LINBO

Linux-basiertes Interaktives Netzwerk-Bootsystem

Klaus Knopper

10.05.2007

Inhaltsverzeichnis

Abbildungsverzeichnis

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

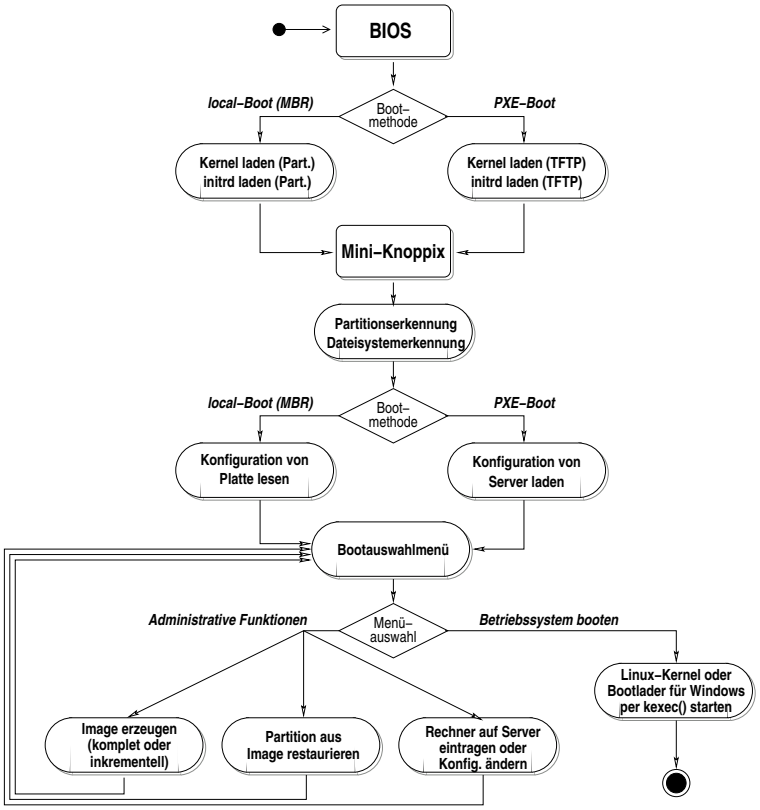

Abbildung 1: UML Aktivitätsdiagramm für LINBO

Vom Bootlader (lilo oder 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.)

Das zentrale **Makefile** im LINBO-Entwicklungsverzeichnis bietet drei Testszenarien mit Hilfe von **qemu** als virtuelle Machine(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

Für die zuletzt genannte Option ist die Installation von qemu ab Version 0.9.0+cvs erforderlich, da frühere Versionen den „bootp“-Parameter noch nicht kannten.

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 verwalteten Betriebssysteme lokal vorzuhalten, und auch ohne Netzwerk starten zu können.

1.3.2 Multicast

Der von LINBO verwendete TFTP-Client **atftp** zum Download von Images ist multicast-fähig, d.h. es ist möglich, dass mehrere Clients Daten von einem gemeinsamen Stream vom Server gleichzeitig beziehen, wodurch die Netzwerk-Auslastung gerade bei parallelen Installationen dramatisch reduziert werden kann. Hierfür ist allerdings ein ebenfalls multicast-fähiger TFTP-Server, z.B. **atftpd** erforderlich.

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, oder eine ensprechende Bootdiskette mit Treiber von **http://www.rom-o-matic.net**.

2 Installation

LINBO wird üblicherweise per PXE gebootet, und kann sich selbst auf die Cache-Partition (1.3.1) kopieren.

2.1 Bootvorgang

LINBO kann wie ein normaler Linux-Kernel gebootet werden, unabhängig ob von lokaler Festplattenpartition oder einem PXE/BOOTP-fähigen DHCP-Server.

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

[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

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 3.2.3.

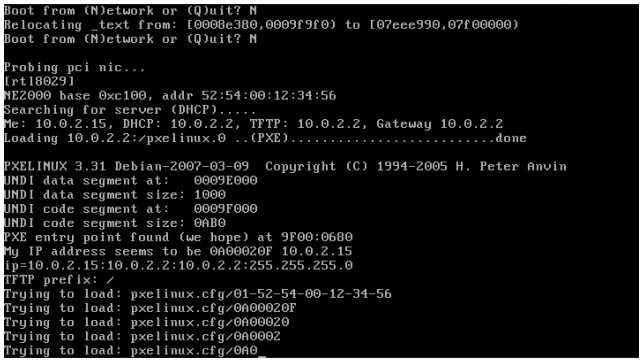

Abbildung 2: PXE-Bootlader in qemu

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

Abbildung 3: Start von LINBO über pxelinux

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 intelligent, dass es bei noch nicht erkannten Festplattenpartitionen einige Zeit wartet, bis diese verfügbar sind.

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 Festplate installiert wurden, aufgeführt.

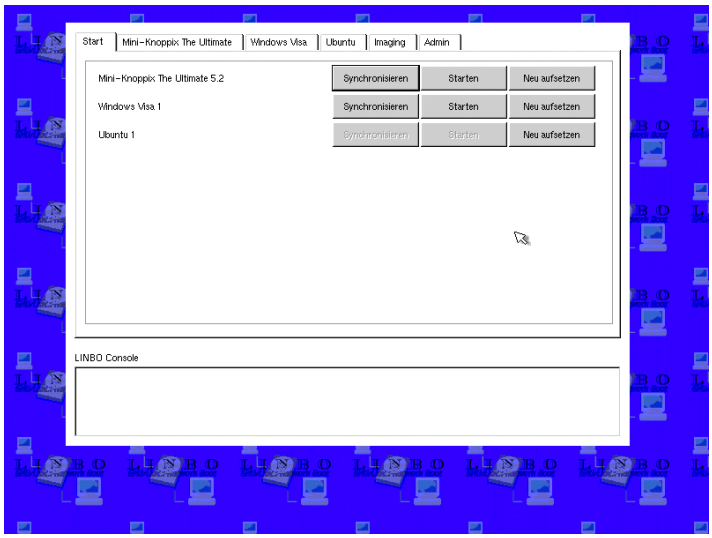

Abbildung 4: Startmenü von LINBO

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

`Neu aufsetzen` (Abbildung 5) lädt eine ggf. neuere Version des jeweiligen Betriebssstems vom Server per TFTP/Multicast auf die Cache-Partition herunter, und installiert diese anschließend wie bei **`Synchronisieren`**.

`Start` bootet das angegebene Betriebssystem so, wie es sich derzeit auf der Festplatte befindet. Der Rechner startet hierbei nicht neu, sondern LINBO führt einen

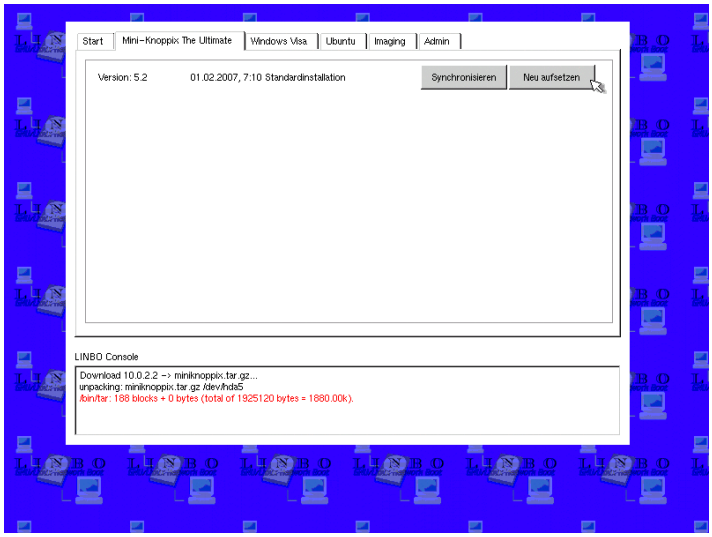

Abbildung 5: Neu aufsetzen eines Betriebssystems mit LINBO

„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 **`Synchronisieren`** oder **`Neu aufsetzen`** wieder der zuletzt gespeicherte, arbeitsfähige Zustand restauriert werden, bevor ein neuer **`Start`** versucht wird.

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

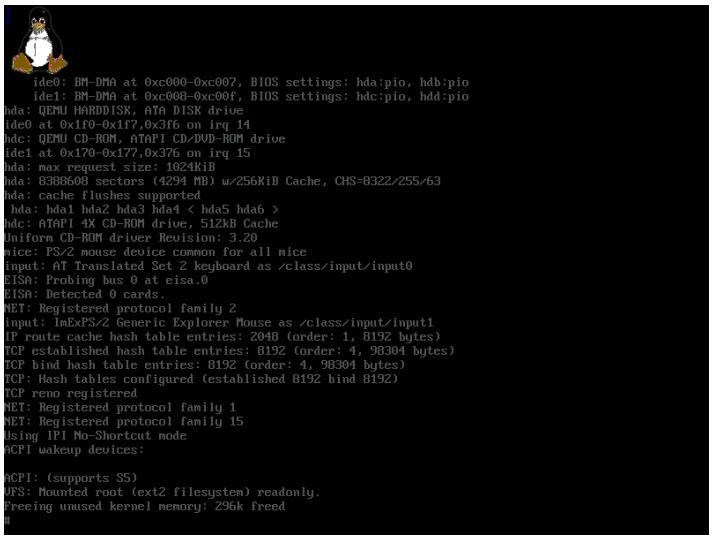

Abbildung 6: Aus LINBO gestartetes Mini-Linux

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 einem Basis-Image aufbauen.

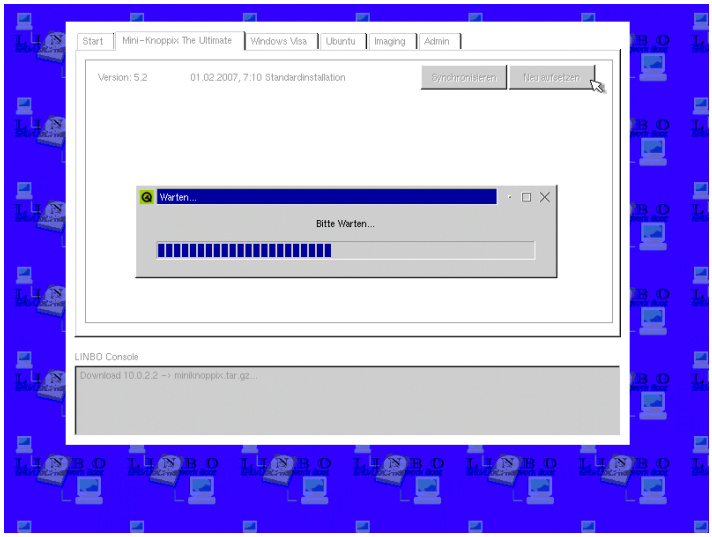

Abbildung 7: Neu aufsetzen - Dekompressionsvorgang

Gegenüber „Synchronisieren“ und „Neu aufsetzen“ aus dem Startmenü erlauben die gleichnamigen Buttons in den Betriebssystem-Reitern also eine genauere Angabe der jeweiligen Version, während im Startmenü immer nur die neuste Version restauriert wird.

3.2.3 Alles neu – Administrationsmenü

Um die Festplatte nach den Vorgaben des Administrators zu partitionieren, die Partitionen zu formatieren und sämtliche angebotenen Images auf der Cache-Partition abzulegen, sowie LINBO bootfähig auf Festplatte zu installieren, ist der Reiter **`Admin`** und die darunter vorhandenen Buttons zuständig.

Bitte beachten Sie, dass beim Neupartitionieren und Formatieren der Festplatte sämtliche dort gespeicherten Daten verloren gehen.

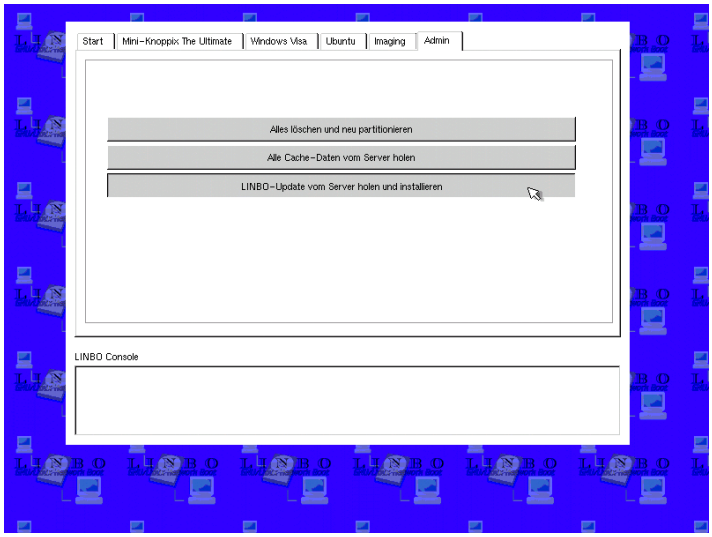

Abbildung 8: Admin-Menü

4 Administration

4.1 Konfiguration

4.1.1 `start.conf` - Partitionen und Images

Die im **`linbofs.gz`** für jeden Client-Rechner befindliche Datei **`start.conf`** ist im Stil der bekannten KDE-Desktop-Iconbescreibungen verfasst. Kommentare werden durch **`#`** eingeleitet, dürfen am Anfang einer Zeile oder mitten im Text aufauchen, und werden inklusive bis zum Zeilenende folgendem ext von **`linbo_gui`** ignoriert.

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

```
[LINBO]                 # LINBO global config
Cache = /dev/hda2       # Cache Partition with local Images
Server = 10.0.2.2       # First TFTP Server with remote Images

[Partition]             # Start of a Partition config section
Dev = /dev/hda1         # Device name of partition (hda1 = first partition on first IDE disk)
Size = 100000           # Partition size in kB
Id = 6                  # Partition type (83 = Linux, 82 = swap, c = FAT32, 6 = NTFS, ...)
FSType = ntfs           # File system on Partition
Bootable = yes          # Mark this partition as bootable

[Partition]
Dev = /dev/hda2         # Device name of partition
Size = 100000           # 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 (or Linux)

[Partition]
Dev = /dev/hda3         # Device name of partition
Size = 512              # Partition size in kB
Id = 82                 # Partition type (83 = Linux, 82 = swap, c = FAT32, ...)
FSType = swap           # File system on Partition

[Partition]
Dev = /dev/hda4         # 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/hda5         # Device name of partition
Size = 10000            # Partition size in kB
Id = 83                 # Partition type (83 = Linux, 82 = swap, c = FAT32, ...)
FSType = ext2           # File system on Partition
Bootable = no           # Mark this partition as non-bootable (or Linux)

[Partition]
Dev = /dev/hda6         # Device name of partition
Size = 500000           # 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 (or Linux)

[OS]
Name = Mini-Knoppix The Ultimate          # Name of OS
Version = 5.2           # Version/Date of OS (optional)
Description = 01.02.2007, 7:10 Standardinstallation # Descriptive Text
Image = miniknoppix.tar.gz # Filename of (incremental?) tar archive or partition image
BaseImage  = # Filename of base tar archive or base partition image (optional)
Timestamp = 200705102307 # Timestamp of image creation
Boot = /dev/hda5        # Partition containing Kernel & Initrd
Root = /dev/hda5        # root=/dev/partition Parameter (Root FS)
Kernel = vmlinuz        # Relative filename of Kernel or Boot image
Initrd =                # Relative filename of Initrd
Append =                # Kernel cmdline, root= will be added (optional)
StartEnabled = yes      # Enable "Start" Button
SyncEnabled = yes       # Enable "Synchronize" Button
RemoteSyncEnabled = yes # Enable "Synchronize from Server" Button

[OS]
Name = Windows Visa     # Name of OS
Description = 06.02.2007, 10:10 Es bootet.                        # Descriptive Text
Version = 1             # Version/Date of OS (optional)
Image = visa-20070206.tar.gz # Filename of (incremental?) tar archive or partition image
BaseImage  = visa.img.gz # Filename of base tar archive or base partition image (optional)
Timestamp = 200705102307 # Timestamp of image creation
Boot = /dev/hda1        # Partition containing Kernel & Initrd
Root = /dev/hda1        # root=/dev/partition Parameter (Root FS)
Kernel = grub.exe       # Relative filename of Kernel or Boot image
Initrd =                # Relative filename of Initrd
Append = --config-file=map(rd) (hd0,0); map --hook; chainloader (hd0,0)+1; rootnoverify(hd0,0) --device
StartEnabled = yes      # Enable "Start" Button
SyncEnabled = yes       # Enable "Synchronize" Button
RemoteSyncEnabled = yes # Enable "Synchronize from Server" Button
```

```
[OS]
Name = Ubuntu           # Name of OS
Version = 1             # Version/Date of OS (optional)
Description = 01.02.2007, 7:10 Standardinstallation # Descriptive Text
Image = ubuntu-1.tar.gz  # Filename of (incremental?) tar archive or partition image
BaseImage  = ubuntu.tar.gz # Filename of base tar archive or base partition image (optional)
Timestamp = 200705102307 # Timestamp of image creation
Boot = /dev/hda6        # Partition containing Kernel & Initrd
Root = /dev/hda6        # root=/dev/partition Parameter (Root FS)
Kernel = boot/vmlinuz   # Relative filename of Kernel or Boot image
Initrd = boot/initrd.gz # Relative filename of Initrd
Append =                # Kernel cmdline, root= will be added (optional)
StartEnabled = no       # Enable "Start" Button
SyncEnabled = no        # Enable "Synchronize" Button
RemoteSyncEnabled = yes # Enable "Synchronize from Server" Button

[OS]
Name = Ubuntu           # Name of OS
Version = 2             # Version/Date of OS (optional)
Description = 02.02.2007, 7:10 Standardinstallation mit xemacs   # Descriptive Text
Image = ubuntu-2.tar.gz  # Filename of (incremental?) tar archive or partition image
BaseImage  = ubuntu.tar.gz # Filename of base tar archive or base partition image (optional)
Timestamp = 200705110201 # Timestamp of image creation
Boot = /dev/hda6        # Partition containing Kernel & Initrd
Root = /dev/hda6        # root=/dev/partition Parameter (Root FS)
Kernel = boot/vmlinuz   # Relative filename of Kernel or Boot image
Initrd = boot/initrd.gz # Relative filename of Initrd
Append =                # Kernel cmdline, root= will be added (optional)
StartEnabled = yes      # Enable "Start" Button
SyncEnabled = yes       # Enable "Synchronize" Button
RemoteSyncEnabled = yes # Enable "New from Server" Button
```

Auffällig dürfte sein, dass im **[OS]**-Abschnitt Namen von Betriebssystemen mehrfach genannt werden dürfen, wobei die nachfolgenden Einstellungen und Image-Namen dann als Versionspaket dieses Betriebssystems interpretiert werden, und in den einzelnen Reitern für die Betriebssysteme im **linbo_gui** als Subversionen auftauchen.

4.1.2 PXE-Konfiguration (DHCP-Server)

LINBO-Kernel, LINBO-Dateisystem und LINBO-Images 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 **atftpd** könnte dementsprechend auf dem Server wie folgt gestartet werden:

```
sudo atftpd -daemon --port 69 --retry-timeout 5 \
     --mcast-port 1758 --mcast-addr 239.239.239.0-255 \
     --mcast-ttl 1 --maxthread 100 --verbose=5 \
     /tftpboot
```

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.[2]

Beispiel für einen entsprechenden Abschnit 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 für Testzwecke mit

```
atftp -r /linbo -l linbo 10.0.2.2
```

getestet werden, ob der LINBO-Kern über den TFTP-Server erreichbar ist, zumal ohne führenden **/** der TFTP-Server mitunter mit einem Fehler antwortet, statt die im TFTP-Exportverzeichnis liegenden Dateien zu liefern. V.a. der qemu-interne TFTP-Server zeigt dieses Verhalten.

[2] Später wird diese Konfiguration ein graphisches Verwaltungstool mit übernehmen helfen.

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
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.

Documentation wird später das Benutzer- und Administrationshandbuch von LINBO enthalten (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 ; apt-get 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 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. ;-)
```