Das LINBO Handbuch

(Linux-basiertes Interaktives Netzwerk-Bootsystem)

Klaus Knopper

16. Dezember 2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

LINBO ist ein halb- bis vollautomatisch (je nach Konfiguration) arbeitender Bootmanager, der nicht nur in der Lage ist, verschiedene Betriebssysteme von Festplatte zu starten, sondern der auch Wartungs-, Update- und Reparaturfunktionen für Festplatteninstallationen übernimmt.

1.1 Funktionsweise / Technik

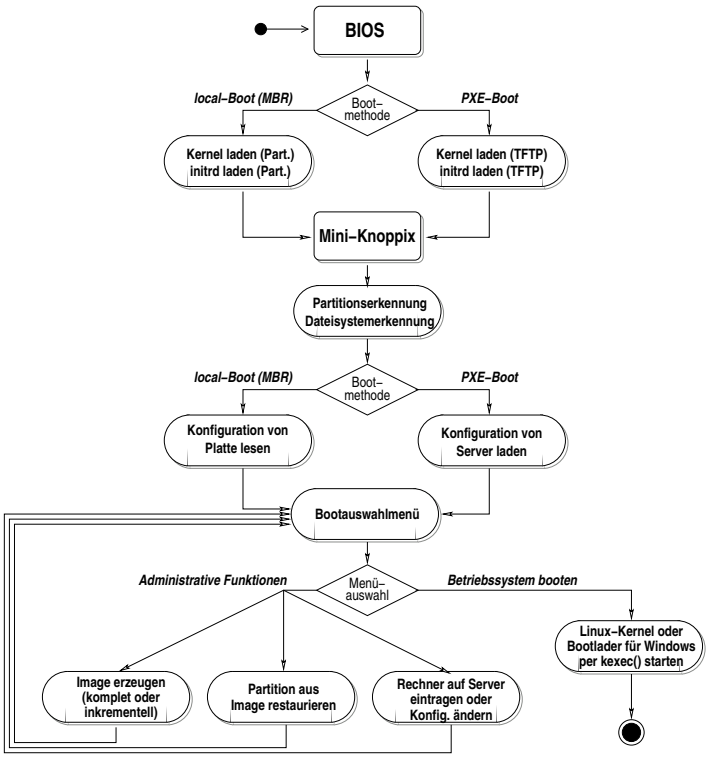

Abbildung 1: UML Aktivitätsdiagramm für LINBO

Vom Bootlader (**grub** lokal von Platte, oder Netzwerk PXE/Bootp-Server) werden Kernel und initiales Ram-Dateisystem geladen und gestartet. Nach normalerweise recht kurzer Startzeit wird eine graphische Oberfläche, **linbo_gui** mit Auswahlmöglichkeit präsentiert, während parallel dazu die Hardwareerkennung von Netzwerkkarten und Festplatten läuft. Nach Auswahl eines Buttons werden verschiedene Aktionen über das Worker-Backend-Skript **linbo_cmd** abgewickelt.

1.2 Testlauf in `qemu`

(Installation in einer richtigen DHCP-Server-Umgebung: Siehe Abschnitt 4.1.2 auf Seite 17.)

Das zentrale **`Makefile`** im LINBO-Entwicklungsverzeichnis bietet drei Testszenarien mit Hilfe von **`qemu`** als virtuelle Maschine(n):

`make test`	Direktes Booten von LINBO-Kern und Start des internen TFTP-Servers (Directory „Images“)
`make hdtest`	Booten von simulierter Festplatte **`Images/hda.img`**
`make pxetest`	Booten per PXE von einem durch qemu simulierten PXE-Server

Tabelle 1: Test-Start von LINBO aus der Entwicklungsumgebung

Achtung: Für die zuletzt genannte Option „Booten vom qemu PXE-Server“ ist die Installation von qemu mindestens Version 0.9.0+cvs erforderlich, da frühere Versionen den „bootp“-Parameter noch nicht kannten! Im Buildsystem befindet sich ein aktueller Snapshot von qemu, der mit **`make qemu`** als Debian-Paket gebaut wird (gcc-3.4 erforderlich).

1.3 Namen und Beschreibungen

1.3.1 Cache-Partition

Auf den mit LINBO verwalteten Rechnern wird eine Cache-Partition verwendet, um LINBO selbst und die von LINBO verwalteten Betriebssysteme lokal vorzuhalten und notfalls auch ohne Netzwerk starten zu können.

1.3.2 Multicast

Um den Cache mit den großen Image-Dateien (komprimiert ca. 500MB-2GB pro Betriebssystem je nach Ausstattung) effizient zu füllen, kann optional Multicast verwendet werden. Hierzu muss auf dem LINBO-Server **`udpcast`**, v.a. der **`udp-server`** installiert sein, welcher nach einer einstellbaren Mindestanzahl anfordernder Clients und ebenfalls einstellbarer Wartezeit das Senden der Images an mehrere Rechner gleichzeitig unterstützt. Hierdurch werden die Daten nur einmal tatsächlich übertragen, wenn mehrere Clients gleichzeitig den Cache mit Daten füllen, wodurch der Zeitaufwand beim erstmaligen Installieren oder Update von Clients drastisch reduziert wird. Beim Klick auf **`Sync+Start`** auf

den Clients wird hingegen stets auf dem Server nach einem Update des gewählen Images geschaut, und die Änderungen zur älteren Version per RSYNC übertragen. Sind keine Änderungen vorhanden, so wird die Version aus dem Cache weiterverwendet.

1.3.3 PXE

„Pre Execution Environment“ bezeichnet eine standardisierte Methode, ein Bootmenü oder Betriebssystem übers Netzwerk zu laden und zu starten. Hierfür ist entweder eine PXE-fähige Netzwerkkarte erforderlich (die in den meisten modernen Rechnern verfügbar ist), oder eine ensprechende Bootdiskette mit Treiber von **http://www.rom-o-matic.net**.

1.3.4 cloop

Das „Compressed Loopback Device“ ist ein von *iptables*-Autor Paul Russel und Klaus Knopper entwickeltes *Block-Device* Kernelmodul, das typischerweise eine Festplattenpartition in komprimierter Datei-Form enthält. In LINBO haben diese Dateien die Endung **.cloop**. Im Gegensatz zu den bekannten **zip** oder **tar.gz**-Archiven verhält sich ein über cloop eingebundenes Archiv wie eine echte Fesplattenpartition mit wahlfreiem Zugriff, die enthaltenen Daten und Teile davon werden beim Zugriff im Speicher dekomprimiert. In diesem Dateiformat ist es möglich, komplette Festplattenpartitionen mit allen Zusatzdaten wie *Boot-Record* und „versteckten“ Informationen leicht zugänglich zu halten. Auch das Herauskopieren einzelner Dateien ist dadurch möglich, ohne das gesamte Archiv auspacken zu müssen.

In LINBO werden alle Basis-Images (direkte Partitionsabzüge) in diesem Format unverändert gespeichert, was auf der Cache-Partition Platz spart und den Lesevorgang dadurch, dass weniger Lesezugriffe erfolgen, stark beschleunigt. Dieses Verfahren ist auch von der KNOPPIX-DVD bekannt. Die Kompressionsrate beträgt bei ausführbaren Programmen zirka 3:1, bei Textdateien bis 12:1, bei Zufallsdaten oder verschlüsselten Dateien oder bereits komprimierten Bildern allerdings nur noch ca. 1:1 bis 0,9:1.

1.3.5 rsync und das rsync-Batch-Format

rsync ist ein Synchronisierungs-Programm, das zwar wie viele Kopierprogramme auch zunächt eine 1:1 Kopie erzeugt, wobei aber tatsächlich nur die *Änderungen* zwischen Quelle und Ziel übertragen werden.

Weiterhin können, statt von einem Quellverzeichnis zu einem Zielverzeichnis zu kopieren, neuere Versionen von rsync das sog. "Batch"-Format verwenden, was besser mit „Binärdifferenz-Archiv" übersetzt werden kann. In diesem Dateiformat werden die Differenzen zum Originalverzeichnis inklusive zu löschender Dateien gespeichert, optional ebenfalls ähnlich wie bei *cloop* auch komprimiert, so dass es sich hervorragend für inkrementelle Archive eignet. LINBO legt inkrementelle Partitions-Images in diesem Format ab, in Dateien mit der Endung **`.rsync`**. Im Gegensatz zum mountbaren cloop-Format können diese Dateien aber ausschließlich von rsync verarbeitet werden.

Die für LINBO verwendete Version von **`rsync`** enthält einen Patch, der die die für das unter Windows verwendete NTFS-Dateisystem notwendigen Systemattribute mitkopieren kann, und eine TFS-freundlichere Namensgebung der Temporärdateien verwendet. Im Entwicklungssystem sind die entsprechenden Quelltexte unter **`Sources/rsync-*`** untergebracht.

2 Installation

LINBO wird üblicherweise per PXE gebootet, und kann sich selbst bootfähig auf die Cache-Partition (1.3.1) kopieren. Dadurch kann LINBO auch nach der ersten Installation direkt von Festplatte gestartet werden, wenn kein Netz vorhanden ist.

Wie im Abschnitt 4.1.2 beschrieben, sind grundsätzlich nur die beide Dateien **linbo** und **linbofs.gz** und die entsprechende Boot-Umgebung erforderlich, um Clients mit LINBO zu starten. Der Großteil der Installationsaufgabe besteht in der individuellen Anpassung von Konfigurationsdateien.

2.1 Bootvorgang

LINBO kann wie ein normaler Linux-Kernel gebootet werden, von lokaler Festplattenpartition, bootfähigem CD-Rom, USB Flashdisk oder über das Netzwerk von einem PXE/BOOTP-fähigen DHCP-Server.

Aus technischen Gründen[1] ist LINBO aufgesplittet in einen Kernel-Teil, Dateiname **linbo** (ca. 3-4 MB komprimiert), und einen Dateisystem-Teil, Dateiname **linbofs.gz** (ca. 8MB komprimiert), wobei der Kernelteil bereits ein kleines Dateisystem mit **busybox** als Minimalshell, und der Dateisystem-Teil die größeren Programme und Tools wie **linbo_gui**, Systembibliotheken, und eine Beispieldatei für **start.conf** enthält.

linbo und **linbofs.gz** werden für den Netzwerk-Boot üblicherweise auf dem DHCP+TFTP-Server installiert, siehe auch Abschnitt 4.1.2.

2.2 LINBO- und Dateisystem-Konfiguration

LINBO erhält seine Boot- und Konfigurationsdaten über folgende Methoden:

1. Bootparameter (Kernel „append“-Zeile), die sich z.B. per DHCP/pxelinux setzen lassen, diese sind:

   ip=*ip-adresse* FESTE IP-Adresse (wenn gewünscht) für diesen Client

   server=*ip-adresse* IP-Adresse des Servers, der die Images vorhält

[1] Es hat sich in Tests gezeigt, dass einige Netzwerkkarten keine Einzeldateien größer 8MB per TFTP beziehen können, außerdem ist der absolute Adressraum für den Kernel auf wenige MB begrenzt

cache=/dev/*Partitionsname* (s.a. Abschnit 6.4) die Partition, die die Betriebssystem-Images vorhält

debug Startet auf dem Client eine Debug-Shell vor dem GUI, um Fehlern auf die Spur zu kommen, oder manuell Einfluss auf die Konfiguration oder Partitionierung zu nehmen.

2. Aus der Datei **start.conf-*IP-Adresse***, die auf dem Server per rsync-Download angeboten wird. *IP-Adresse* ist die für diesen Client per Bootkommandozeile oder per DHCP festgelegte IP-Adresse. Hiermit kann für jeden Rechner eine spezielle Konfiguration vereinbart werden. Um Gruppen von Rechnern mit dergleichen Konfiguration zu definieren, genügt es, einen Symlink zu definieren. Ein Beispiel:
   ln -s start.conf-Klasse1A start.conf-192.168.0.2
   Hier wird ein Link auf eine gemeinsame Konfigurationsdatei (**start.conf-Klasse1A** in diesem Beispiel – Groß- und Kleinschreibung werden beachtet!) unter dem neuen Namen **start.conf-192.168.0.2** angelegt.

3. Aus einer Datei **start.conf**, die auf dem Server per rsync-Download angeboten wird. Dies ist quasi die „Default"-Einstellung, wenn weder per Bootkommandozeile, noch per Rechnerspezifischer **start.conf**-Datei Einstellungen vorgenommen werden.

4. Aus einer Datei **start.conf** auf der Cache-Partition des Client-Rechners.

5. Aus einer im LINBO-Dateisystem **linbofs.gz** integrierten **start.conf**-Datei. Dies ist der letzte Fallback, wenn kein Rsync-Server vorhanden und noch keine Cache-Partition eingerichtet ist, ansonsten wird kein LINBO-Menü angezeigt.

Der Aufbau der **start.conf**-Datei ist in Abschnitt 6.2 genau beschrieben.

3 Anwenderhandbuch

3.1 LINBO booten

LINBO kann sowohl übers Netz per PXE (1.3.3) als auch von einer bereits mit LINBO installierten Festplatte gestartet werden. Die Installation eines Bootservers für LINBO ist unter 4.1.2 beschrieben, die Installation des LINBO-Bootladers auf Festplatte unter 4.1.3.

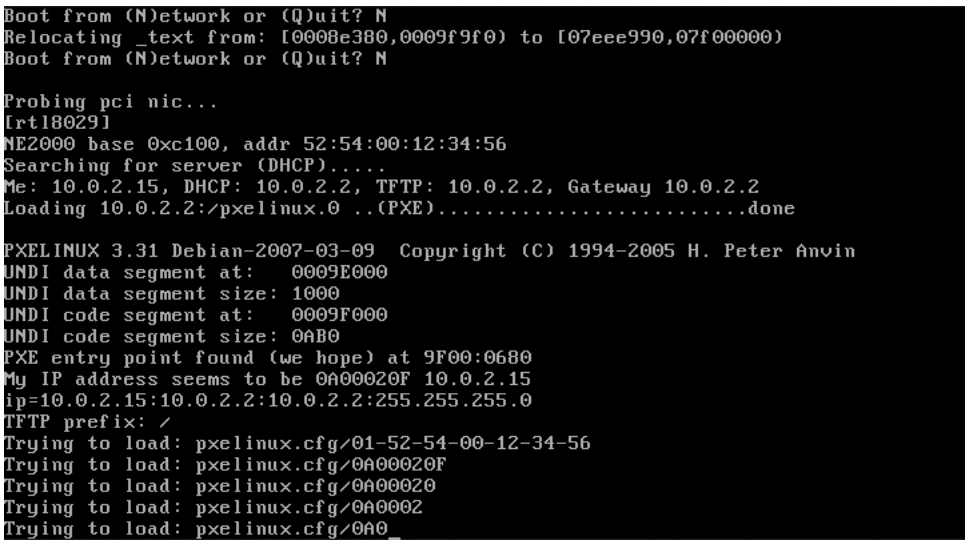

Abbildung 2: PXE-Bootlader in qemu

LINBO besteht aus einem Kernel- und einem Dateisystem-Teil, die separat geladen und anschließend automatisch im Hauptspeicher zusammengesetzt werden. Nach einer minimalen Hardwareerkennung (i.e. Grafikkarte, IDE+SATA-Controller, Netzwerkkarte und USB-Geräte) durch den Kernel, wird die graphische Oberfläche von LINBO, **`linbo_gui`**, gestartet.

Hinweis: Da parallel zum Start der Oberfläche eine weitere Hardwareerkennug stattfindet (Netzwerk, Festplattencontroller und -partitionen), stehen einige LINBO-Funktionen erst nach einigen Sekunden zur Verfügung. Normalerweise ist das vom GUI aufgerufene **`linbo_cmd`** Worker-Backend aber so flexibel, dass es bei noch nicht erkannten Festplattenpartitionen einige Zeit wartet, bis diese verfügbar sind.

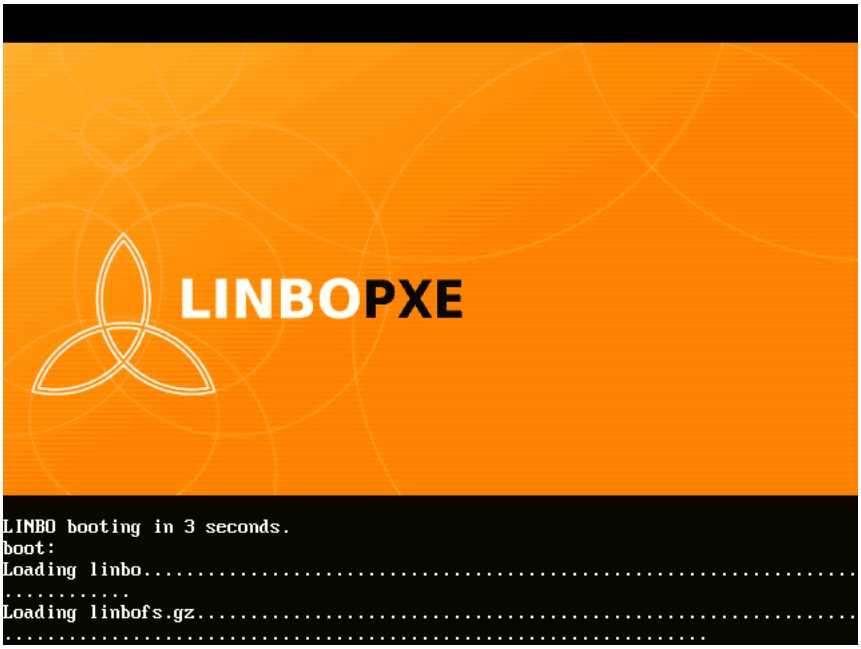

Abbildung 3: Die Bootkonsole von LINBO über `pxelinux`

3.2 Graphische LINBO Oberfläche

3.2.1 Betriebssysteme wiederherstellen und starten

Abbildung 4 zeigt das Startmenü von LINBO. Hier sind alle Betriebssysteme, die für LINBO vorbereitet und auf Festplatte installiert wurden, aufgeführt.

Mit Klick auf **`Sync+Start`** hinter dem Namen des Betriebssystems, wird das auf einer Partition befindliche System mit Hilfe eines auf der Cache-Partition (1.3.1) befindlichen Archivs Datei für Datei überschrieben bzw. in den Ursprungszustand versetzt. *Achtung: Hierbei gehen alle Änderungen, die in der letzen Session mit diesem Betriebssystem erstellt wurden, verloren.*

`Neu+Start` (Abbildung 5) lädt eine ggf. neuere Version des jeweiligen Betriebssstems vom Server, wahlweise per RSYNC oder Multicast, auf die Cache-Partition herunter, und installiert diese anschließend wie bei **`Sync+Start`**.

`Start` bootet das angegebene Betriebssystem so, wie es sich derzeit auf der Festplatte befindet. Der Rechner startet hierbei nicht neu, sondern LINBO führt einen „Soft-Reboot“ durch, was den Startvorgang stark beschleunigt. Treten beim Starten Fehler auf, oder befindet sich das installiere Betriebssystem nicht mehr in einem benutzbaren Zustand, so sollte nach dem nächsten Reboot mit Hilfe von **`Sync+Start`** oder **`Neu+Start`** wieder der zuletzt gespeicherte, arbeitsfähige

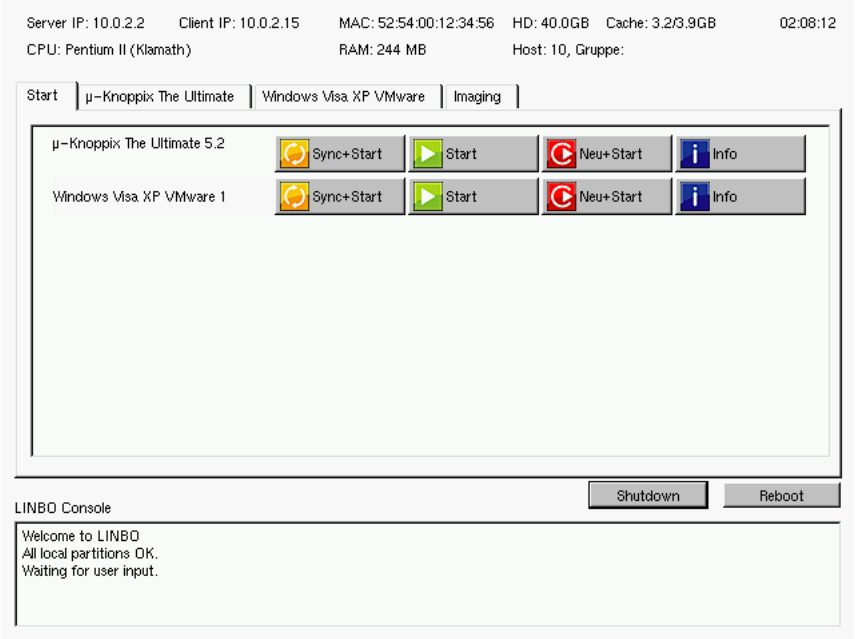

Abbildung 4: Startmenü von LINBO

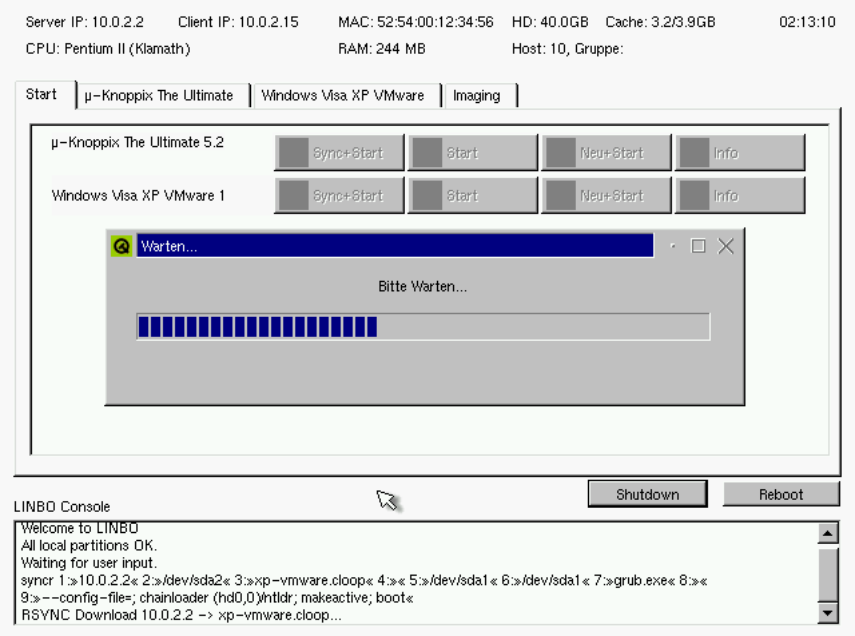

Abbildung 5: Neu aufsetzen eines Betriebssystems mit LINBO

Zustand restauriert werden, bevor ein neuer **`Start`** versucht wird.

Abbildung 6 zeigt ein Mini-Linux, das durch LINBO gestartet wurde.

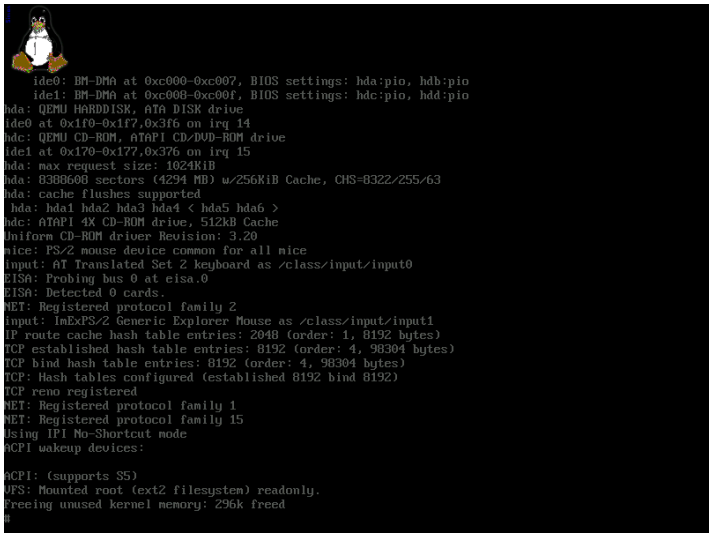

Abbildung 6: Aus LINBO gestartetes Mini-Linux

3.2.2 Betriebssystem-Images verwalten

Die „Reiter“ hinter dem Startmenü sind für die Verwaltung von Images (Archiven) der installierten oder zu installierenden Betriebssysteme zuständig. Hier können verschiedene Versionen eingespielt werden, die inkrementell auf ein- und demselben Basis-Image aufbauen.

Gegenüber „Sync+Start“ und „Neu+Start“ aus dem Startmenü erlauben die gleichnamigen Buttons in den Betriebssystem-Reitern also eine genauere Angabe der jeweiligen Version, während im Startmenü immer nur die erste Version aus der **`start.conf`**-Konfiguration restauriert wird.

4 Administration

4.1 Installation und Konfiguration

4.1.1 `start.conf` - Partitionen und Images

Die Konfigurationsdatei **`start.conf`** ist im Stil der bekannten KDE-Desktop-Iconbeschreibungen verfasst. Sie befindet sich normalerweise im gleichen Verzeichnis auf dem RSYNC-Server, in dem auch die Betriebssystem-Images für LINBO untergebracht sind.

Kommentare werden durch **#** eingeleitet, dürfen am Anfang einer Zeile oder mitten im Text aufauchen, und werden inklusive dem bis zum Zeilenende folgendem Text von **`linbo_gui`** ignoriert.

`[LINBO]` Abschnitt mit allgemeinen Einstellungen zu LINBO.

`[Partition]` Abschnitt mit der Definition einer Partition.
Partitionen müssen in der Reihenfolge definiert werden, in der sie auf der Festplattengeometrie eingerichtet werden sollen, und müssen mit Parameter **`Dev = /dev/partitionsname`** durchgehend (lückenlos) nach Unix-Konvention benannt werden (s.a. Tabelle 6.4). „Leere“ Partitionen mit Partitions-Id **`Id = 0`** und **`Size = 0`** sind erlaubt. Partitionen, die den verfügbaren Rest der Festplatte umfassen sollen, werden mit **`Size =`** ohne Wert gekennzeichnet (z.B. sinnvoll bei erweiterten artitionen oder der letzten Partition auf der Festplatte).

`[OS]` Abschnitt mit der Definition eines Betriebssystems.

Eine Beispieldatei, die sich im Entwicklungssystem in **`Binaries/linbo_gui/start.conf`** befindet, ist hier angegeben.

```
[LINBO]                 # LINBO global config
Cache = /dev/sda5       # Local (Client) Cache Partition
Server = 10.0.2.2       # RSYNC Server with remote Images

[Partition]             # Start of a Partition config section
Dev = /dev/sda1         # Device name of partition (sda1 = first partition on first IDE disk)
Size = 4200000          # Partition size in kB
Id = 7                  # Partition type (83 = Linux, 82 = swap, c = FAT32, 7 = NTFS, ...)
FSType = ntfs           # File system on Partition
Bootable = yes          # Mark this partition as bootable

[Partition]
Dev = /dev/sda2         # Device name of partition
Size = 4200000          # Partition size in kB
```

```
Id = 83                 # Partition type (83 = Linux, 82 = swap, c = FAT32, ...)
FSType = reiserfs       # File system on Partition
Bootable = no           # Mark this partition as non-bootable

[Partition]
Dev = /dev/sda3         # Device name of partition
Size = 1200000          # Partition size in kB
Id = 82                 # Partition type (83 = Linux, 82 = swap, c = FAT32, ...)
FSType = swap           # This is Linux swap space

[Partition]
Dev = /dev/sda4         # Device name of partition
Size =                  # Partition size in kB (empty if "remaining space")
Id = 5                  # Partition type (5 = Extended)
FSType =                # File system on Partition (none for extended partition)
Bootable = no           # Mark this partition as non-bootable (or Linux)

[Partition]
Dev = /dev/sda5         # Device name of partition
Size =                  # Partition size in kB (empty if "remaining space")
Id = 83                 # Partition type (83 = Linux, 82 = swap, c = FAT32, ...)
FSType = ext3           # File system on Partition
Bootable = no           # Mark this partition as non-bootable (or Linux)

[OS]
Name = $\mu$-Knoppix The Ultimate          # Name of OS
Version = 6.2           # Version/Date of OS (optional)
Description = 01.02.2009, 7:10 Standardinstallation # Descriptive Text
Image =                 # Filename of rsync batch, empty for none
BaseImage  = microknoppix.cloop # Filename of base partition image
Boot = /dev/sda2        # Partition containing Kernel & Initrd
Root = /dev/sda2        # root=/dev/partition Parameter (Root FS)
Kernel = vmlinuz        # Relative filename of Kernel or Boot image
Initrd = initrd.gz      # Relative filename of Initrd
Append = acpi=noirq     # Kernel cmdline, root= will be added (optional)
StartEnabled = yes      # Enable "Start" Button
SyncEnabled = yes       # Enable "Sync+Start" Button
NewEnabled = yes        # Enable "Overwrite partition" Button

[OS]
Name = Windows HastaLaVista  # Name of OS
Description = 06.02.2007, 10:10 Es bootet.          # Descriptive Text
Version = 1             # Version/Date of OS (optional)
Image = xp-20070727.rsync # Filename of rsync batch
BaseImage  = xp.cloop   # Filename of base partition image
Boot = /dev/sda1        # Partition containing Kernel & Initrd
Root = /dev/sda1        # root=/dev/partition Parameter (Root FS)
Kernel = grub.exe       # Relative filename of Kernel or Boot image
Initrd =                # Relative filename of Initrd
Append = --config-file=chainloader (hd0,0)/ntldr; boot
StartEnabled = yes      # Enable "Start" Button
SyncEnabled = yes       # Enable "Sync+Start" Button
NewEnabled = yes        # Enable "Overwrite partition" Button
```

Im **[OS]**-Abschnitt dürfen Namen von Betriebssystemen mehrfach genannt werden, wobei die nachfolgenden Einstellungen und inkrementellen Image-Namen dann als Versionspaket dieses Betriebssystems interpretiert werden, und in den einzelnen Reitern für die Betriebssysteme im **linbo_gui** als „Derivat“ auftau-

chen.

Jede Konfigurationsoption muss in einer Zeile für sich stehen, ohne Zeilenumbrüche!

Parameter	Wert (Beispiel)	Bedeutung
	[LINBO]	
AutoFormat	`yes`	Die Partitionen werden automatisch formatiert.
AutoInitCache	`yes`	Die Cache-Partition wird automatisch gefüllt.
AutoPartition	`yes`	Die Partitionstabelle wird automatisch wiederhergestellt.
Cache	`/dev/sda5`	Die erste logische Partition (/dev/sda5) enthält die gespeicherten Betriebssystem-Images.
RootTimeout	`300`	Die Zeit, bis die Anmeldung als „Admin“ abläuft, beträgt zu Beginn 300 Sekunden.
Server	`10.0.2.2`	Der Rechner 10.0.2.2 ist der RSYNC-Server, und liefert die Images.
UseMulticast	`yes`	Multicast anstelle von rsync für die Image-Übertragung vom Server verwenden (muss auf dem Server eingerichtet werden, s.a. Abschnitt 1.3.2 Seite 4).
	[Partition]	
Bootable	`yes`	„Bootable“-Kennung in der Partitionstabelle für diese Partition setzen (wird in seltenen Fällen vom Rechner-BIOS benötigt).
Dev	`/dev/sda1`	Partitionsname (Unix-Syntax), hier: erste Partition der ersten SCSI oder SATA/PATA-Festplatte.
Id	`83`	Partitions-ID, wie in **`fdisk`**. `83` = Linux, `83` = Swap `c` = FAT32, `5` = „Erweiterte Partition“, `0` = Leer ...
FSType	reiserfs	Zu formatierendes Dateisystem, z.B. `ext2`, `ext3`, `ntfs`, `reiserfs`, `swap`, `vfat` (FAT32, DOS)
Size	`41943040`	Partitionsgröße in kB, hier: 40GB. Wenn leer: kompletter noch verfügbarer Platz.
	[OS]	
Append	`vga=791`	Boot-Optionen für den angegebenen Betriebssystem-Kern. Bei **`grub.exe`** als `Kernel` kann hier beispielsweise als Option **`--config-file=chainloader (hd0,0)/ntldr`** oder **`--config-file=chainloader (hd0,0)+1`** stehen, um den Windows-Bootlader **`ntldr`** bzw. direkt den Partitions-Bootrecord anzusprechen.

Autostart	`yes`	Das Betriebssystem wird ohne Klick direkt gestartet
BaseImage	`debian.cloop`	Vollständiger Partitions-Abzug im **`cloop`**-Format
Boot	`/dev/sda2`	Partition, auf der sich der Betriebssystem-Kern (Kernel) befindet. Bei Windows wird hier **`grub.exe`** also Bootloader angegeben.
Description	`Mathe-Kurs`	Einzeilige Kurzbeschreibung
Hidden	`yes`	Das Betriebssystem bekommt keinen separten Reiter in LINBO.
Image	`debian-m.rsync`	Auf dem **`BaseImage`** aufbauende Installation im **`rsync-Batch`**-Format
Initrd	`initrd.gz`	Initial Ramdisk (wird von den meisten Linux-Distributionen verwendet) auf der Boot-Partition
Name	`Debian`	Betriebssystem-Name, der in LINBO angezeigt wird
NewEnabled	`yes`	Der „Neu+Start“ (Überschreiben der kompletten Partition erzwingen) Knopf ist klickbar
Root	`/dev/sda2`	Partition, auf der sich das Haupt-Dateisystem (**`/`**) des Betriebssystems befindet
StartEnabled	`yes`	Der „Start“ (ohne Synchronisation) Knopf ist klickbar
SyncEnabled	`yes`	Der „Sync+Start“ (mit vorzugsweise lokaler Synchronisation) Knopf ist klickbar
Version	`1.0 vom 5.5.2009`	Versionsnummer für dieses Betriebssystem, die in LINBO angezeigt wird

Tabelle 3: **`start.conf`** Einträge und ihre Bedeutung

ACHTUNG: Die Optionen **`AutoFormat`** und **`AutoPartition`** im Konfigurationsabschnitt **`[`**LINBO] löschen bei jedem Start alle bereits installierten Partitionen, und sind nur für die automatisierte Erstinstallation der Rechner sinnvoll. Sie sollten im Normalbetrieb immer auf „no“ gesetzt werden.

Die Cache-Partition darf auch per NFS oder CIFS (Samba, Windows-Freigabe) eingebunden werden, wenn statt einer Partitionsangabe die entsprechende Netzfreigaben-Syntax verwendet wird, beispielsweise:

Cache = 10.0.2.2:/linbo-cache für NFS
Cache = //10.0.2.2/linbo-cache für CIFS/SAMBA

Sofern die Netwzerk-Verzeichnisse nur Read-Only freigegeben sind, was aus Sicherheitsgründen sehr zu empfehlen ist, ist das Erzeugen bzw. Überschreiben von Images von LINBO aus nicht möglich.

4.1.2 PXE-Konfiguration (DHCP-Server)

LINBO-Kernel (**linbo**) und LINBO-Dateisystem (**linbofs.gz**) müssen sich in einem per TFTP erreichbaren Verzeichnis auf dem Server befinden. Ein klassischer Name für dieses Verzeichnis ist auf vielen Unix-Systemen **/tftpboot**. Der für LINBO empfohlene TFTP-Server **tftpd-hpa** könnte beispielsweise auf dem Server wie folgt gestartet werden, wenn **linbo** und **linbofs.gz** im Verzeichnis **/var/linbo** liegen.

```
sudo tftpd -l /var/linbo
```

Im DHCP-Server sind dann die Clients bzw. Client-Netze anzugeben, die per LINBO verwaltet werden sollen. Optional können für verschiedene Rechner auch entsprechend verschiedene **linbofs.gz** in der Konfiguration von **pxelinux.cfg/CLIENT-ADRESSE** angegeben werden, in denen sich jeweils eine andere **start.conf**-Konfigurationsdatei befinden kann.

Beispiel für einen entsprechenden Abschnitt aus der **dhcpd.conf** des ISC-dhcpd Version 3:

```
allow booting;
allow bootp;

subnet 10.0.2.0 netmask 255.255.255.0 {
  next-server 10.0.2.2;
  filename "pxelinux.0";
  option subnet-mask 255.255.255.0;
  range 10.0.2.10 10.0.2.15;
  option domain-name-servers 10.0.2.2;
  option routers 10.0.2.2;
}
```

In diesem Beispiel werden die IP-Adressen 10.0.2.10 bis einschließlich 10.0.2.15 dyamisch vergeben, die Clients starten per TFTP den PXE-Bootlader **pxelinux.0**, der seine Konfigurationsdatei unter **pxelinux.cfg/default** nachlädt. LINBO-Kern und Images müssen ebenfalls per TFTP erreichbar sein.

Hinweis: Üblicherweise fügen die Clients ein Pfad-Präfix **/** zum Dateinamen hinzu, daher sollte mit **tftp [host]** getestet werden, ob der LINBO-Kernel auf dem Server mit **get /linbo** erreichbar ist.

4.1.3 RSYNC-Konfiguration (Server)

Zur Synchronisation von Images sowie zum Upload neu erzeugter Images wird **rsync** (s. Abschnit 1.3.5) verwendet.

Unter Debian wird rsync installiert mit **aptitude install rsync**. Damit die Clients Zugriff auf die Images bekommen, muss zunächst eine rsync-Freigabe **[linbo]** in **/etc/rsyncd.conf** auf dem Server eingerichet werden:

```
[linbo]
comment = LINBO Image directory (read-only)
path = /var/linbo
use chroot = no
lock file = /var/lock/rsyncd
read only = yes
list = yes
uid = nobody
gid = nogroup
dont compress = *.cloop *.rsync *.gz
```

Dieses Beispiel erlaubt einen nur lesenden Zugriff auf die Dateien im Verzeichnis **/var/linbo** für alle Clients ohne Passwort. Mit

rsync *server-adresse*::linbo

können Sie das Verzeichnis testweise per rsync auflisten lassen, ohne eine Datei übertragen zu müssen.

Für die Übertragung von Images vom Client-Rechner zum Server, für neu erstellte Images, ist außerdem die Einrichtung eines *schreibbaren* rsync-Repository erforderlich. Der entsprechende zusäzliche Eintrag in **/etc/rsyncd.conf**:

```
[linbo-upload]
comment = LINBO Upload directory
path = /var/linbo
use chroot = no
lock file = /var/lock/rsyncd
read only = no
list = yes
uid = root
gid = root
```

```
dont compress = *.cloop *.rsync *.gz
auth users = linbo
secrets file = /etc/rsyncd.secrets
```

Das tatsächliche Verzeichnis im Dateisystem ist in diesem Beispiel wieder das Verzeichnis **/var/linbo**. Dort sollten sich der Linbo-Kernel **linbo** und das Linbo-Dateisystem **linbofs.gz** befinden, sowie die Image-Dateien mit den Betriebssystemen, Partitionsdumps und inkrementelle Änderungen (Abschnitt 1.3.5), für die Clients. Mit (Beispiel)

rsync datei.txt linbo@*server-adresse*::linbo-upload

können Sie eine Testdatei (datei.txt) an den Server übertragen (allerdings klappt dies erst nach dem nächsten Konfgurationsschritt). Hierbei sollten Sie nach einem Passwort gefragt werden, was auch der Authentifizierung in LINBO dient.

Dieses Login/Passwort-Paar für die rsync-Freigabe **linbo-update** muss noch eingetragen werden, in die oben angegebene Datei **/etc/rsyncd.secrets**. Beispiel:

```
linbo:test
```

Der Benutzername „**linbo**“ ist momentan vom LINBO-System vorgegeben, das Passwort (hier: **test**) können Sie frei wählen. Das Passwort ist für die Sicherheit des LINBO-Systems essentiell, und sollte nur den Administratoren, die LINBO-Clients erstmalig aufsetzen und auch neue Images auf dem Server einspielen dürfen, bekannt sein. Für den normalen Betrieb von LINBO, also das Aktualisieren und Booten von Betriebssystemen auf LINBO-Clients, ist das Passwort nicht erforderlich.

Bitte beachten Sie, dass die Datei **/etc/rsyncd.secrets** nur für den rsync-Server lesbar sein darf, sonst verweigert rsync jedes Passwort. Mit dem Linux-Kommando **chmod 400 /etc/rsyncd.secrets** (als Administrator) sollte dies gewährleistet sein.

Fehlermeldungen, Warnungen und Statusinformationen von rsync finden Sie auf den meisten Linux-Distributionen in den Logdateien **/var/log/syslog** oder **/var/log/daemon.log**.

Falls der rsync-Server die Änderungen an seiner Konfiguration nicht automatisch erkennt, muss er neu gestartet werden: **/etc/init.d/rsync restart**.

Ist der rsync-Server konfiguriert, so müssen noch der LINBO-Kernel (**linbo**) und das LINBO-Dateisystem ((**linbofs.gz**), sowie für das Booten von Windows, **grub.exe** in das LINBO-Verzeichnis (in unserem Beispiel **/var/linbo**) kopiert werden.

Damit LINBO diese Daten nicht jedesmal erneut herunterlädt, sollten auch die zu den genannten Dateien passenden **.info**-Dateien kopiert bzw. erzeugt werden, wie in diesem Beispiel **linbofs.gz.info**:

```
[linbofs.gz]
timestamp=200707251505
imagesize=4198533
```

In diesen ist ein Zeitstempel und die Dateigrößen der Dateien vermerkt, so dass der Download der großen Dateien ggf. von LINBO übersprungen werden kann, wenn die Dateien im Cache noch aktuell sind. Das gleiche Verfahren wird auch bei den wirklich großen Image-Dateie angewandt. Hier erzeugt allerdings LINBO automatisch die entsprechenden info-Dateien und lädt sie mit hoch.

Beispiel: Kopieren der LINBO-Dateien ins rsync-Repository.

```
for i in linbo linbo.info linbofs.gz linbofs.gz.info; do
 install -m 644 /home/development/LINBO/Images/$i /var/linbo/
done
```

5 LINBO-Buildsystem

5.1 Systemvoraussetzungen

1. Installiertes POSIX-konformes Betriebssystem mit Bourne-kompatibler Shell (z.B. Debian „etch“). Cygwin sollte auch evtl. auch funktionieren, mit Linux-Crosscompiler.
2. GNU-Tar (zum Entpacken das Archives)
3. GNU-Make
4. GNU C-Compiler Version 3 oder höher (gcc-3.4 für qemu)
5. GNU-Binutils (zum Compilieren verschiedener Kernel-Komponenten notwendig)
6. Root-Rechte sind zum Bauen von LINBO NICHT erforderlich. Das Kernel-Buildsystem sorgt dafür, dass die Dateien im initramfs die erforderlichen Rechte erhalten, und dass auch Device-Dateien korrekt angelegt werden, daher kann als normaler User am System gearbeitet werden.
7. Zum Testen/Debuggen: qemu Version 0.9.0+cvs oder höher (-bootp Option erforderlich für simulierten PXE-Boot).

Der Bau von LINBO wird durch ein Makefile im LINBO-Verzeichnis gesteuert. **`make`** ohne Parameter liefert eine Kurzhilfe. Die einzelnen Schritte des Bauvorgangs sind recht selbsterklärend.

5.2 Verzeichnisse

`Binaries` enthält statische Binaries sowie dynamische Executables und Libraries für das initramfs. Die **`Binaries/*.sh`**-Dateien sind Shellskripte, die in LINBO den Bootvorgang und das GUI steuern.

Die für LINBO benötigten Dateien und Libraries werden in den Dateien **`Kernel/initramfs_kernel.d/*.conf`** sowie **`Kernel/initramfs.d/*.conf`** verwaltet. Dort sind auch neu hinzugefügte Dateien einzutragen, wenn sie in das initramfs aufgenommen werden sollen.

Das Verzeichnis **`Graphics`** enthält die Quellen des LINBO-Logos Binaries/linbo.xpm, das im GUI dargestellt wird, sowie den Desktop-Hintergrund und das PXE-Bootbild

für LINBO. Diese Dateien werden nicht direkt in **Graphics** verwendet, sondern müssen bei Bedarf nach **Images** kopiert werden.

GUI enthält die Sourcen für **linbo_gui**, LINBOs graphische Oberfläche, sowie embedded Qt als Abhängigkeit (aus Platzgründen nicht im Repository).

Documentation enthält das Benutzer- und Administrationshandbuch von LINBO (u.a. dieses Dokument).

Kernel enthält den für LINBO verwendeten Linux-Kernel-Source. Wenn dieser aktualisiert wird, sollte die alte .config-Datei weiterverwendet werden, da sie die für das initramfs notwendigen Einstellungen enthält.

Images enthält den fertig gebauten LINBO-Kernel **linbo** als Hardlink auf Kernel/linux-*/arch/i386/boot/bzImage, **linbofs.gz** als zusätzliches initramfs mit den größeren Dateien, sowie den PXE-Bootlader **pxelinux.0**, Images und Archive zur Installation von LINBO und den gewünschten Betriebssystemen auf den Clients.

Sources ist ein Archiv der für die gebauten Binaries verwendeten Quelltexte, um die Binaries selbst neu bauen zu können, und die GNU GENERAL PUBLIC LICENSE §3 zu erfüllen. Für das Buildsystem ist das Verzeichnis eigentlich irrelevant, da es im Makefile nicht verwendet wird. Neu integrierte Programme sollten jedoch gewissenhaft in `Sources` archiviert werden (Debian: **cd Sources ; aptitude source paketname**).

5.3 Bauvorgang

Durch „**make**“ ohne Parameter dokumentiert:

```
WELCOME TO THE LINBO BUILD SYSTEM

make kernel (Re-)Build Kernel and Modules (recommended before "make linbo")
make linbofs (Re-)Build LINBO-FS
make linbo (Re-)Build LINBO-Kernel and LINBO-FS
make config Configure LINBO kernel and edit LINBO filesystem.
make clean Cleanup LINBO kernel source for recompilation.
make test Run LINBO kernel in qemu
make hdtest Run LINBO from a harddisk-installed LINBO session
make pxetest Run LINBO in a qemu simulated PXE network boot
             (qemu 0.9.0+cvs version required that supports '-bootp')

Don't worry about the sequence of build commands, this Makefile will tell you
what to do first, in case anything is missing.

Have a lot of fun. ;-)
```

6 Das LINBO-Kochbuch, „How do I ...?“

6.1 Der allererste Start: Wie richte ich ein Master-System für LINBO-Images ein?

Grundsätzlich ist es praktisch, wenn das Master-System mit Hilfe von LINBO partitioniert wurde, damit die Partitionsgrößen und -namen zuverlässig festgelegt sind. Siehe Rezept 6.2.

```
[LINBO]
Cache = /dev/sda2
Server = 10.0.2.2

[Partition]
Dev = /dev/sda1         # Windows-Partition
Size = 4000000
Id = 7
FSType = ntfs
Bootable = yes

[Partition]
Dev = /dev/sda2         # Cache-Partition
Size = 4000000
Id = 83
FSType = reiserfs
Bootable = no

[OS]
Name = Win
Description = Installation vom 06.02.2007
BaseImage  = win.cloop     # Komplette Partition (Basis)
Image = win-20070727.rsync # Inkrementelles Update
Boot = /dev/sda1
Root = /dev/sda1
Kernel = grub.exe
Initrd =
Append = --config-file=map(rd) (hd0,0); map --hook; ... (s.o.)
StartEnabled = yes
SyncEnabled = yes
RemoteSyncEnabled = yes
```

Mit einer minimalen **`start.conf`** wir der hier gezeigten könnte beispielsweise begonnen werden. Bitte beachten Sie, dass für LINBO eine Cache-Partition (hier: /dev/sda2) eingerichtet werden muss, die groß genug ist, um alle Betriebssysteme vorzuhalten, die auf den Clients installiert werden sollten (plus etwas Platz für vielleicht einmal selbsterzeugte Voll- und Inkrementalimages). Siehe Rezept 6.3.

Die etwas längliche **`grub.exe`**-Zeile unter **`Append`** ist hier nur unvollständig wiedergegeben (s.a. Abschnitt 4.1.1 auf S. 15).

Nach der Partitionierung durch LINBO kann das gewünschte Betriebssystem auf der (in diesem Beispiel) ersten Partition mit einer Installations-CD eingerichtet

werden, und anschließend mit LINBO in ein Image (Basisimage **`win.cloop`**, spätere Änderungen in **`win-20070727.rsync`**) umgewandelt und zum LINBO-Server übertragen werden.

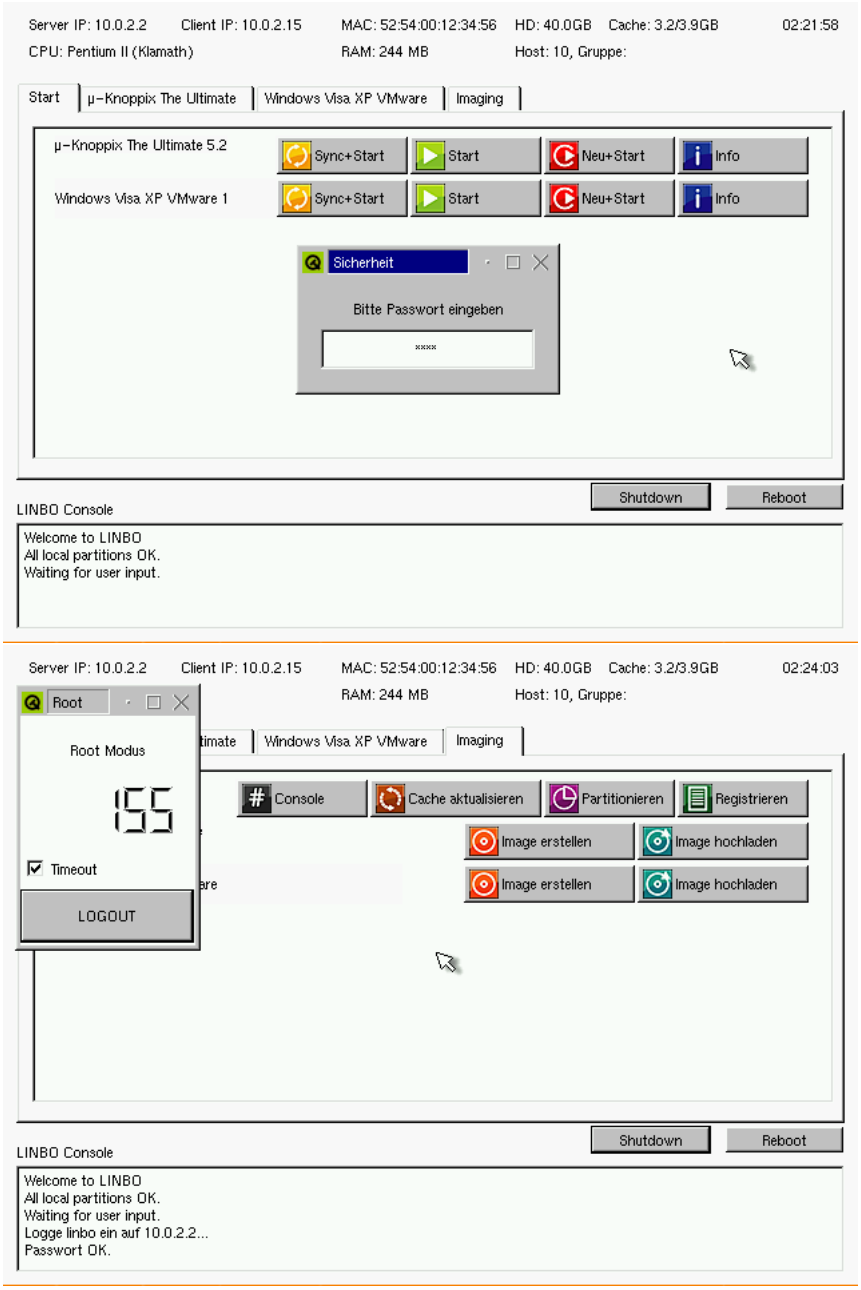

Abbildung 7: Einloggen als Admin bei LINBO

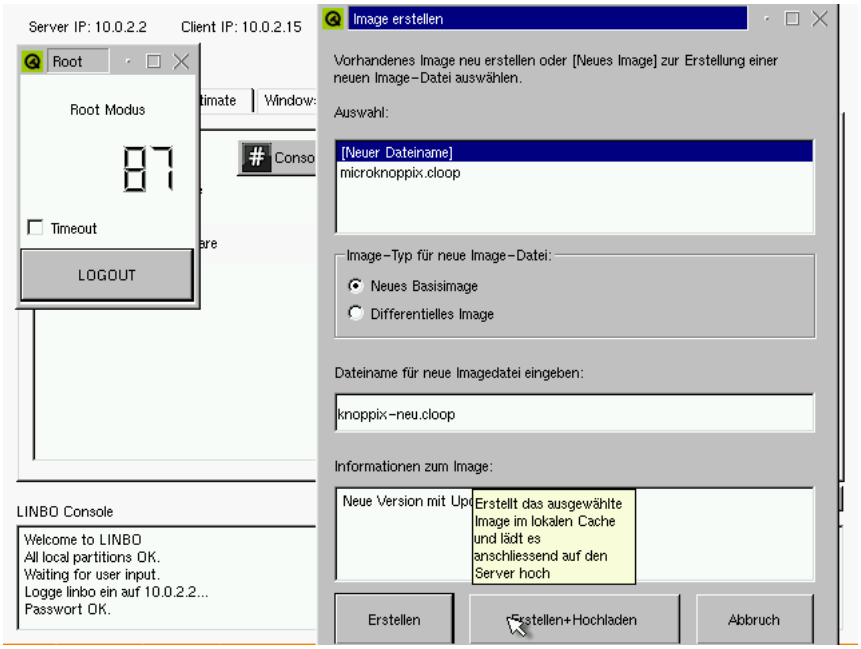

Abbildung 8: Image von Partition erzeugen, Dateiname

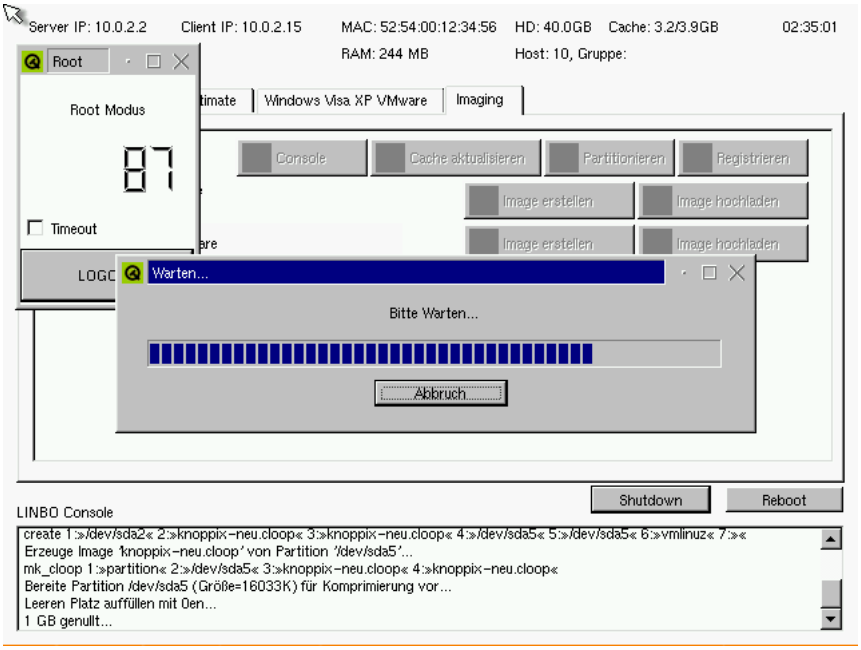

Abbildung 9: Image von Partition erzeugen, Kompression

6.2 Was muss in der `start.conf` stehen?

Dies ist im Detail in Abschnitt 4.1.1 auf Seite 13 beschrieben.

6.3 Wie groß soll die Cache-Partition sein, und wo genau soll sie auf der Festplatte liegen?

Die Cache-Partition hält eine Kopie der Installations-Images für jedes Betriebssystem, das auf den Clients bei Bedarf neu aufgesetzt, synchronisier oder aktualisiert werden soll. Grundsätzlich ist es LINBO egal, auf welcher Partitionsnummer diese Partition liegen muss, bzw. ob es eine „primäre“ (/dev/sda1...sda4) oder „logische“ (/dev/sda5...∞) Partition (Festplatten-Jargon, s.a. Rezept 6.4) ist. Der Name der Partition muss in der **`start.conf`** unter Abschnitt **`[LINBO]`**, Parametername **`Cache`** eingetragen werden, und die Größe in einem Abschnitt **`[Partition]`** so wie in diesem Beispiel:

```
[LINBO]
Server = 10.0.2.2
Cache = /dev/sda2  # <- Das ist die Cache-Partition

[Partition]
Dev = /dev/sda2    # Name
Size = 20000000    # Größe in kB
Id = 83            # Partitionstyp (83 = Linux)
FSType = reiserfs  # Dateisystem
Bootable = no      # Egal
```

In diesem Beispiel (die Kommentare mit **#** sind optional) wird eine 20 GB große Cache-Partition verwendet, die mit **`reiserfs`** formatiert ist.

6.4 Wie setzen sich die Partitionsnamen unter Linux zusammen, was muss ich angeben?

Das standard-Partitionsschema bei PC-Festplatten erlaubt, unabhängig davon, ob Linux oder Windows zum Einsatz kommt, maximal 4 „primäre“-Partitionen. Unter Linux heißen diese **`/dev/sda1`** ... **`/dev/sda4`** (SCSI, SATA, USB-Flash oder PATA) bzw. **`/dev/hda1`** ... **`/dev/hda4`** (ältere IDE-Controller). Bei Linux als Betriebssystem kann es durchaus vorkommen, dass LINBO **`/dev/sd*`**

verwendet, aber die instalierte Distribution **/dev/hd*** für die gleiche Festplatte, da dies von den verwendeten Treibern bzw. Kernel-Modulen für de Festplattencontroller abhängt. Bei Windows gibt es hingegen gar kein einheitliches Schema für die „aufwerksbuchstaben“, wobei aber **C:** für die Festplattenpartition, auf der Windows installiert ist, als Konvention verwendet wird, und **A:** und **B:** eher für die aus der Mode gekommenen Diskettenlaufwerke, die nichts mit Festplatten zu tun haben, verwendet wird.

Eine der primären (ersten 4) Partitionen darf eine „erweiterte“ Partition (Partitions-Kennung 5) sein, auf der sich dann weitere sog. „logische“ Partitionen einrichten lassen. Dies ist sinnvoll, wenn die Anzahl der benötigten Personen größer als die erlaubten 4 Partitionen sind.

Windows fühlt sich auf der allerersten Partition einer Festplatte (**/dev/sda1** unter Linux bzw. „**C:**“ unter DOS/WIndows) am wohlsten, wobei dies eine primäre Partition sein sollte. Linux hingegen kann auf jeder beliebigen Partition installiert werden, der Kernel findet auch die „logischen“ Partitionen des erweiterten Partitionsschemas, um das Wurzelverzeichnis zu mounten. Für Linux sollte allerdings auch eine Swap-Partition vorhanden sein, die als RAM-Erweiterung dient und bei Speichermangel die gerade nicht benötigten Daten aufnehmen bzw. „auslagern“ kann. Empfohlene Größe ist je nach Bedarf und Größe des Hauptspeichers üblicherweise 1-4 GB, Partitionstyp ist 82 („Linux swap“).

Name (Device)	**Bedeutung**
/dev/sda	Erste Festplatte komplett (SCSI, SATA, PATA, USB)
/dev/sda1	Erste Festplatte, erste (1. primäre) Partition
/dev/sda2	Erste Festplatte, zweite (2. primäre) Partition
/dev/sda3	Erste Festplatte, dritte (3. primäre) Partition
/dev/sda4	Erste Festplatte, vierte (4. primäre) Partition
/dev/sda5	Erste Festplatte, fünfte (bzw. 1. logische) Partition
/dev/sda6	Erste Festplatte, sechste (bzw. 2. logische) Partition
...	
/dev/sdb	Zweite Festplatte komplett (SCSI, SATA, PATA, USB)
/dev/sdb1	Zweite Festplatte, erste (1. primäre) Partition
...	
/dev/hda	Erste Festplatte komplett (älterer IDE-Treiber)
/dev/hda1	Erste Festplatte, erste (1. primäre) Partition (wie **sda1**)
...	

Tipps:

1. Die in der **start.conf** angegebene Partitionsgröße sollte sicherheitshalber etwas größer gewählt werden, als eigentlich notwendig für das ausgepackte cloop-archivierte Betriebssystem, da die meisten Partitionierungsprogramme die angegebenen Partitionsgrößen immer auf volle Zylindergrenzen aufrunden, daher kann der Inhalt der entsprechenden **.cloop**-Datei größer sein, als die ursprünglich gewählte Partitionsgröße zulassen würde.

2. Wird ein vorinstalliertes Linux mit LINBO geklont, so sollte darauf geachtet werden, dass entweder die Partitionsreihenfolge identisch zum Originalsystem gewählt wird, oder in der Konfigurationsdatei **/etc/fstab** des installierten Systems eine Anpassung vorgenommen wird, in der die neue Partitionierung berücksichtigt wird. S.a. Rezept 6.7. Die Partition, auf der sich das Hauptdateisystem befinden (**/**) wird von LINBO autommatisch aufgrund des Parameters **Root = ...** im Abschnitt **[OS]** der **start.conf** eingetragen.

6.5 Welche Partitions-IDs muss ich in start.conf angeben? (Id = ?)

Die folgende Tabelle resutiert aus der Ausgabe des unter Linux gebräuchlichen Partitionierungs-Programms **fdisk**:

```
 0  Empty           1e  Hidden W95 FAT1 80  Old Minix       be  Solaris boot
 1  FAT12           24  NEC DOS         81  Minix / old Lin bf  Solaris
 2  XENIX root      39  Plan 9          82  Linux swap / So c1  DRDOS/sec (FAT)
 3  XENIX usr       3c  PartitionMagic  83  Linux           c4  DRDOS/sec (FAT)
 4  FAT16 <32M      40  Venix 80286     84  OS/2 hidden C:  c6  DRDOS/sec (FAT)
 5  Extended        41  PPC PReP Boot   85  Linux extended  c7  Syrinx
 6  FAT16           42  SFS             86  NTFS volume set da  Non-FS data
 7  HPFS/NTFS       4d  QNX4.x          87  NTFS volume set db  CP/M / CTOS
 8  AIX             4e  QNX4.x 2nd part 88  Linux plaintext de  Dell Utility
 9  AIX bootable    4f  QNX4.x 3rd part 8e  Linux LVM       df  BootIt
 a  OS/2 Boot Manag 50  OnTrack DM      93  Amoeba          e1  DOS access
 b  W95 FAT32       51  OnTrack DM6 Aux 94  Amoeba BBT      e3  DOS R/O
 c  W95 FAT32 (LBA) 52  CP/M            9f  BSD/OS          e4  SpeedStor
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a0  IBM Thinkpad hi eb  BeOS fs
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a5  FreeBSD         ee  EFI GPT
10  OPUS            55  EZ-Drive        a6  OpenBSD         ef  EFI (FAT-12/16)
11  Hidden FAT12    56  Golden Bow      a7  NeXTSTEP        f0  Linux/PA-RISC b
12  Compaq diagnost 5c  Priam Edisk     a8  Darwin UFS      f1  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       a9  NetBSD          f4  SpeedStor
16  Hidden FAT16    63  GNU HURD or Sys ab  Darwin boot     f2  DOS secondary
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fd  Linux raid auto
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fe  LANstep
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid ff  BBT
1c  Hidden W95 FAT3 75  PC/IX
```

Tabelle 4: Partitions-IDs nach **fdisk**

Aus dieser recht umfangreichen Tabelle sind für die Praxis eigentlich nur die folgenden Werte für Windows und übliche Linux-Distributionen interessant:

ID	Bedeutung
0	Leere Partition, völlig unsichtbar
5, f	Erweiterte Partition (Linux, Windows), auf der sich weitere "logische" Partitionen befinden können
7	Windows (NTFS) Dateisystem Partition
c	Windows (95, 98) Datesystem Partition
82	Linux Swap Partition (virtueller Speicher Auslagerungsbereich)
83	Linux Dateisystem Partition

Tabelle 5: Zusammenfassung der für LINBO wichtigen Partitions-IDs

Die Partitionstypen **0**, **5** und **f** nehmen eine Sonderstellung ein, da auf ihnen nicht direkt Daten gespeichert werden, sondern sie dienen der weiteren Strukturierung (**5** oder **f**) der Festplatte, bzw. als „Platzhalter" für zukünftige Weiterpartitionierung (**0**). Speziell bei der Kennung für erweiterte Partitionen, auf denen (fast) beliebig viele „logische" Partitionen utergebracht werden können, wird gerne der restliche verfügbare Platz der Festplatte eingetragen (leer hinter **Size =** in **start.conf**).

Obwohl bei den meisten Distributionen standardmäßig Partitionstyp **83** verwendet wird, um das Dateisystem für Linux zu installieren, ist Linux selbst der Wert der Partitions-ID egal, d.h. es läuft auch, wenn ein vermeintlich „falscher" Partitionstyp wie `6`, `7`, `c` ... angegeben ist. Relevant für Linux ist vielmehr, mit welchem Dateisystem die Partition tatsächlich formatiert wird (**FSType = ...**), was im Rezept 6.6 beschrieben ist.

6.6 Welche Dateisystem-Typen muss ich in `start.conf` angeben? (`FSType = ?`)

Grundsätzlich ist die Angabe eines speziellen Datesystem-Typs eigentlich nur bei der Cache-partition notwendig, da diese als einzige von LINBO selbst benötihgt wird. Alle Betriebssystem-Partitionen werden durch das komplette Überschreiben mit den als `cloop`-Archiv geseicherten Partitionsinhalten automatisch beim ersten „Sync" formatiert. Zur Beschleinigung der Erkennung, ob ein Datei-Sync oder partitionsweises Überschreiben notwendig ist, formatiert LINBO jedoch auch, sofern ein Dateisystemtyp („File System Type", **FSType = ...**) angegeben ist,

die Partitionen schon beim ersten Einrichten mit dem gewählten Dateisystem. So erkennen auch die installierten Betriebssysteme, dass es sich um Datenpartitionen handelt. Windows kennt allerdings kaum Linux-eigene Dateisysteme, und bietet mitunter an, die „ungenutzten Bereiche“ der Festplatte zu formatieren und für Windows nutzbar zu machen, was ein kundiger Windows-Administrator unterbinden sollte.

Folgende Dateisystem-Typen sind unter Linux und Windows gebräuchlich:

Dateisystem	Beschreibung
ext2	Linux-tyisches Dateisystem, unterstützt alle für Unix-Systeme typischen Dateitypen und Dateirechte.
ext3	Wie **ext2**, und auch kompatibel dazu, aber mit zusätzlichem „Journal“ für die beschleunigte Reparatur bei einem Crash. Im Gegensatz zu **reiserfs** wird **ext2** beim Mounten im Fehlerfall nicht automatisch repariert, sondern muss mit **fsck.ext3** vom gestarteten Betriebssystem geprüft und ggf. repariert werden.
ntfs	„New Technology File System“, wird von neueren Windows-Versionen verwendet. Es unterstützt viele der von Unix/Linux her bekannten Features wie lange Dateinamen, internationale Zeichensätze in Dateinamen, Verknüpfungen, erweiterte Dateirechte und Dateien über 4 GB Größe.
reiserfs	Reiserfs merkt sich in einem „Journal“ die zuletzt durchgeführten Änderungen, und stellt nach einem Systemausfall beim Einbinden des Dateisystems den letzten konsistenten Zustand automtisch wieder her. Es ist durch Verwendung von balancierten Suchbäumen sehr schnell beim Zugriff auf viele kleine Dateien und verschachtelte Verzeichnisse, aber reagiert empflindlicher auf Festplattendefekte als `ext2` oder `ext3`.
swap	Kein Dateisystem, sondern eine Kennung für den Linux-Kernel, damit die Partition als Auslagerungsbereich für gerade nicht benötigte Teile des Hauptspeichers genutzt werden kann. Das Programm **swapon** aktiviert im gestarteten Linux-Betriebssystem eine mit **mkswap** (oder von LINBO) formatierte Partition.

vfat	FAT32, das auf neuen Flashdisks und Festplatten üblicherweise vorformatierte Dateisystem, unterstützt Dateien mit maximal 4GB, keine Dateirechte und keine UNIX-typischen Dateien wie Devices, Fifos, Symlinks oder ähnliches. Auch die Dateinamen unterliegen gewissen Einschränkungen. Wird von Windows bis 95 und DOS verwendet. Für die Cache-Partition nur dann brauchbar, wenn alle komprimierten Betribssystem-Images kleiner als 4GB sind.

Tabelle 7: Übersicht Dateisysteme

6.7 Hilfe, es bootet nicht!

Grundsätzlich verschieden, wenn auch nicht unbedingt voneinander unabhängig, spielen 2 Dinge eine Rolle, wenn trotz ansonsten korrekter Konfiguration von LINBO ein Start des Rechners fehlschlägt:

1. LINBO selbst (d.h. der Kernel **linbo**, oder
2. das zu startende Betriebssystem.

6.7.1 LINBO startet nicht / bleibt stehen

Wenn per PXE gebootet wird, und statt des Bootloaders eine Fehlermeldung ähnlich „`File not found.`“ erscheint, dann ist die Konfiguration des PXE-Bootladers nicht korrekt, oder es kann vom Rechner aus nicht per TFTP auf die benötigten Startdateien des Servers zugegriffen werden. Einige TFTP-Server verwegern den Zugriff auf Dateien, die per *Symlink* („Verknüpfung“) in das Bootverzeichnis gelegt wurden, wenn sie sich tatsächlich außerhab dieses Verzeichnisses befinden. Dies kann ein Sicherheitsfeature des TFTP-Servers sein, um zu verhindern, dass das komplette Dateisystem durch einen falschen Symlink auf ein in der Hierarchie sehr weit vorne liegendes Verzeichnis (z.B. **/**) nach außen exportiert wird. Weiterhin kann in der **dhcpd.conf** oder **pxelinux.cfg/*** ein falscher Dateiname oder Dateipfad für die beiden wesentlichen Dateien **linbo** und **linbofs.gz** angegeben sein, dann werden diese vom Bootloader selbst nicht gefunden. Linux-Dateisysteme sind groß-/kleinschreibungs-sensitiv, auch dies muss beachtet werden.

Wird hingegen der LINBO-Kernel geladen, aber danach bleibt der Bildschirm schwarz, dann ist höchstwahrscheinlich während der Hardware-Initialisierung ein Fehler aufgetreten. Der von LINBO verwendete Linux-Kernel enthält alle zum Start und zum Ansprechen der Hardware benötigten Module („Treiber“ im Windows-Jargon), diese werden nacheinander ausprobiert und versetzen die angesprochenen Hardwarekomponenten in einen funktionsfähigen Zustand. Leider sind bei den vielen verschiedenen Mainboards und CPUs hin und wieder Implementationsfehler vorhanden (d.h. Teile funktionieren gar nicht, oder werden vom Rechner-BIOS schon von vornherein falsch angesteuert), was auf Software-Seite durch Umgehung der nicht korrekt funktionierenden Komponenten „ausgetrickst“ werden kann. Unter Windows erledigen das sogenannte „Board-Treiber“, die für jeden Rechner speziell angeboten werden. Unter Linux sind die sogenannten „Kernel-Bootoptionen“ zuständig, bestimmte Hardwarekomponenten einfach nicht zu nutzen. Dabei werden diese nicht explizit dauerhaft „abgeschaltet“, sondern lediglich nicht weiter benutzt, und sind nach einem Reset des Rechners wieder so verfügbar, wie sie es vor dem Start von Linux auch waren. Auch die von LINBO gestarteten Betriebssysteme müssen sich nicht an die vom LINBO-Kernel zuvor „deaktivierten“ Komponenten halten. Allerdings kann es z.B. für Windows eine Rolle spielen, ob sich bestimmte Komponenten beim Start in einem Grubndzustand befinden, oder bereits einmal benutzt wurden. S.a. Rezept 6.7.2.

Die Optionen in nachfolgender Tabelle sind entweder direkt in der Konfogurationsdatei des Bootloaders anzugeben oder können, wenn interaktives Starten von LINBO erlaubt ist, nachträglich in der Bootkkonsole als Parameter nach **`linbo`** angegeben werden, z.B. für Tests. Es ist allerdings nicht möglich, einen fest in der Bootkonfiguration eingetragenen Parameter durch interaktive Eingabe wieder zu „löschen“, dies muss in der Konfigurationsdatei geschehen.

Beispiel für einen Eintrag im PXE-Bootloader,
Datei **`/var/linbo/pxelinux.cfg/default`**:

```
KERNEL linbo
APPEND initrd=linbofs.gz nosmp acpi=off
```

Hier wird angegeben, dass kein Symmetrisches Multiprozessing (SMP) verwendet wird, also nur ein Prozessor benutzt werden soll, und das „Advanced Configuration and Power Interface“ (ACPI) nicht verwendet wird, das bei modernen Computern auch die automatische Konfiguration von Erweiterungskarten anders als das BIOS dies vorgibt, erledigen kann.

Bootparameter	**Bedeutung**

acpi=off	Abschalten des „Advanced Configuration and Power Interface“, Interrupts werden über andere Mechanismen zugewiesen. Dies kann helfen, wenn sich bestimmte Hardwarekomponenten „aufhängen“, oder das System ohne erkennbaren Grund stehenbleibt. Allerdings funktionieren bei einigen Rechnern USB oder Netzwerkkarten nicht, wenn sie ausschließlich per ACPI initialisierbar sind.
acpi=noirq	Schaltet nur die Interrupt-Zuweisung durch ACPI ab. Powermanagement per ACPI bleibt aber eingeschaltet.
noapic	Schaltet den „Advanced Interrupt Controller“ auf dem Board ab, der auf neueren Boards das Interrupt-Handling übernimmt. Wenn der Rechner überhaupt nicht bis zud Hardware-Initialisierung kommt, ist der Grund manchmal ein falsch arbeitender APIC.
nolapic	Schaltet den lokalen „Advanced Interrupt Controller“ ab, der auf neueren CPUs das Interrupt-Handling übernimmt. Ähnlich **noapic**.
nolapic_timer	Schaltet nur den Zeitgeber des lokalen „Advanced Interrupt Controller“ ab. Nicht ganz so scharfe Form von **nolapic**.
nosmp	Benutzt nur die erste CPU bei Mehrprozessor-Systemen, und deaktiviert die Multiprozessor-spezifischen Verwaltungskomponenten. Für LINBO bedeutet dies keinen erheblichen Geschwindigkeitsverlust, da das Dekomprimieren der Daten auf Festplatte zwar theoretisch mit mehr CPUs schneller geht, aber die Schreibgeschwindigkeit der Festplatte der entscheidende Faktor bei der Restaurierungszeit ist, worauf die Rechengeschwindigkeit einer oder mehrere CPUs keinen Einfluss hat. Die durch LINBO gestarteten Betriebssysteme können von sich aus SMP wieder aktivieren, ohne dass ein Reset erforderlich ist.
pnpbios=off	Verhindert die Initialisierung von Komponenten durch das (eigentlich nur noch für ältere ISA-Karten vorhandene) „Plug & Play BIOS“.
pci=bios	Verwende ausschließlich die durch das Rechner-BIOS eingestellten Interrupt-Tabellen für die Konfiguration von PCI-Karten.

debug	Zeige beim Start des Kernels an, was vor sich geht, um auch die Fehlermeldungen zu sehen. Außerdem wird, sofern der Start bis kurz vor der graphischen Oberfläche noch klappte, eine keine interaktive Shell von LINBO gestartet, mit der sich Linux-Kenner im System „umschauen“ und ggf. **linbo_cmd** oder **linbo_gui** manuell starten können.

Tabelle 9: Die häufigsten „Workaround“-Bootparameter bei Bootproblemen

6.7.2 Das von LINBO gestartete Betriebssystem startet gar nicht / bleibt stehen

LINBO versucht beim Klick auf einen der „Start*“-Knöpfe, ggf. nach Synchronisation, das gewählte Betriebssystem direkt und ohne den sonst üblichen Reset des Rechners zu starten. Bei Linux ist dies via **kexec** möglich, bei Windows wird ein weiterer Bootloader, **grub.exe** dazwischengeschaltet, der wiederum über spezielle Bootoptionen erfährt, was er eigentlich starten soll.

Windows ist sehr empfindlich, was den Zustand der Hardware angeht, wenn der Windows-Kernel startet, was den Entwicklern schon bei den ersten Experimenten mit dem Direktboot auffiel.

Unter Linux werden die Hardwarekomponenten üblicherweise so konfiguriert, dass die die für den Betriebszustand optimalen Eigenschaften in der Priorität *Stabilität* und *Performanz* aufweisen. Bis auf wenige Ausnahmen sind alle Hrdwareparemeter zur Laufzeit wieder änderbar, und ein Betriebssystem sollte unabhängig vom IST-Zustand in der Lage sein, die Hardware auf seine eigenen favorisierten Einstellungen umzukonfigurieren. Leider ist das offenbar nicht in allen Fällen so. Während in der Virtualisierung unter „definitionsgemäß fehlerfreien“ Hardware-Kondtionen alles rund läuft, kommen in der Praxis Fehler durch spezielle, selten dokumentierte Eigenschaften der Hardware doch recht häufig vor. Manchmal helfen auch hier die im Rezept 6.7.1 genannten LINBO-Bootoptionen (Tabelle 9 Seite 34) weiter, auch wenn sie eigentlich keinen direkten Einfluss auf das gestartete Betriebssystem haben sollten, das die Hardware selbst noch einmal initialisiert.

Symptom: Beim Start von Windows kommt noch eine Meldung „`Launching GRUB ...\`, dann passiert aber nichts mehr.

Erklärung: `grub.exe` findet die Festplatte nicht, da der Festplattencontroller sich in einem Zustand befindet, in derm er im sog. „real mode“ der CPU nicht mehr ansprechbar ist.

Mögliche Lösung: Nach einem alternativen Betriebsmodus des Festplattencrontrollers im BIOS oder in den Linux-spezifischen Controller-Optioen schauen. Manchmal liegt es auch daran, dass die Interrupts anders verteilt wurden, als vom BIOS vorgegeben, dann könnten die Bootoptionen **`acpi=noirq`**, **`nosmp`**, **`nolapic`** oder **`pci=bios`** helfen, Tabelle 9 auf Seite 34 zeigt eine Übersicht. Manchmal sind es auch Optionen wie **`sata_nv.swncq=0`**, die sich nur auf einen speziellen (in diesem Fall SATA-) Controller beziehen, die den Controller in de gewünschten Zustand bringen, und somit das Booten ermöglichen. Diese Bootparameter sind zunächst einmal für LINBO selbst zu setzen, damit das neue Betriebssystem überhaupt einmal startet. Es ist aber nicht auszuschließen, dass, zumindest bei Linux, ähnliche Bootoptionen auch für das Betriebssystem selbst anzugeben sind. In diesem Fall werden sie unter "**`Append =`** " in **`start.conf`** beim entsprechenden Betriebssystem-Eintrag angegeben.

Symptom: Linux startet, aber der Bildschirm bleibt dunkel, bis die Grafikoberfläche erscheint.

Erklärung: LINBO benötigt für die graphische Oberfläche den Framebuffer-Modus. Dieser kann bei einigen Grafikkarten nicht ohne Reset des Rechners zurück auf den Textmodus geschaltet werden, daher ist die Nutzung der normalen Textkonsole, oder auch Umstellen auf einen Framebuffer-Modus anderer Auflösung oder Farbtiefe nicht möglich.

Mögliche Lösung: Das zu bootende System so umstellen, dass je nach Fehlerart entweder kein Bootsplash, oder ein bestimmter Framebuffer-Modus verwendet wird. Tabelle 10 zeigt einige beliebte Framebuffer-Modi und ihre Angabe als Bootparameter.

Symptom: Windows oder Linux starten, aber irgendetwas ist falsch (z.B. Maus geht nicht, Netzwerk ist nicht erreichbar). Direkt ohne LINBO von Platte gestartet, funktioniert alles.

Erklärung: Das Abschalten oder Umkonfigurieren von Hardwarekomponenten ist nach dem "Soft-Reset" immer noch aktiv, und das gestartete Betriebssystem ist verwirrt.

Mögliche Lösung: Für Linux als gestartetes Betriebssystem lässt sich durch explizite Angabe von z.B. **`acpi=force`** in der **`Append =`** Zeile erreichen, dass bestimmte Eigenschaften re-aktiviert werden. Für Windows ist

das nicht ohne weiteres möglich, hier müssten für LINBO Ersatz-Boot-Optionen gefunden werden, die das Booten ermöglchen, aber keine Auswirkungen auf das gestartete Betriebssystem haben. In einem Beispielszenario bewirkten sowohl die APPEND-Optionen **acpi=noirq** als auch **nosmp** in der **pxlinux.cfg/default** das einwandfreie Booten von Windows, allerdings funktionierte nur mit **nosmp** der USB-Controller. Mit **acpi=nboirq** wurden hingegen USB-Maus und USB-Tastatur nicht von LINBO erkannt.

6.8 Ich möchte die Auflösung ändern

LINBO verwendet den Framebuffer der Grafikkarte, um die Fenster seiner Oberfläche zu zeichnen. Die Auflösung hierfür wird beim Start des LINBO-Kernels über die Bootoption **vga=...** festgelegt, um die Grafikkarte in den gewünschten Modus zu schalten, indem ein Wert in das VESA-Register der Grafikkarte geschrieben wird. Eine vollständige Tabelle findet sich in den Kernel-Sourcen im Dokument **Documentation/fb/vesafb.txt**. Ein Auszug hiervon ist in Tabelle 10 wiedergegeben.

Bootoption	Framebuffer-Modus
vga=769	Auflösung 640x480 mit 256 Farben (8bit)
vga=785	Auflösung 640x480 mit 64k Farben (16bit)
vga=788	Auflösung 800x600 mit 64k Farben (16bit)
vga=791	Auflösung 1024x768 mit 64k Farben (16bit)

Tabelle 10: Auflösungen und Framebuffer-Modi

Eine weitere Möglichkeit, die Auflösung beim Start von LINBO zu setzen, ist der direkte Eintrag in die LINBO-Kerneldatei **linbo** mit dem Linux-Programm **rdev**:

```
rdev -v linbo 785
```

Hiermit wird auch bei fehlender **vga=**-Bootoption die Auflösung 640x480 mit 64.000 Farben (16bit) verwendet.