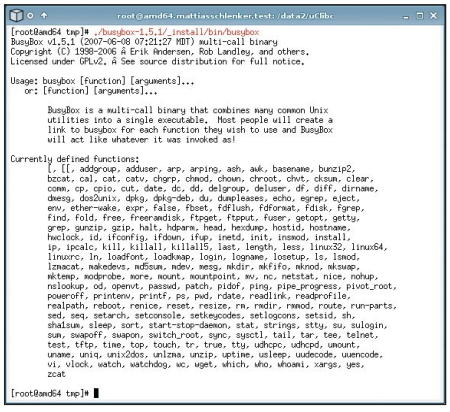
MINI-LINUX AUF 4 MBYTE SELBST GEBAUT

Dich krieg ich klein!

Dutzende Systembibliotheken, ein aufwändiges Init und viele bereits beim Start notwendige Tools sorgen dafür, dass selbst schlanke Distributionen kaum in den zweistelligen MByte-Bereich abzuspecken sind. Radikal weniger geht mit der BusyBox, die z. B. auch in der FRITZ!Box Verwendung findet: Ein einziges Binary enthält alle Linux-Standardtools. VON **MATTIAS SCHLENKER**

Schon die ersten Unix-Systeme bewegten sich im zweistelligen MByte-Bereich: Dicke monolithische Kernel, umfangreiche Bibliotheken und nicht zuletzt die enorme Werkzeugsammlung machen ein normales Unix oder Linux schon ohne grafische Oberfläche zu einem System, das keinen Platz mehr auf Disketten oder in einem kleinen Flash-Speicher findet. Doch in den letzten Jahren wurde Linux immer beliebter beim Einsatz auf Routern, Appliances, Mobiltelefonen oder PDAs. Eines der bekanntesten Beispiele ist auch AVMs FRITZ!Box. Zwei Komponenten aus der Embedded-Entwicklung halten dieses Linux kompakt: Die Erste ist die BusyBox, ein Multicall-Binary, also eine einzige Binärdatei, auf die viele Softlinks zeigen. Je nachdem, mit welchem Namen man die BusyBox aufruft, verhält sie sich wie das entsprechende Werkzeug. Entwickelt wurde das sie als Hilfsmittel für den Start von Linux-Installationen von Floppy-Disks. Zweite Komponente ist die kompakte C-Bibliothek *uClibc*. Sie wurde im Umfeld des uClinux-Projektes für Microcontroller entwickelt, emanzipierte sich und harmoniert auch mit gewöhnlichen Kerneln. Die uClibc ist kompakt und benötigt wenig Arbeitsspeicher zur Laufzeit. Zudem lassen sich mit der uClibc auch für die Nutzerauthentifizierung notwendige Programme statisch linken – die gewöhnliche glibc bietet diese Möglichkeit nicht. Nun ist für die Erstellung uClibc-basierter Anwendungen auf einem glibc-basierten Linux eine komplette „Crosscompile-Toolchain“ notwendig, bestehend aus Compiler, Linker und einigen weiteren Tools. Diese einzurichten, ist jedoch recht aufwändig, weshalb wir zu einem kleinen Trick gegriffen haben: Ein komplettes, uClibc-basiertes Linux dient als Chroot-Umgebung, in der BusyBox und Co. kompiliert werden. Das Mini-Linux finden Sie als Tarball auf der Heft-DVD. Neuere Versionen können Sie mit den Buildroot-Skripten von **www.uclibc.org** selbst bauen.



Alles drin: Die BusyBox vereint in einem Binary mit ca. 500 KByte viele Linux-Standardtools – die ideale Basis für ein Mini-Linux.

Nach dem Entpacken der Chroot-Umgebung müssen Sie einige spezielle Dateisysteme in die neue Umgebung mounten:

```
mount -o bind /dev uclibc-chroot/dev/
mount -t proc none uclibc-chroot/proc/
```

Der Autor verwendet in der Regel auch ein gemeinsames /tmp:

```
mount -o bind /tmp uclibc-chroot/tmp/
```

Jetzt können Sie mit dem Befehl *chroot* in das uClibc-Linux wechseln, wo Ihnen alle Tools zur Erstellung der BusyBox bereitstehen:

```
LC_ALL=C chroot uclibc-chroot/
```

Wichtigste Komponente: BusyBox

Auch die zum Zeitpunkt der Entstehung dieses Artikels aktuellste BusyBox (Version 1.5.1) finden Sie als Quellcode-Paket auf der Heft-DVD. Entpacken Sie diese in */tmp* und wechseln Sie in das entstandene Verzeichnis, wo Sie das Menü-basierte Konfigurations-Frontend aufrufen:

```
make menuconfig
```

Denken Sie bei der Auswahl der Tools daran, dass einzelne zusätzliche Werkzeuge oder die Auswahl luxuriöser Aufrufparameter oft nur einige Kilobyte zusätzlichen Speicher benötigen, aber die Arbeit mit dem Mini-Linux später deutlich vereinfachen können. Keinesfalls vergessen sollten Sie *init*, es wird später für den Systemstart benötigt, und die Shells *ash* oder *lush*.

In der Vergangenheit schlug bei einigen Versionen der BusyBox das *make menuconfig* in einer uClibc-Chroot-Umgebung fehl. In diesem Fall muss die Konfiguration im Host-System vorgenommen werden, wobei penibel darauf zu achten ist, vorher und nachher aufzuräumen, damit keine gegen Glibc kompilierten Binärdateien zurückbleiben:

```
make clean
make menuconfig
make clean
```

Alternativ können Sie natürlich auch die auf DVD beiliegende Konfigurationsdatei verwenden, welche Sie nach *.config* im BusyBox-Quellcode-Verzeichnis kopieren. Gebaut und installiert wird mit bekannten Befehlsfolgen:

```
make
make install
```

Die komplette BusyBox finden Sie nun im Unterverzeichnis *_install*: Einziges Binary dort ist *bin/busybox*, alle weiteren Programme sind Softlinks.
Zum Testen der BusyBox und der später fürs Mini-Linux vorgesehenen Programme sowie zur Ermittlung der später benötigten Größe der Initrd empfehlen wir, unter */tmp* ein Verzeichnis anzulegen, in das alle Dateien mit *rsync* kopiert werden:

```
mkdir /tmp/miniroot
rsync -avHP _install/ /tmp/miniroot/
```

Für ein komplettes Linux fehlen in diesem Verzeichnis noch einige Ordner:

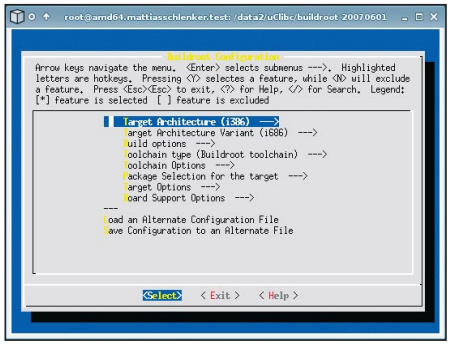
```
mkdir /tmp/miniroot/{lib,proc,var,dev,
➥etc,tmp,root,sys}
```

Noch lassen sich die BusyBox-Tools nicht verwenden: Es fehlen notwendige Bibliotheken. Welche das sind, ermittelt das Werkzeug *ldd* im Chroot-Käfig – sein Name bedeutet *list dynamic dependencies*:

```
ldd _install/bin/busybox
```

Die so ermittelten Dateien und die auf diese verweisenden Softlinks müssen ebenfalls in das Verzeichnis */tmp/miniroot*. Bei einer guten Hand voll Dateien ist der Vorgang noch überschaubar:

```
rsync -avHP /lib/libcrypt*so* /tmp/
➥miniroot/lib/
rsync -avHP /lib/libm.so* /tmp/
➥miniroot/lib/
rsync -avHP /lib/libm-*.so* /tmp/
➥miniroot/lib/
rsync -avHP /lib/libc.so* /tmp/
➥miniroot/lib/
rsync -avHP /lib/ld-uClibc*so* /tmp/
➥miniroot/lib/
rsync -avHP /lib/libuClibc*.so /tmp/
➥miniroot/lib/
```



Mit den *Buildroot-Skripten* können Sie eine aktuelle Chroot-Umgebung zum Bauen von BusyBox und Tools erstellen (auch auf Heft-DVD).

Bruder im Geiste: Dropbear

Einen ähnlichen Ansatz wie BusyBox verfolgt Dropbear, ein kompakter SSH-Client, Server und Schlüsselwerkzeug, wahlweise als separate Binaries oder in einem Binary auf wenigen hundert Kilobyte.
Dessen Konfiguration erfolgt zunächst durch das gewohnte

```
./configure --prefix=/usr/
```

Anschließend steht aber ein Eingriff in der Datei *options.h* an, wo der Wert

```
#define DROPBEAR_RANDOM_DEV „/dev/
➥urandom“
```

gesetzt werden muss. So ist sichergestellt, dass auch auf (Firewall-) Systemen mit wenig Entropie die Nutzung des sicheren SSH möglich ist.
Beim Bauen und Installieren müssen die gewünschten Binaries explizit angegeben werden:

```
make PROGRAMS=“dropbear dbclient
➥dropbearkey dropbearconvert scp“
make PROGRAMS=“dropbear dbclient
➥dropbearkey dropbearconvert scp“
➥install
```

Auch die Dropbear-Binaries werden über den Befehl *rsync* in das Verzeichnis */tmp/miniroot* kopiert:

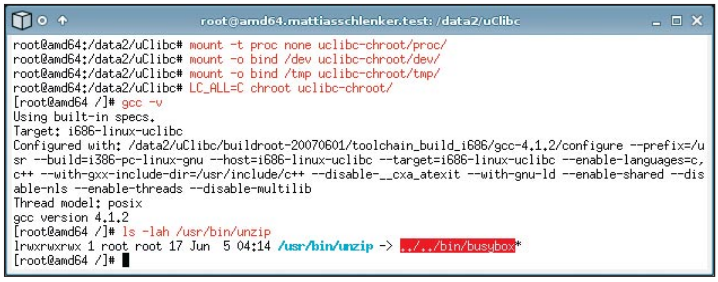
```
rsync -avHP /usr/sbin/dropbear \
  /tmp/miniroot/usr/sbin
for i in dbclient dropbearkey \
  dropbearconvert scp
do
    rsync -avHP /usr/bin/$i \
    /tmp/miniroot/usr/bin/
done
```

Da Dropbear noch Debug-Symbole enthält, sollten Sie alle Binaries noch strippen:

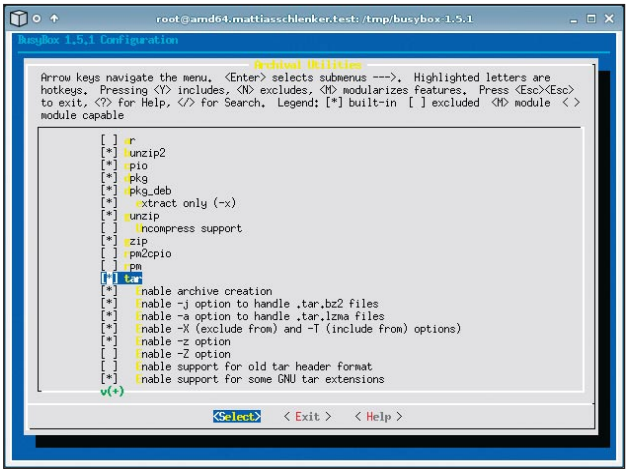
```
strip /tmp/miniroot/usr/sbin/dropbear
for i in dbclient dropbearkey \
```

Der Einkaufszettel

Unser Mini-Linux entsteht in zwei Stufen. Zunächst wird mit dem Buildroot-Script von www.uclibc.org ein uClibc-basiertes Root-Dateisystem mit kompletter Entwicklungsumgebung erstellt:
cd buildroot
make menuconfig
make
Das *rootfs.i686.ext2* genannte Image kann anschließend gemountet und sein Inhalt mit rsync -avHP mountpoint/ ziel/ in ein separates Verzeichnis kopiert werden. Per *chroot* entstehen dort BusyBox und die weiteren Komponenten des Mini-Systems. Damit das Buildroot-System eingesetzt werden kann, ist eine komplette Toolchain nebst *ncurses*-Entwicklerbibliotheken notwendig. Sie finden das Buildroot-Paket inclusive unserer Konfiguration und allen notwendigen Quellcodes auf Heft-DVD. Um in der gleichen Umgebung wie wir arbeiten zu können, haben wir eine fertige Chroot-Umgebung *uclibc-chroot-20070601.tar.bz2* beigelegt. Eine aktuelle Version des Buildroot-Skriptes finden Sie unter http://buildroot.uclibc.org
In der Chroot-Umgebung werden BusyBox 1.5.1 und Dropbear 0.49 gebaut. Wir haben deren Quellcodes und Konfiguration auf Heft-DVD beigelegt. Zudem finden Sie den Tarball einer fertig gebauten BusyBox auf der DVD. Aktuelle Versionen von BusyBox und Dropbear können Sie unter www.busybox.net und http://matt.ucc.asn.au/dropbear herunterladen.
Für den Bau eines eigenen Kernels haben wir den Vanilla-Kernel 2.6.21.5 von http://www.kernel.org beigelegt. Bauen Sie diesen nicht in der Chroot-Umgebung, sondern im Hostsystem. Sollten Sie an dieser Stelle abkürzen wollen, finden Sie auf der DVD einen fertig gebauten Kernel, der die gängigsten Treiber für Dateisysteme, IDE- und SATA-Chipsätze sowie viele Netzwerkchips statisch enthält.
Um das eigene Mini-Linux von USB-Stick, -Festplatte oder CD zu starten, benötigen Sie einen Bootloader. Sowohl *Syslinux* (für FAT16-formatierte USB-Sticks) als auch *Isolinux* (für CDs) sind im Syslinux-Paket 3.51 vorhanden. Aktuelle Versionen finden Sie unter http://syslinux.zytor.com.
Etwas aufwändigere Startup-Skripte, die auch Bootvariablen („Cheatcodes“) auslesen, sowie regelmäßig aktualisierte Tipps zum Bauen mit neueren uClibc-Buildroot-Systemen finden Sie auf der Webseite des Autors unter http://blog.rootserverexperiment.de/category/mini-linux.
Auf der DVD haben wir eine Textdatei untergebracht, welche die Befehle aus dem Fließtext enthält. Sie erspart lästiges Abtippen.



Mittels *chroot* wechseln Sie in das *uClibc-Buildsystem*. Hier steht ein für die kompakte C-Bibliothek *uClibc* gebauter Compiler zur Verfügung.



Mit *make menuconfig* wählen Sie die benötigten Applets der BusyBox aus. Unsere *.config* finden Sie auf der Heft-DVD.

```
  dropbearconvert scp
do
    strip /tmp/miniroot/usr/bin/$i
done
```

Wie bei der BusyBox müssen bei Dropbear über den Befehl *ldd* die benötigten Bibliotheken ermittelt und anschließend kopiert werden:

```
rsync -avHP /lib/libutil* /tmp/
➥miniroot/lib/
rsync -avHP /lib/libcrypt* /tmp/
➥miniroot/lib/
rsync -avHP /usr/lib/libz*so* /tmp/
➥miniroot/usr/lib/
rsync -avHP /lib/libgcc* /tmp/miniroot/
➥lib/
```

Ein erster Test

In */tmp/miniroot* finden Sie jetzt tatsächlich ein komplettes Linux-System: Der Befehl *du* offenbart etwa 500 KByte für Bibliotheken und jeweils weitere 500 KByte für das BusyBox-Binary mit Softlinks sowie die SSH-Tools. Ein Test dieses Mini-Linux ist wieder mit dem Chroot-Aufruf möglich, der nach dem Mounten einiger Dateisysteme von außerhalb des bereits vorhandenen Chroot-Käfiges erfolgen muss. Die Angabe einer Shell beim Chrooten ist diesmal notwendig, weil das BusyBox-Minisystem keine Bash enthält, sondern nur die etwas einfachere *ash*. Ist die Bash per Umgebungsvariable als Standardshell definiert, schlägt sonst der Chroot fehl:

```
mount -o bind /dev /tmp/miniroot/dev/
mount -t proc none /tmp/miniroot/proc/
LC_ALL=C chroot /tmp/miniroot /bin/sh
```

Die BusyBox-Shell *ash* begrüßt Sie wie nach dem Log-in an einem normalen System und rein subjektiv ist beim Rundgang in der Shell kein großer Unterschied zu einem richtigen Linux spürbar. Sogar die Befehls-History funktioniert. Und mit dem SSH-Client *dbclient* können Sie Verbindung zu entfernten Rechnern (oder dem Hostsystem) aufnehmen.

Natürlich ist das Linux-System noch nicht komplett. Selbst wenn wir den Kernel außen vor lassen, benötigen wir noch das Init-System, welches beim Systemstart einige Dienste startet und beispielsweise das Netzwerk initialisiert. Üblicherweise kommt in der *Initrd* – welche ja unser gesamtes Minisystem enthält – ein Script *linuxrc* zum Einsatz, das benötigte Treiber lädt und das spätere Root-Dateisystem mounten sowie anschließend mit dem Tool *pivot_root* die Initrd von der Verzeichniswurzel entfernt. All das ist

Troubleshooting

In der uClibc-Chroot-Umgebung läuft nicht immer alles glatt. Gelegentlich fehlen Programme oder von der (auch dort vorhandenen) BusyBox bereitgestellte Befehle arbeitet anders als von einigen *./configure*-Skripten erwartet. Fehlt ein Befehl wie *grep*, sollten Sie zunächst mit dem Befehl

```
busybox
```

ermitteln, ob die BusyBox den gewünschten Befehl mitbringt und falls ja anschließend einen Softlink setzen:

```
ln -s /bin/busybox /bin/grep
```

Etwas mehr Aufwand ist notwendig, wenn ein Befehl zwar im uClibc-Entwicklungssystem vorhanden ist, aber nicht wie von einigen Makefiles erwartet funktioniert. So geschehen ausgerechnet beim Befehl *ar*, der beim Linken von Programmen notwendig ist. Wir sahen uns deshalb gezwungen, in der Hostumgebung die *binutils* neu zu bauen und dabei statisch zu linken:

```
tar xvjf binutils-2.17.tar.bz2
cd binutils-2.17
./configure --prefix=/usr --disable-
nls
make configure-host
make LDFLAGS="-all-static"
```

Anschließend löschten wir den Softlink */usr/bin/ar* auf der BusyBox und kopierten das statische *ar* aus dem Ordner *binutils-2.17/binutils/* nach */usr/bin*. Die Kompilation weiterer Pakete klappte danach auf Anhieb. Sollten Sie weitere Befehle im Chroot-System durch statisch gelinkte Binaries ersetzen wollen, reicht in den meisten Fällen als einzige Abweichung vom „Dreisatz"

```
make LDFLAGS="-static"
```

oder

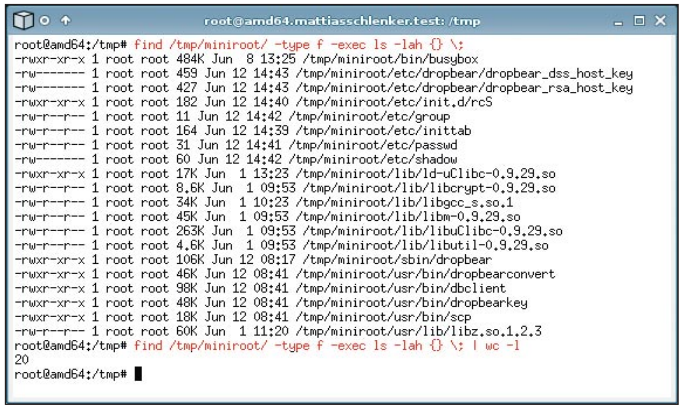
```
make LDFLAGS="-all-static"
```

Nicht immer werden beim *make install* der BusyBox alle Softlinks sauber gesetzt. Sie können Softlinks zu allen in der BusyBox enthaltenen Tools setzen, wenn Sie nach dem Kopieren der BusyBox (und der von ihr benötigten Bibliotheken), aber vor dem Kopieren weiterer Anwendungen, nach */tmp/miniroot* chrooten und dort

```
busybox --install
```

aufrufen. Das generiert möglichwerweise fehlende Softlinks neu.

Das Tool *ldd* listet Bibliotheksabhängigkeiten auf. Die so identifizierten Libraries müssen ebenfalls in das Initrd-Image.



***find* zeigt, dass sich in der Initrd lediglich 20 reguläre Dateien befinden, daneben aber auch noch Dutzende Softlinks und Gerätedateien.**

nicht notwendig, da die *Initrd* das Wurzeldateisystem bleibt – vom Boot bis zum Shutdown. So kann ein gewöhnliches *init* zum Einsatz kommen, das die BusyBox in einer abgespeckten Version mitbringt. Der Softlink *linuxrc* ist dennoch notwendig, da bei einem von Initrd gestarteten System dieses Programm aufgerufen wird. Wie das Original nutzt die BusyBox eine */etc/inittab*, die jedoch etwas anders aufgebaut ist: Statt Runlevels steht im ersten Feld das ausführende Terminal, das zweite Feld kennzeichnet aufgerufene RC-Dateien (*sysinit*), Programme, die nach Terminierung neu gestartet werden (*respawn*) oder die Reaktion auf den „Affengriff“ (*ctrlaltdel*):

```
::sysinit:/etc/init.d/rcS
tty1::respawn:/sbin/getty 38400 tty1
tty2::respawn:/sbin/getty 38400 tty2
tty3::respawn:/sbin/getty 38400 tty3
::ctrlaltdel:/sbin/reboot
```

Unser Beispiel führt das Script */etc/init.d/rcS* aus, öffnet ein Log-in auf drei Konsolen und reagiert auf den „Affengriff“ mit dem typischen Reboot. Das Script */etc/init.d/rcS* kann dazu verwendet werde, ein komplettes System-V oder BSD-Init mit eigenen Start-Stop-Skripten nachzubauen – wir haben es bei einer simplen Initialisierung der Netzwerkschnittstellen, gefolgt vom Start eines SSH-Servers, belassen:

```
#!/bin/sh
/bin/mount -t proc none /proc

/bin/mount -t devpts none /dev/pts
/sbin/ifconfig lo inet 127.0.0.1
/sbin/ifconfig eth0 inet 192.168.1.82
/bin/sleep 3
/usr/sbin/dropbear -E
```

Statt der statischen Konfiguration des Netzwerkinterfaces können Sie auch eine dynamische Konfiguration mit dem in der BusyBox enthaltenen DHCP-Client vornehmen:

```
/sbin/udhcpc eth0
```

In diesem Fall müssen Sie jedoch das DHCP-Clientscript */usr/share/udhcpc/default.script* vom uClibc-Buildsystem an dieselbe Stelle im Mini-Linux kopieren.

/etc vollständig besiedeln

Noch enthält das Verzeichnis */etc*, welches die systemweiten Einstellungen beherbergt, nicht die notwendigen Dateien, um ein späteres Log-in zu ermöglichen. Es muss minimal eine */etc/passwd* vorhanden sein, die wie bei einem normalen Linux aufgebaut ist. Unsere sieht so aus:

```
root:x:0:0:root:/root:/bin/sh
```

Die dazugehörige, nur für *root* lesbare */etc/shadow*:

```
root:pwhash:10933:0:99999:7:::
```

Statt *pwhash* ist der mit *openssl passwd -1* erzeugte MD5-Hash einzutragen. Die */etc/group* enthält den einzigen Nutzer:

```
root:x:0:
```

Dropbear benötigt seine beiden Schlüssel unter */etc/dropbear/dropbear_dss_host_key* und */etc/dropbear/dropbear_rsa_host_key*. Am einfachsten ist es, diese im Build-Chroot mit dem Befehl

```
dropbearkey -t dss -f /etc/dropbear/
➥dropbear_dss_host_key
```

zu erzeugen (den RSA-Schlüssel natürlich mit

Wie boote ich mein Mini-Linux

Der einfachste Weg zu einem bootfähigen Medium mit dem eigenen Mini-Linux führt über ein ISO-Image. Erstellen Sie zunächst einen Ordner *build*, der später den Inhalt der CD aufnimmt. Dieser Ordner bekommt ein Bootverzeichnis:

```
mkdir build
mkdir -p build/boot/isolinux
```

Kopieren Sie den Kernel (hier *vmlinuz*), die Initrd (hier *initrd.gz*) des Mini-Linux und die Datei *isolinux.bin* aus dem syslinux-Paket in den Ordner *build/boot/isolinux*. Erstellen Sie dann eine Datei *build/boot/isolinux/isolinux.cfg* mit dem folgendem Inhalt. Das Label darf nur acht Zeichen enthalten, bei den Dateinamen gelten nur die Einschränkungen des ISO-Dateisystems ohne Erweiterungen:

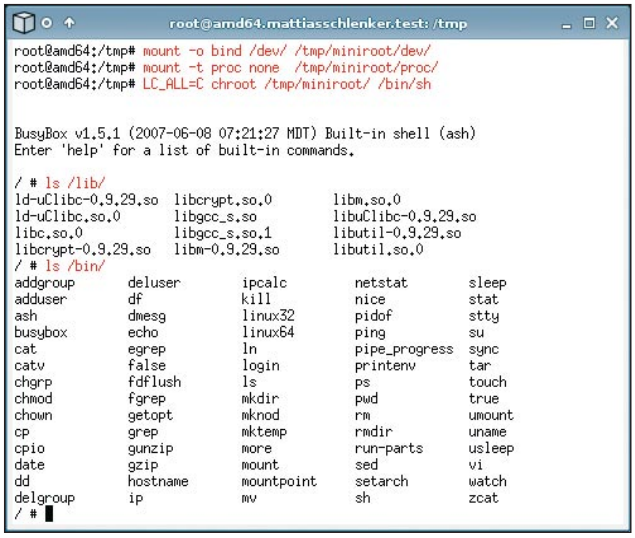
```
DEFAULT minilinx
PROMPT 1
TIMEOUT 100

LABEL minilinx
KERNEL vmlinuz
APPEND initrd=initrd.gz rw vga=791
```

Jetzt kann ein ISO-Image erzeugt werden:

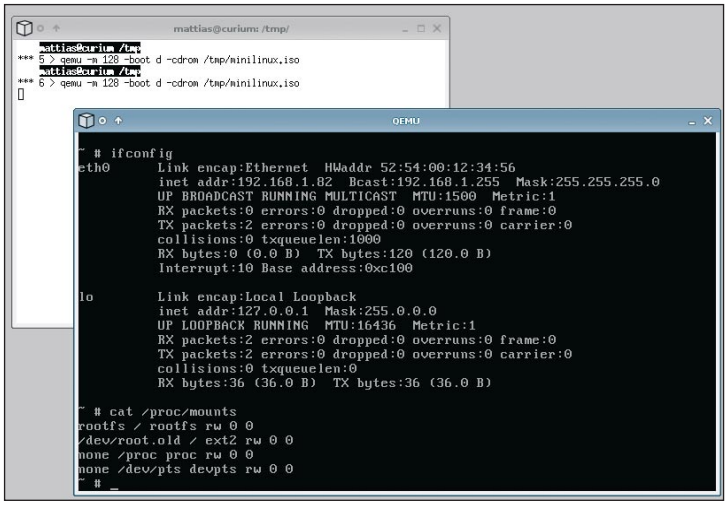
```
mkisofs -r -J -pad \
-b boot/isolinux/isolinux.bin \
-c boot/isolinux/boot.cat \
-no-emul-boot -boot-info-table \
-boot-load-size 4 \
-o minilinux.iso build
```

Das erzeugte ISO-Image können Sie in einer Virtualisierungssoftware wie *VMware* oder *Qemu* direkt starten. Bei *Qemu* gelingt dies ohne Konfigurationsdatei:

```
qemu -boot d -cdrom minilinux.iso
```



Auch der Test der Initrd-Zusammenstellung erfolgt mit *chroot*. Testen Sie den SSH-Client *dbclient* und einige BusyBox-Applets, um sicherzugehen, dass alle benötigten Bibliotheken da sind.



Für den Test des Mini-Linuxes inclusive selbstgebautem Kernel ist es am sinnvollsten, ein ISO-Image zu erstellen und dieses in einer virtuellen Maschine (hier *qemu*) zu booten.

-t rsa) und dann ins Verzeichnis */tmp/minilinux* zu kopieren. Alternativ können Sie auf fixe Schlüssel verzichten und die Zeilen zur Schlüsselerzeugung in das Startupscript */etc/init.d/rcS* einbauen.

Fehlt uns noch */dev*, das Verzeichnis mit den Gerätedateien. Im Prinzip kann mit *rsync -avHP* der komplette Inhalt dieses Verzeichnisses aus dem *uClibc-Chroot* kopiert werden. Vergessen Sie in diesem Fall bitte nicht, vorher ein mit der Option *bind* drübergemountetes */dev* auszuhängen.

Die Initrd besiedeln

Bei der Initrd handelt es sich um ein – im Idealfall 4096 Kilobyte großes – Partitionsimage, das mit einem *Ext2*-Dateisystem versehen ist. Sie erzeugen es zunächst mit *dd*:

```
dd if=/dev/zero of=/tmp/initrd.img
➥bs=1024 count=4096
```

Anschließend wird es formatiert:

```
mkfs.ext2 /tmp/initrd.img
```

Danach kann es gemountet werden:

```
mkdir /tmp/initrd
mount -o loop /tmp/initrd.img /tmp/initrd
```

Auf das Image kopieren Sie jetzt alle Inhalte des zuvor befüllten Ordners */tmp/minilinux*. Vergessen Sie nicht, dort gemountete Dateisysteme auszuhängen, und prüfen Sie noch einmal, ob die zuvor kopierten Gerätedateien in */dev* vorhanden sind:

```
rsync -avHP /tmp/miniroot/ /tmp/initrd/
```

Die Ramdisk wird anschließend wieder ungemountet und mit *gzip* komprimiert:

```
umount /tmp/initrd/
gzip -c /tmp/initrd.img > /tmp/
➥initrd.gz
```

Die Datei */tmp/initrd.gz* kann nun jedem Kernel als Initrd übergeben werden, der einen Treiber für das Ext2-Dateisystem enthält und der Unterstützung für initiale Ramdisks bietet.

Der erste Bootvorgang

Um das Mini-Linux zu starten, haben wir auf dem Bausystem zunächst einen Vanilla-Kernel gebaut, der alle wichtigen Dateisystemtreiber sowie Treiber für viele Netzwerkkarten und ATA-/SATA-Chipsätze statisch enthielt. Das ersparte uns, auf die Initrd mit dem Mini-Linux auch noch Treiber packen zu müssen.

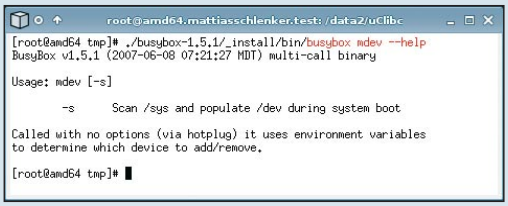
Diesen Kernel trugen wir in die lokale GRUB-Konfiguration ein und: Bingo! – bei der Auswahl des Mini-Linux fuhr unser wenige Megabyte großes selbst gebautes Linux problemlos hoch. Der Eintrag in der */boot/grub/menu.lst*

```
title BusyBox Minisystem
root (hd0,0)
kernel /boot/vmlinuz-2.6.21.5 rw splash
initrd /boot/busysbox-miniroot.gz
boot
```

Natürlich ist der Start über den Bootloader des Hostsystems die denkbar unflexibelste Methode, und mit Kernel und Ramdisk auf der regulären Systempartition ist auch der Einsatz als Backup und Recovery-Tool nicht sinnvoll. Wie Sie das eigene Mini-Linux auf CD oder USB-Stick bannen, erklären wir deshalb in einem separaten Kasten. **jkn**

Devices mit *mdev* erstellen

Beim eigenen Mini-Linux ist in den meisten Fällen der Einsatz von *udev* zur dynamischen Erzeugung von Gerätedateien der Overkill. Eine einfache, kompakte Lösung bringt BusyBox in Form von *mdev* mit. Ist das spezielle Dateisystem */sys* gemountet, kann *mdev* beim Systemstart aufgerufen werden, um alle benötigten Gerätedateien zu erzeugen. Ergänzen Sie dafür Ihr Startscript *rcS* um die folgenden Zeilen:



Das BusyBox-Applet *mdev* ist eine kompakte Alternative zum Einsatz von *udev*.

```
/bin/mount -t sysfs none /sys
/sbin/mdev -s
```