schueler@debian964:~$ uname -r # Kernel-Version
```

Die Treiber findet man so:

Terminal
```
schueler@debian964:~$ ls /lib/modules/`uname -r`/kernel/
```

Das automatische Einbinden der Treiber wird von `kerneld` (alt) oder `kmod` (neu) bewerkstelligt. Optionen kann man in der Datei `/etc/modules.conf` angeben.

Zunächst ist es sinnvoll, die geladenen Treiber anzuzeigen:

Terminal
```
root@debian964:~# cat /proc/modules # anzeigen aller Treiber
root@debian964:~# lsmod             # anzeigen aller Treiber
...
sr_mod                  24576  0
cdrom                   49152  1 sr_mod
scsi_mod               180224  8 sd_mod,usb_storage,libata,sr_mod,...
```

Die erste Spalte zeigt den Namen des Treibers, danach folgen die Größe in Byte und die Anzahl der von diesem Treiber abhängigen Treiber. Die letzte Spalte zeigt diese abhängigen Treiber an.

Wenn man z. B. `cdrom` löschen will, dann klappt das nicht, weil zuerst `sr_mod` gelöscht werden müsste:

Terminal
```
root@debian964:~# rmmod cdrom # versuche cdrom zu loeschen
rmmod: ERROR: Module cdrom is in use by: sr_mod
root@debian964:~# rmmod sr_mod # zuerst dies
root@debian964:~# rmmod cdrom  # dann das: jetzt klappt es
```

Grundregel: Man kann nur Treiber löschen, bei denen in der vorletzten Spalte eine 0 steht.

Zum Einbinden eines Treibers kann man `insmod` benutzen. `insmod` braucht aber den vollständigen Pfadnamen.

Terminal
```
root@debian964:~# insmod sr_mod
insmod: ERROR: could not load module sr_mod: No such file or directory
root@debian964:~# insmod \
   /lib/modules/`uname -r`/kernel/drivers/scsi/sr_mod.ko
```

Wenn ein Treiber fehlt, von dem sr_mod abhängt, gibt es ein weiteres Problem:

Terminal
```
root@debian964:~# rmmod sr_mod # wir loeschen mal sr_mod
root@debian964:~# rmmod cdrom  # und cdrom loeschen wir auch
root@debian964:~# insmod \
   /lib/modules/`uname -r`/kernel/drivers/scsi/sr_mod.ko
insmod: ERROR: could not insert module /lib/.../sr_mod.ko: Unknown
symbol in module
```

Wenn man denn weiß, welches Modul zuerst geladen werden muss, dann kann man das natürlich machen:

Terminal
```
root@debian964:~# insmod \
   /lib/modules/`uname -r`/kernel/drivers/cdrom/cdrom.ko
root@debian964:~# insmod \
   /lib/modules/`uname -r`/kernel/drivers/scsi/sr_mod.ko
```

Einfacher ist da der Befehl modprobe: Er braucht nur den Namen des Treibers und lädt alle benötigten Treiber mit dazu:

Terminal
```
root@debian964:~# rmmod sr_mod # wir loeschen mal sr_mod
root@debian964:~# rmmod cdrom  # und cdrom loeschen wir auch
root@debian964:~# modprobe sr_mod # laedt _beide_
```

Woher weiß das Programm modprobe, welche Module sr_mod braucht? Dazu gibt es das Programm depmod. Es findet alle Abhängigkeiten und speichert sie in der Datei:
/lib/modules/`uname -r`/modules.dep.bin
Eine lesbare Version wird abgelegt in:
/lib/modules/`uname -r`/modules.dep

Terminal
```
root@debian964:~# grep sr_mod /lib/modules/`uname -r`/modules.dep
kernel/drivers/scsi/sr_mod.ko: kernel/drivers/cdrom/cdrom.ko
```

Etwas mehr Information über einen Treiber bekommt man mit modinfo:

Terminal
```
root@debian964:~# modinfo sr_mod    # Info zum Treiber sr_mod
filename:   /lib/modules/4.9.0-5-686-pae/kernel/drivers/scsi/sr_mod.ko
license:    GPL
alias:      scsi:t-0x04*
alias:      scsi:t-0x05*
alias:      block-major-11-*
license:    GPL
description:SCSI cdrom (sr) driver
depends:    scsi_mod,cdrom
intree:     Y
vermagic:   4.9.0-5-686-pae SMP mod_unload modversions 686
parm:       xa_test:int
```