

Linux auf einem USB Stick

Oliver Kaiser

tok@uni-math.gwdg.de

Source Talk Kongress 2005

Mathematisches Institut Universität Göttingen

4. März 2005



Überblick

- 1 Booten von USB
 - Voraussetzungen
 - Kernel-support
 - Bootloader
 - initrd
- 2 Erweiterungen
 - read-only root Dateisystem
 - verschlüsseltes Dateisystem
- 3 *LFS* & Management einer Distribution
 - Linux From Scratch, Gentoo & Gnu
 - uclibc, uwoody & emdebian

Die Fähigkeit des BIOS eine USB Festplatte als Bootlaufwerk zu verwenden ist in jedem Fall unumgänglich; nur dadurch können der Bootloader bzw. der Kernel geladen werden, ohne dass die dafür eigentlich notwendigen Treiber geladen sind.

Bis vor einigen Monaten war dies nicht generell möglich; auf modernen Rechnern sollte es inzwischen funktionieren.

einige Hinweise

- USB Tastatur Unterstützung aktivieren (Epi-a-M, Shuttle mv42n)
- „alte“ Schnittstellen deaktivieren
- IDE Controller vollständig deaktivieren

Teil 1

- CONFIG_BLK_DEV_LOOP
Ermöglicht es eine normale Datei als *Blockdevice* (also z.B. als Partition) anzusprechen.
- CONFIG_BLK_DEV_RAM
Erzeugt virtuelle *Blockdevices* in einem Teil des Hauptspeichers; sie können über `/dev/ram*` angesprochen und sollten ebenfalls mit `dd` initialisiert und danach mit einem Dateisystem gefüllt werden.
- CONFIG_BLK_DEV_INITRD
Eine spezielle Ramdisk, in die vom *Bootloader* ein vorbereitetes Image geladen wird und die als *root* Dateisystem verwendet wird.
- CONFIG_USB
Allgemeine USB Unterstützung.

Teil 2

- `CONFIG_USB_UHCI_ALT` bzw. `CONFIG_USB_OHCI`
Es sollte nur einer dieser USB Controller Treiber vorhanden sein, diese Wahl hängt vom Chipsatz des Mainboard ab.
- `CONFIG_USB_DEVICEFS`
Diese Option ist nicht notwendig, erzeugt aber eine Abbildung der vorhanden USB Gerte im `proc` Dateisystem. Dies könnte bei der Unterscheidung mehrerer Laufwerke helfen.
- `CONFIG_USB_STORAGE`
Treiberunterstützung fr USB Massenspeichergeräte.
- `CONFIG_SCSI`
Allgemeine SCSI Unterstützung.
- `CONFIG_BLK_DEV_SD`
Der USB Stick wird intern als SCSI Festplatte behandelt; daher ist diese Option erforderlich.

In diesem Fall wird Lilo verwendet; je nach Geschmack wären auch *grub* oder eventuell *syslinux* möglich.

Zuerst wird *Stage 1* aus dem MBR des Bootlaufwerks gelesen; dieser wenige Byte grosse Teil verweist auf *Stage 2*. Erscheint beim Booten nur ein „L“ gefolgt von einem zweistelligen Fehlercode (der sich auch oft wiederholen kann) konnte der zweite Teil (meistens `boot*.b`) nicht geladen werden.

Eine häufige Ursache sind unterschiedliche Geometriewerte der Festplatte beim Schreiben bzw. Booten; `lilo -t -v5 | $PAGER` zeigt die beim Erstellen des Bootsektors verwendeten Werte an, `lilo -Tgeom` simuliert(!) die Sicht des BIOS. Manchmal ist es notwendig die Größen der Partitionen (insbesondere derjenigen, die `/boot` enthält) so anzulegen, daß sie auf Zylindergrenzen beginnen (und enden).

Lilo Konfiguration

```
disk=/dev/sda
  bios=0x80
boot=/dev/sda
map=/boot/map
install=/boot/boot-text.b
default=something
read-only
lba32
#vga=0x314
append="ramdisk_size=4096 quiet root=/dev/ram0 \
  init=/linuxrc ro"
image=/boot/vmlinuz
  label=something
  initrd=/boot/initrd.img.gz
```

Enthält:

- einige *device nodes* im `/dev` Verzeichnis
- ein ausführbares Programm (Kernel-parameter: `init=`)
- ein minimales root Dateisystem (optional)

Wird zum Beispiel auf Installationsmedien verwendet, um Kernelmodule die zum Ansprechen des eigentlichen root Dateisystems notwendig sind zu laden.

In diesem Fall muss lediglich einige Sekunden gewartet werden (sleep); das Problem scheint eine technisch bedingte Verzögerung beim Erkennen der USB Geräte und der anschließenden Intialisierung zu sein; das software-seitige Laden des Kernels endet bevor auf `/dev/sda*` zugegriffen werden kann, daher scheitert das *Mounten* der root Partition, was letztlich zur Kernel-Panik führt.

Erstellen einer virtuellen Partition

```
dd if=/dev/zero of=file.img bs=1M count=1  
mke2fs -F -m0 file.img  
# mke2fs -F -m0 -b 1024 file.img  
mkdir test  
mount -o loop file.img test
```

Besonderheiten

- Flag: `-b 1024` für `initrd` Images
- `initrd` Images können mit `gzip -9 file.img` komprimiert werden.
- Man sollte keine `ext3/reiserfs` verwenden

Ein *linuxrc* Skript

```
sleep 3
mount -n -t proc proc /proc
# /new_root MUST exist in the initrd image
/bin/mount -n -o ro -t ext2 /dev/sda3 /new_root
cd /new_root
# /new_root/initrd must EXIST
pivot_root . initrd
cd /
umount -n /initrd/proc 2>/dev/null
mount -n -t proc proc /proc
# do something else?
# maybe unmount /proc again
exec chroot . /sbin/init 3 <dev/console >dev/console 2>&1
```

Problem: offene Filedeskriptoren

Der Vorteil von *pivot_root* gegenüber *chroot* ist, daß die Ramdisk nach dem Wechsel des root Dateisystems unmounten kann. Dies ist durch Fehler im Kernel manchmal nicht möglich (z.B. 2.4.22), da gestartete Kerneldienste offene Filedeskriptoren enthalten, die auf Device-Nodes im „alten“ /dev Verzeichnis verweisen.

Ein Lösungsmöglichkeit besteht aus dem „Verschieben“ eines gemounteten Dateisystems (siehe: `man 2 mount` und `man mount`).

Motivation

- Dateisystem ist unempfindlich gegenüber Abstürzen und Benutzerfehlern
- dauerhafte Manipulation ist nur bei physikalischem Zugriff möglich (Schutz vor Angreifern)

Da viele USB Sticks mit einem mechanischen Schreibschutz-schalter ausgestattet sind, kann dieser Mechanismus nicht per Software überwunden werden.

Viele Programme benötigen jedoch schreibenden Zugriff auf diverse Dateien und Verzeichnisse. Eine einfache Lösung ist das Erstellen einer (oder mehrerer) Ramdisk(s), die beim Booten mit Daten gefüllt werden (meist mit *dd* oder *tar*). Mit *symbolischen links* kann man die schreibbaren Pfade in den „normalen“ Baum einbinden.

- offensichtlich: `/var` & `/tmp`
- `/dev` **Verzeichnis** muß schreibbar sein (*syslogd*)
- einige Dateien in `/etc` (oder komplett)
GNU mount verwendet `/etc/mtab` (*busybox* nicht); weiterhin
`/etc/ppp`

Alternativ könnte man *devfs* oder im 2.6er Kernel die Neuentwicklungen *udev* und *tmpfs* verwenden.

Allerdings wird der Verzeichnisbaum an einigen Stellen immer Verweise auf die im Hauptspeicher simulierten Partitionen enthalten; dies läßt sich nur durch weitere *patches* ändern:

- translucency overlay file system
- union fs

- cryptoloop Verwendet die CryptoAPI; wird in 2.6 offiziell durch dm-crypt abgelöst (funktioniert aber; „alte“ Partitionen sind kompatibel); für „späte“ 2.4er Kernel nur noch ein patch, im 2.6er enthalten.
- loop-aes <http://loop-aes.sourceforge.net/> Größerer Aufwand (mehr Veränderungen am Kernel, *gpg* patchen); angeblich optimierte Implementation der Chiffren; Hinweise zum verschlüsselten *swap* Dateisystem.
- dm-crypt (Kernel 2.6)
<http://www.saout.de/misc/dm-crypt/>

Beispiel: cryptoloop

```
modprobe cryptoloop
modprobe aes
losetup -e aes /dev/loop0 /dev/hda5
mke2fs /dev/loop0
e2fsck /dev/loop0
mount /dev/loop0 /mnt/somewhere
```

Dieser Vorgang läßt sich nur bedingt in einem Skript verwenden, da *losetup* das Passwort aus Sicherheitsgründen direkt aus der Tastatur ausliest. Einige gepatchte Variante machen dies mit den Parametern „-p“ und „-pass-fd“ rückgängig; man **sollte** sich auf jeden Fall über die Risiken informieren.

Warum ?

- alternative Speichermedien bieten erheblich weniger Platz als Festplatten
- hardware-optimierte Kompilation kann in einigen Fällen meßbar die Leistung verbessern
- die USB 1 Datenübertragungsrate bremst das System bei I/O Zugriffen; geringere Größe der Programme und Bibliotheken verringern die Ladezeiten
- spezielle Lösungen erfordern in der Regeln erheblich weniger Software als ein Desktop-System (Vorteil: geringere Komplexität)
- eine „ungewöhnliche“ Konfiguration bietet Schutz vor einigen Angriffen (bekannte Sprungadressen zu Funktionen)

LFS (<http://www.linuxfromscratch.org/>): Erzeugen eines „persönlichen“ Linuxsystem aus den verfügbaren Quellen.

- man ein schon laufendes System braucht
- ein vollständiges System zu erzeugen dauert recht lange dauert
- keine (zentrale) Verwaltung der Abhängigkeiten der verschiedenen Komponenten

Die Gentoo Distribution wird meist in Zusammenhang mit speziellen *compiler* Optimierungs-Schaltern und besonders aktueller Software genannt. Das „Paket-management“ Programm *emerge* kann (entsprechende Meta-Dateien vorausgesetzt) Software herunterladen und erzeugen (eventuelle Abhängigkeiten werden selbständig gelöst).

Dieses Konzept ist mit dem „build framework“ *GAR* vergleichbar, das von den Entwicklern der LNX-BBC Distribution stammt und inzwischen auch vom *Gnome* Projekt verwendet wird.

Weitere (Platz-)Optimierung:

- uclibc statt *libc*
- *multi-call-binary*: busybox

Debian woody & *uclibc*:

<http://people.debian.org/~andersee/>

Debian (*uclibc*) für *embedded* Systeme:

<http://www.emdebian.org/>