LTSP − Linux Terminal Server Project − v3.0

James McQuillan

jam@LTSP.org

Martin Herweg

m.herweg@gmx.de

Wolfgang Schweer

schweer@cityweb.de

Versionsgeschichte

Version 1.0.4	2002−03−02	Geändert durch: jam
Version 1.0.4−de	2. März 2002	Geändert durch: ws

GNU/Linux bietet eine gute Grundlage für den Einsatz von plattenlosen "thin clients". In erster Linie soll dieses Dokument zeigen, wie plattenlose "thin clients" mit LTSP eingerichtet und betrieben werden können. Es werden allerdings auch viele Aspekte zum Thema plattenlose Arbeitsplatzrechner im Allgemeinen behandelt.

Inhaltsverzeichnis

Inhaltsverzeichnis

Einführung

Das LTS−Projekt ermöglicht es, auf einfache Weise preisgünstige Arbeitsplatzrechner als Terminals eines GNU/Linux−Servers zu benutzen − mit einer graphischen Oberfläche oder textbasiert.

In einer klassischen Büroumgebung finden sich auf jedem Schreibtisch relativ gut ausgestattete PCs, jeder mit eigener Festplatte von mehreren Gigabytes. Die Benutzer speichern ihre Daten lokal, Backups werden selten bis nie durchgeführt.

Macht es wirklich Sinn, auf jedem Schreibtisch solch einen Rechner stehen zu haben?

Unsere Antwort lautet: Nein.

Glücklicherweise gibt es eine Alternative. Wenn man LTSP benutzt, dann reichen PCs am unteren Ende der Preisskala − Festplatte, Diskettenlaufwerk und CDROM−Laufwerk sind überflüssig. Lediglich eine Netzwerkkarte, von der der Rechner gebootet werden kann, ist notwendig. Viele Netzwerkkarten haben einen Bootrom−Sockel.

Während der Bootphase erhält der plattenlose Client seine IP−Adresse, weitere Informationen und den Kernel vom Server. Danach wird das Root−Verzeichnis vom Server per NFS gemountet.

Der Client kann auf drei Weisen konfiguriert werden:

- **Graphische Oberfläche − X Window**

  Unter dem X Window System kann der Arbeitsplatzrechner alle Anwendungsprogramme auf dem Server oder auf anderen Servern im Netzwerk benutzen.

- **Textbasiert: Telnet−Sitzungen**

  Der Arbeitsplatzrechner kann bis zu 9 Telnet−Sitzungen zum Server aufbauen. Jede Sitzung läuft auf einer virtuellen Konsole. Mit ALT−F1 bis ALT−F9 kann zwischen den einzelnen Konsolen hin− und her geschaltet werden.

- **Textbasiert: Shell−Modus**

  Der Client hat nach Voreinstellung auf den beiden ersten Konsolen eine bash. Man ist bereits als root angemeldet − nützlich, wenn es Probleme mit dem Start der graphischen Oberfläche gibt oder etwas mit NFS nicht klappt.

Richtig gut ist es, dass man mit einem einzigen GNU/Linux−Server eine ganze Menge von Arbeitsplatzrechnern bedienen kann. Wie viele das sein können, hängt von der Art der Anwendungen auf den Clients und der Leistungsfähigkeit des Servers ab.

Es ist nicht ungewöhnlich, 40 Arbeitsplatzrechner mit jeweils Netscape und StarOffice über einen Dual PIII−650 mit 1GB RAM zu betreiben. Wir wissen, dass das geht. In der Tat liegt der load−average Wert selten über 1.0!

---

1. Haftungsausschluss

Weder der Autor noch die Übersetzer, Distributoren oder Mitautoren dieses Dokumentes sind in irgendeiner Weise verantwortlich zu machen für physikalische, finanzielle, moralische oder irgendwelche anderen Schäden, die infolge der Vorschläge in diesem Text eintreten.

2. Copyright und Lizenz

Kapitel 1. Die Theorie

Das Booten eines plattenlosen Netzwerkrechners umfasst mehrere Stufen. Wenn man den Ablauf kennt, ist es viel einfacher, möglicherweise auftretende Probleme zu lösen.

Der im weiteren beschriebene Ablauf beruht auf folgender Konfiguration:

- Üblicher Standard−Rechner
- Linksys LNE100TX Netzwerkkarte mit Etherboot −Bootrom
- Intel i810 Graphik Chipset
- Server mit Redhat 7.2
- DHCP
- Netzwerk−Adressen 192.168.0.0/24

Wenn der Server mit Hilfe von LTSP konfiguriert wurde, passiert auf dem Client folgendes:

1. Nach dem Einschalten läuft der "Power On Self Test" (POST) ab.

2. Während des Selbsttests sucht das Bios nach zusätzlichem ROM. Das Etherboot Bootrom auf der Netzwerkkarte wird vom Bios als solch ein zusätzlich vorhandenes ROM erkannt.

3. Sobald der Selbsttest beendet ist, geht die Abarbeitung an den Etherboot Code über.

4. Der Etherboot Code sucht nach einer Netzwerkkarte. Sobald diese entdeckt wird, erfolgt deren Initialisierung.

5. Der Etherboot Code schickt dann per Broadcast eine DHCP−Anfrage ins lokale Netz. Dabei wird die MAC−Adresse der Netzwerkkarte mitverschickt.

6. Der auf dem Server laufende dhcpd−Prozess nimmt die Anfrage des Clients entgegen und sieht in seiner Konfigurationsdatei nach, ob dort für diese MAC−Adresse ein Eintrag vorhanden ist.

7. Der DHCP−Server schickt bei vorhandenem Eintrag ein Antwort−Paket, das mehrere Informationen für den Client enthält:

    - Die IP−Adresse des Clients
    - Die NETMASK für das lokale Netz.
    - Den Namen des Kernels, der auf dem Client gestartet werden soll (inclusive Pfadnamen)
    - Den Pfadnamen des zu mountenden Root−Dateisystems
    - Optionale Kommandozeilen−Parameter für den Kernel

8. Der Etherboot Code konfiguriert dann das TCP/IP−Interface der Netzwerkkarte entsprechend.

9. Danach fordert der Etherboot Code beim Server den Download des Kernels an. Dazu wird das Protokoll TFTP (Trivial File Transfer Protocol) verwendet.

10. Nach dem vollständigen Download des Kernels legt der Etherboot Code diesen an der richtigen Stelle im Hauptspeicher ab.

11. Die Kontrolle geht dann an den Kernel über. Dieser initialisiert das gesamte System inclusive aller erkannter Hardware.

12. Jetzt wird es richtig spannend. An den Kernel angehängt ist nämlich ein Image eines Dateisystems. Dieses wird als Ramdisk in den Hauptspeicher geladen und vorübergehend als Root−Dateisystem gemountet. Dazu wird dem Kernel der Kommandozeilen−Parameter **root=/dev/ram0** mitgegeben.

13. Normalerweise führt der Kernel nach Beenden des Bootens den **init** Prozess aus. Hier wird das anders gemacht: Mit dem Kommandozeilen−Parameter **init=/linuxrc** weisen wir den Kernel an, ein shell−Script auszuführen.

14. Das **/linuxrc** Script scannt zunächst den PCI−Bus nach einer Netzwerkkarte. Für jedes gefundene PCI−Device erfolgt ein Vergleich mit der Datei bekannter Netzwerkkarten (/etc/niclist). Falls dort ein Eintrag gefunden wird, wird das entsprechende Kernel−Modul geladen. Für ISA−Karten muss der Name des Moduls dem Kernel als Kommandozeilen−Parameter übergeben werden. Zusätzlich sind noch eventuell erforderliche Werte für IO−Adresse und IRQ zu übergeben.

15. Wenn die Werte stimmen, wird die Karte vom Kernel−Modul identifiziert und das Modul geladen.

16. Danach wird der Prozess **dhclient** gestartet, um erneut eine Anfrage an den DHCP−Server zu stellen. Diese zweite Anforderung von Daten, diesmal im user−space, ist aus zwei Gründen notwendig: Einmal wird die durch den Etherboot Code erhaltene Antwort vom Kernel 'verschluckt' und zweitens ignoriert der Kernel die Angabe eines möglicherweise in der root−path Option spezifizierten NFS−Servers. Dies ist von Bedeutung, falls der NFS−Server und der TFTP−Server unterschiedlich sein sollen.

17. Wenn der **dhclient** eine Antwort vom Server bekommt, führt er das **/etc/dhclient−script** aus und konfiguriert das Netzwerk−Interface eth0 entsprechend.

18. Bis zu diesem Zeitpunkt liegt das root−Dateisystem auf einer Ram−Disk. Nun mountet das /linuxrc−Script ein neues root−Dateisystem per NFS. Typischerweise wird dies **/opt/ltsp/i386** auf dem NFS−Server sein. Das neue root−Dateisystem kann nicht sofort von /linuxrc als / gemountet werden sondern zun]chst als /mnt. Danach wird vom Script das Kommando **pivot_root** ausgeführt. pivot_root mountet das alte System auf /oldroot und macht damit Platz für das neue, per NFS gemountete System, was dann unter / zur Verfügung steht.

19. Nach Mounten und Verschieben (pivoting) des neuen root−Dateisystems ist das /linuxrc−Script abgearbeitet und das eigentliche **init** Programm kann gestartet werden.

20. Der init−Prozess liest die Datei `/etc/inittab` und setzt die Systemumgebung entsprechend.

21. Init ermöglicht sogenannte *runlevel*. Jedes runlevel definiert eine Menge von Diensten für den Client. Der LTSP−Client startet im runlevel '2'. Dies wird durch die *initdefault* Zeile in der Datei inittab festgelegt.

22. Einer der ersten Eintr]ge in der inittab−Datei ist das Script **rc.local**, das während der '***sysinit***' Phase des Clients abläuft.

23. Das rc.local Script legt eine Ram−Disk von 1 Mb an. Diese wird alles enthalten, was geschrieben oder in irgendeiner Weise verändert werden muss.

24. Die Ram−Disk wird als Verzeichnis `/tmp` gemountet. Alle Dateien, die Schreibrecht gesetzt haben müssen, sind tatsächlich nur hier im Verzeichnis anzusiedeln. Symbolisch Links verweisen von der notwendigen Position aus dann hierhin.

25. Dann wird das `/proc` Dateisystem gemountet.

26. Falls für den Client Swapping über NFS eingestellt wurde, dann wird nun das Verzeichnis **/var/opt/ltsp/swapfiles** als /tmp/swapfiles gemonutet. Falls noch kein Swapfile für diesen Client vorhanden ist, wird nun einer angelegt; die Größe des Swapfiles wird in in der Konfigurationsdatei `lts.conf` festgelegt.

    Durch das **swapon** Kommando wird der Swapfile aktiviert.

27. Das Netzwerk−Device ***loopback*** wird konfiguriert. Dies hat *127.0.0.1* als IP−Adresse.

28. Wenn der Client per Konfigurationsdatei so eingerichtet wurde, dass Programme auf dem Client selbst ausgeführt werden (local apps), dann wird nun **/home** vom Server auf /home lokal gemountet, so dass die Programme Zugriff auf das Home−Verzeichnis des Benutzers haben.

29. Etliche Verzeichnisse werden nun im `/tmp` Verzeichnis angelegt, damit dort Dateien abgelegt werden können, die während der Laufzeit des Client−Rechners notwendig sind. Es handelt sich um folgende Verzeichnisse:

    a. `/tmp/compiled`

    b. `/tmp/var`

    c. `/tmp/var/run`

    d. `/tmp/var/log`

    e. `/tmp/var/lock`

    f. `/tmp/var/lock/subsys`

    Alle sind damit verfügbar.

30. Nun wird das X Window System konfiguriert. In der Konfigurationsdatei **lts.conf** gibt es den Parameter **XSERVER**. Fehlt dieser Parameter oder ist er auf "**auto**" gesetzt, dann wird versucht, den passenden Graphiktreiber automatisch zu bestimmen. Für eine PCI−Karte lassen sich nämlich die Angaben "PCI Vendor" und "Device id" feststellen. Diese werden mit den Einträgen in der Datei **/etc/vidlist** verglichen.

    Wenn die Karte von XFree86 4.x unterstützt wird und in der Liste ein Eintrag vorhanden ist, dann wird der Name des Treiber−Moduls an das Script zurückgegeben. Falls in der Konfigurationsdatei lts.conf für eine nur von XFree86 3.3.6 unterstützte Karte der entsprechende X−Server (Namen, die mit 'XF86_' beginnen) angegeben wurde, dann wird dieser Wert zurückgegeben. Damit ist klar, welcher X−Server zu starten ist.

31. Falls Version 4.x verwendbar ist, dann wird nun mittels des Skripts **/etc/rc.setupx** die Konfigurationsdatei XF86Config erzeugt. Für XFree86 3.3.6 übernimmt diese Aufgabe das Script

/etc/rc.setupx3.

Zum Erzeugen dieser Konfigurationsdatei XF86Config werden die Einträge in der Datei **/etc/lts.conf** benutzt.

32. Nachdem XF86Config angelegt wurde, geht die Kontrolle zurück an das rc.local Script. Das Script **/tmp/start_ws** wird erzeugt. Mit Hilfe dieses Skripts kann der passende X−Server gestartet werden.

33. Die Datei `/tmp/syslog.conf` wird angelegt. Diese enthält Informationen für den **syslogd** Prozess, an welchen Rechner im Netzwerk Systemmeldungen geschickt werden sollen. Dieser Rechnername ist in der Datei `lts.conf` anzugeben. Der symbolische Link `/etc/syslog.conf` zeigt auf die in der Ram/Disk liegende Datei `/tmp/syslog.conf`.

34. Nun kann der Hintergrundprozess (daemon) **syslogd** gestartet werden.

35. Die Steuerung geht wieder an den **init** zurück. Dieser entscheidet anhand des ***initdefault*** Eintrages, welches ***runlevel*** benutzt werden soll; seit lts_core−2.08, lautet der Wert ***2***.

36. In runlevel ***2*** führt der init−Prozess das Script **set_runlevel** aus; dieses liest die Konfigurationsdatei lts.conf ein und legt aufgrund des Eintrages f@r den Client fest, welches aktuelle runlevel zu benutzen ist.

    Die vorhandenen runlevel sind ***3***, ***4*** und ***5***.

    - ***3*** − Startet eine shell (bash). Nützlich zum Debuggen: Wenn der Client nicht so will, wie er soll.
    - ***4*** − Startet eine oder mehrere telnet−Sitzungen (textbasiert). Geeignet, wenn alte serielle Terminals auf einfache Weise ersetzt werden sollen.
    - ***5*** − Graphischer Modus. X wird gestartet, eine XDMCP−Anfrage geht an den Server, der dann seinerseits ein Anmelde−Fenster schickt. Dazu muss auf dem Server ein Display−Manager laufen, wie z.B. ***XDM***, ***GDM*** oder ***KDM***.

Kapitel 2. LTSP auf dem Server installieren

Die LTSP−Pakete gibt es im ***RPM*** und ***TGZ*** Format. Je nach Wahl folge man den Beschreibungen in den entsprechenden Abschnitten

Wenn auf den Clients das X Window System laufen soll, müssen vier Pakete installiert werden. Man beachte, dass in diesem Beispiel von einem Client−Rechner mit einer Tulip−Chip basierten Netzwerkkarte sowie einer Graphikkarte mit Intel i810 Chipsatz ausgegangen wird.

1. Das Paket LTSP Core
2. Das Paket Kernel
3. Das Paket X Core
4. Das Paket X Fonts

Das letzte Paket wird eigentlich nicht benötigt, für eine Erstinstallation jedoch empfolen. Nachdem Server und Clients laufen, kann man einen X Font Server (XFS) einrichten, der dann den Job übernimmt.

Nach Installation der Pakete muss das LTSP System initialisiert werden. Dabei werden unter anderem Änderungen an an Konfiguratoinsdateien auf dem Server vorgenommen. Dies ist notwendig, damit der Server den Clients die erforderlichen Dienste zur Verfügung stellen kann.

2.1. RPM−Pakete installieren

Laden Sie die neuesten ltsp−Pakete herunter und installieren Sie diese mit den folgenden Kommandos auf Ihrem Server:

```
rpm −ivh ltsp_core−3.0.0−1.i386.rpm

rpm −ivh ltsp_kernel−3.0.1−1.i386.rpm

rpm −ivh ltsp_x_core−3.0.1−1.i386.rpm

rpm −ivh ltsp_x_fonts−3.0.0−0.i386.rpm
```

Die Installation erfolgt damit im Verzeichnis `/opt/ltsp/i386`.

2.2. TGZ−Pakete installieren

Laden Sie die neuesten ltsp−Pakete herunter und installieren Sie diese mit den folgenden Kommandos auf Ihrem Server:

```
tar xzf ltsp_core−3.0.0−i386.tgz
cd ltsp_core
./install.sh
cd ..

tar xzf ltsp_kernel−3.0.1−i386.tgz
cd ltsp_kernel
./install.sh
cd ..
```

```
tar xzf ltsp_x_core-3.0.1-i386.tgz
cd ltsp_x_core
./install.sh
cd ..

tar xzf ltsp_x_fonts-3.0.0-i386.tgz
cd ltsp_x_fonts
./install.sh
cd ..
```

Die Installation erfolgt damit im Verzeichnis `/opt/ltsp/i386`. Achten Sie darauf, alle vier Pakete in einem Durchgang zu installieren.

2.3. Initialisieren des Servers

Nachdem die obigen Pakete installiert sind, wechseln Sie bitte in das Verzeichnis `/opt/ltsp/templates`. Dort finden Sie Dateien, die die Konfiguration Ihres Servers verändern werden. Für jede auf dem Server zu modifizierende Datei liegt hier ein Muster vor. Sehen Sie sich den Inhalt der Dateien an und vergewissern Sie sich, dass Sie die entsprechenden Änderungen vornehmen wollen. Die Änderungen könnten die Sicherheit Ihres Systems gefährden. Die Änderungen können Sie per Hand vornehmen − oder automatisch: Starten Sie dazu das Script ltsp_initialize:

```
cd /opt/ltsp/templates
./ltsp_initialize
```

Sie erhalten jeweils eine Abfrage, ob ein Dienst konfiguriert werden soll. Es geht dabei um folgende Dienste:

- XDM − der X Display Manager
- GDM − der Gnome Display−Manager
- Startscript des Display Managers
- bootp
- Die Datei /etc/exports
- tcpwrapper
- portmapper
- syslogd
- TFTP Start−Script

2.4. Konfiguration der Clients

Auf dem Server sind die Informationen zu den einzelnen Clients in drei Dateien enthalten:

1. /etc/dhcpd.conf
2. /etc/hosts
3. /opt/ltsp/i386/etc/lts.conf

2.4.1. /etc/dhcpd.conf

Der Client braucht eine IP−Adresse und weitere Informationen. Er bekommt folgendes vom DHCP−Server:

- IP−Adresse

- hostname
- Server IP
- Default gateway
- Name des zu ladenden Kernels incl. Pfadangabe
- Servername und Verzeichnis auf dem Server, das als root Dateisystem gemountet werden soll

In unserem Beispiel weisen wir per DHCP−Server den Clients feste IP−Adressen zu.

Das ltsp_initialize−Script legt ein Muster namens `/etc/dhcpd.conf.example` an. Sie können dieses nach /etc kopieren als `/etc/dhcpd.conf` und diese Datei dann als Basis benutzen. Selbstverständlich müssen Sie Anpassungen an Ihre eigene Systemumgebung vornehmen und die Angaben zu den Clients anpassen, insbesondere die MAC−Adresse unter 'hardware' eines jeden Clients.

```
default-lease-time            21600;
max-lease-time                21600;

option subnet-mask            255.255.255.0;
option broadcast-address      192.168.0.255;
option routers                192.168.0.254;
option domain-name-servers    192.168.0.254;
option domain-name            "ltsp.org";
option root-path              "192.168.0.254:/opt/ltsp/i386";

shared-network WORKSTATIONS {
subnet 192.168.0.0 netmask 255.255.255.0 {
    }
}

group       {
    use-host-decl-names       on;
    option log-servers        192.168.0.254;

    host ws001 {
        hardware ethernet     00:E0:18:E0:04:82;
        fixed-address         192.168.0.1;
        filename              "/lts/vmlinuz.ltsp";
    }
}
```

Abbildung 2−1. /etc/dhcpd.conf

Seit der Version 2.09pre2 von LTSP ist es nicht mehr notwendig, einen speziellen Kernel anzugeben. Die Kernel des Standard−Paketes enthalten Treibermodule für alle von Linux unterstützten Netzwerkkarten. Es gibt zwei Kernel; einer enthält den Linux Progress Patch (LPP), der andere nicht. Sie sind an ihren Namen zu erkennen:

```
vmlinuz-2.4.9-ltsp-5
vmlinuz-2.4.9-ltsp-lpp-5
```

Vielleicht ist Ihnen aufgefallen, das der Kernel im Verzeichnis `/tftpboot/lts` liegt, der Eintrag "filename" in der Datei `/etc/dhcpd.conf` aber nicht den Vorsatz `/tftpboot` enthält. Das hat folgenden Grund: Seit der Red−Hat Versions 7.1 läuft TFTP mit der Option '−s', d.h der Prozess tftpd läuft im ***secure*** Modus: Beim Start wird ein **chroot** zum Verzeichnis `/tftpboot` ausgeführt. Damit sind für den tftpd−Prozess alle Dateien relativ zum Verzeichnis `/tftpboot` erreichbar.

Falls bei Ihrer Distribution die Option '−s' nicht verwendet wird, muss die Angabe `/tftpboot` vor den Pfadnamen des Kernels gesetzt werden.

2.4.2. /etc/hosts

IP−Adressen und Rechnernamen

Computer kommen i.a. mit IP−Adressen aus. Menschen haben lieber Namen für ihre Rechner. Für die Zuordnung gibt es Nameserver (DNS) oder die Datei `/etc/hosts`. In einer LTSP−Umgebung ist dies unverzichtbar, da es sonst keinen Zugriff der Clients auf das vorgesehene root−Verzeichnis auf dem Server per NFS gibt.

Ausserdem könnte es bei fehlendem Eintrag des Clients in der Datei `/etc/hosts` zu Problemen mit ***GDM*** oder ***KDM*** kommen.

2.4.3. /opt/ltsp/i386/etc/lts.conf

In dieser Datei gibt es eine ganze Reihe von Konfigurationseinträgen.

Die Datei `lts.conf` hat eine einfache Syntax und besteht aus mehreren Sektionen. Es gibt eine Sektion, die für alle Clients gilt: **[default]** und es kann für einzelne Clients spezifische Einträge geben Der Client kann durch Rechnernamen, IP−Adresse oder MAC−Adresse angegeben werden.

Eine `lts.conf` Datei sieht üblicherweise so aus:

```
#
# Config file for the Linux Terminal Server Project (www.ltsp.org)
#

[Default]
        SERVER             = 192.168.0.254
        XSERVER            = auto
        X_MOUSE_PROTOCOL   = "PS/2"
        X_MOUSE_DEVICE     = "/dev/psaux"
        X_MOUSE_RESOLUTION = 400
        X_MOUSE_BUTTONS    = 3

# Diese Werte sollten gesetzt werden, um den deutschen Zeichensatz
# unter X Window benutzen zu können; voreingestellt ist der englische.

        XkbModel           = pc104
        XkbLayout          = de

# Ende der besonderen Einstellungen

        USE_XFS            = N
        LOCAL_APPS         = N
        RUNLEVEL           = 5

[ws001]
        USE_NFS_SWAP       = Y
        SWAPFILE_SIZE      = 48m
        RUNLEVEL           = 5
```

```
[ws002]
        XSERVER             = XF86_SVGA
        LOCAL_APPS          = N
        USE_NFS_SWAP        = Y
        SWAPFILE_SIZE       = 64m
        RUNLEVEL            = 3
```

Beispiel 2−1. lts.conf file

Es folgt eine Liste einiger Konfigurationsvariablen:

XSERVER

Wenn der Client eine von XFree86 4.1 unterstützte PCI−Graphikkarte hat, dann reicht es aus, das Paket ltsp_x_core installiert zu haben. Alles läuft automatisch ab, ein Eintrag ist nicht erforderlich.

Falls der Client eine ISA−Karte hat oder die (PCI)−Karte nicht unterstützt wird, muss hier der richtige X−Server von XFree86 3.3.6 angegeben werden. Es stehen dafür mehrere LTSP−Pakete zur Verfügung.

Einträge können in der Datei `lts.conf` pro Client oder zentral in der default−Sektion erfolgen.

Da unser Beispiel−Client ein Intel i810 video Chipset hat, was automatisch erkannt wird, braucht kein XSERVER−Eintrag gemacht zu werden. Wenn man will, kann man 'auto' eintragen oder das Treibermodul.

RUNLEVEL

Da der Client im Graphik−Modus laufen soll, wird runlevel auf '5' gesetzt.

Kapitel 3. Die Konfiguration des Client−Rechners

Nach dem Einrichten des Servers kommt nun der Client an die Reihe.

LTSP beginnt dann, wenn der Kernel im Arbeitsspeicher des Client ist. Es gibt mehrere Wege, den Kernel zu laden: Etherboot, Netboot, PXE und Diskette.

Hier gehen wir davon aus, den Rechner von Diskette zu booten. Die Diskette wird Code des **Etherboot** Projekts enthalten.

3.1. Bootdiskette herstellen

> Etherboot ist ein Softwarepaket zum Erstellen von ROM−Images. Dieses Image enthält Code, der in einem Ethernet−Netzwerk einen Download von Software durchführen kann. Viele Netzwerkkarten haben einen Sockel für ein Bootrom, auf das dieser Code übertragen werden kann.
>
> −−Ken Yap

Auch Etherboot ist Open Source und steht unter der GNU General Public License, Version 2 (GPL2).

Sie können das Etherboot−Paket downloaden und ein passendes Image herstellen, das dann in ein EPROM gebrannt oder erst einmal zu Testzwecken auf eine Diskette kopiert werden kann.

Eine einfachere Alternative sind Marty Connor's Webseiten unter *www.Rom−O−Matic.net*.

Marty hat ein exzellentes web−basiertes frontend geschaffen, um ein angepasstes Bootrom mit Etherboot zu generieren. Man wählt einfach seine Netzwerkkarte aus und gibt an, welche Art von Image es sein soll. Es gibt viele weitere Konfigurationsmöglichkeiten. Nachdem alles eingegeben ist, reicht ein Klick auf den Button 'Get ROM', um das Image generieren zu lassen.

Unser Beispielrechner hat eine Netzwerkkarte vom Typ Linksys LNE100TX, version 4.1. Die Karte hat einen ADMTek Centaur−P Chipsatz; deshalb ist ***centaur−p*** als NIC/ROM−Typ zu wählen.

Änderungen der Voreinstellungen sind nicht notwendig, daher überspringen wir den 'Configure' Button.

Als 'ROM output format' wählen wir 'Floppy Bootable ROM Image'. Dadurch bekommt das Image einen 512 Byte großen Header als boot−loader für das eigentliche Etherboot Image verpasst. Der boot−loader lädt bei Start des Rechners per Diskette das Image in den Arbeitsspeicher, wo es dann ausgeführt wird.

Jetzt ist der 'Get ROM' Button an der Reihe. Es dauert nur einige Sekunden, bis das Image generiert worden ist und auf dem eigenen Rechner gespeichert werden kann. In diesem Beispielfall hat das Image den Namen 'eb−5.0.2−centaur−p.lzdsk'.

Das Image wird nun auf eine 3,5"−Diskette kopiert (kann natürlich auch eine 5,25"−Diskette sein, dann entsprechend anpassen)

```
cat   /pfad_zum_image/eb−5.0.2−centaur−p.lzdsk   >/dev/fd0
```

Kapitel 4. Starten des Client−Rechners

Eigentlich sollte nun Starten per Diskette reichen − wenn alles richtig konfiguriert wurde:

Der Etherboot−Code wird von der Diskette in den Arbeitsspeicher geladen, die Netzwerkkarte wird erkannt und initialisiert, eine DHCP−Anfrage wird ins Netzwerk geschickt, der DHCP−Server antwortet, der Kernel wird vom Server heruntergeladen und gestartet. Der Kernel erkennt die Hardware, X startet und ein graphisches Login erscheint auf dem Monitor, ähnlich wie im unten gezeigten Beispiel.

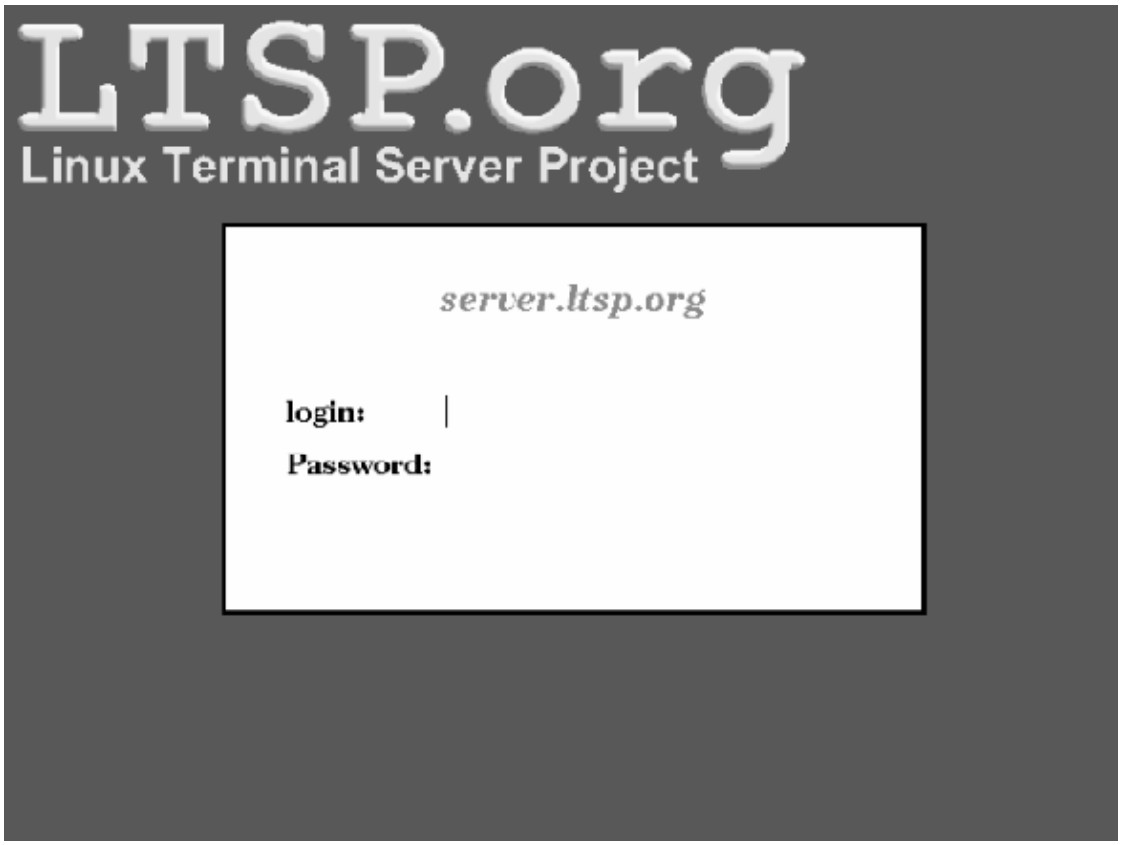

Abbildung 4−1. Login−Bildschirm

Normales Einloggen sollte nun möglich sein. Wichtig ist: Man meldet sich auf dem Server an und arbeitet dort. Alle Kommandos werden auf dem Server aufgeführt, der Output auf dem Client−Monitor dargestellt. Das ist die Netzwerk−Fähigkeit des X Window Systems.

Alle Programme auf dem Server stehen zur Verfügung.

Kapitel 5. Drucken

Der Client kann neben seiner Rolle als voll funktionsfähiges Terminal auch noch als Printserver benutzt werden. Bis zu drei Drucker können an die parallelen und seriellen Ports angeschlossen werden.

Dies ist für den Benutzer des Client−Rechners transparent; die zusätzliche Netzwerklast wird kaum bemerkbar sein.

5.1. Einrichten des Client−Rechners

LTSP benutzt das **lp_server** Programm auf dem Client, um die Druckaufträge des Servers auf die Ports des Client umzuleiten.

Es gibt einige Einträge in der Konfigurationsdatei **lts.conf** um das Drucken zu steuern:

```
[ws001]
    PRINTER_0_DEVICE = /dev/lp0
    PRINTER_0_TYPE   = P
```

Dieser Eintrag lässt das lp_server Programm als Dämon laufen, der auf dem TCP/IP−Port 9100 auf Druckaufträge wartet. Der Drucker am ersten parallelen Anschluss des Client−Rechners bekommt die Daten durchgereicht.

Es gibt viele weitere Einstellmöglichkeiten fürs Drucken. Man findet diese weiter unten in diesem Dokument in der Beschreibung von lts.conf.

5.2. Einrichten des Servers

Das Einrichten auf dem Server beinhaltet die Definition einer Druckerwarteschlange mit einem entsprechenden Konfigurationstool.

Redhat 7.2, hat dafür sowohl GUI− wie textbasierte Tools. Das GUI Tool heißt **printconf−gui**, und das textbasierte **printconf−tui**. ältere Versionen von Redhat haben ein Programm namens **printtool**. Printtool gibt es auch in Redhat 7.2, aber in diesem Beispiel benutzen wir printconf−gui. Andere Linux−Distributionen haben ihre eigenen Konfigurationstools, neuere Versionen von SuSE beispielsweise erlauben die Konfiguration mit YAST, YAST2 oder lprsetup.

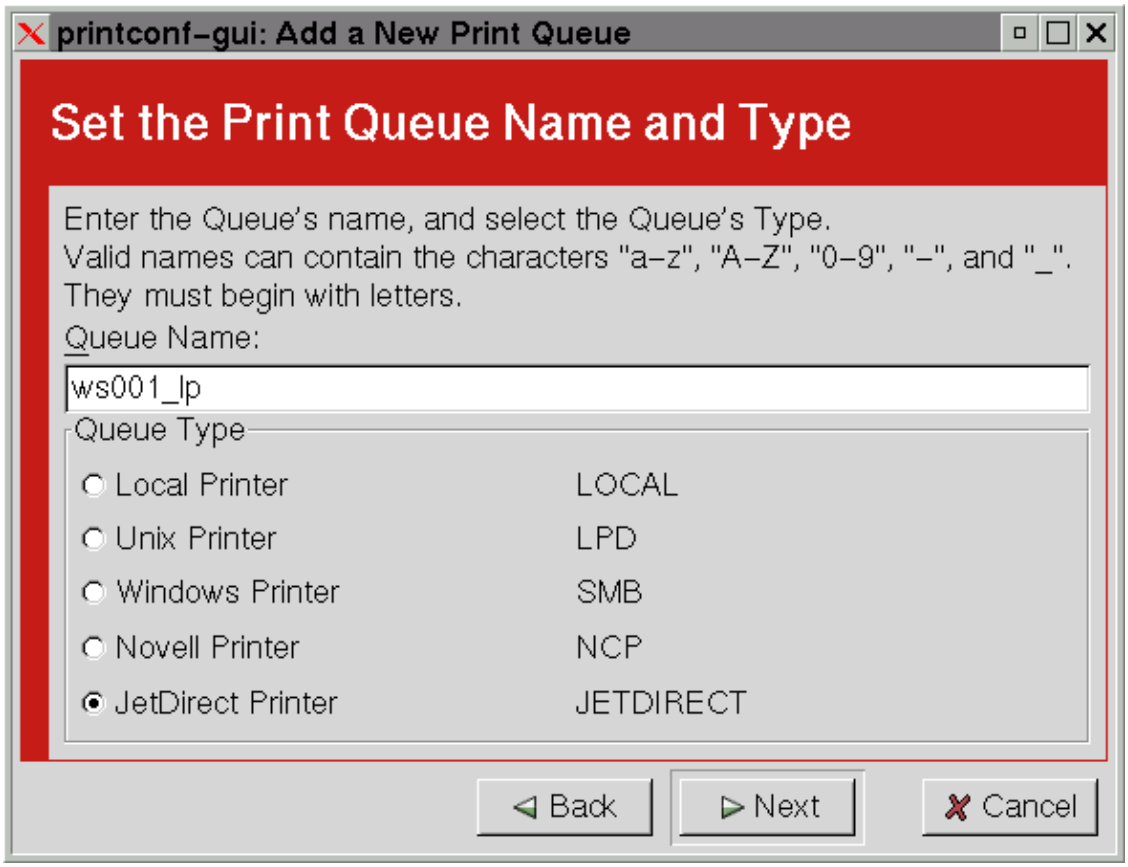

Abbildung 5−1. Drucker definieren mit printconf−gui

Nach Aufruf des Tools definieren wir einen neuen Drucker. Das lp_server Programm kann auf dem Client einen HP JetDirect Printserver emulieren. Auf dem Server muss daher ein **JetDirect** Drucker eingerichtet werden.

Der neu einzurichtende Drucker braucht einen Namen für die Warteschlange. Dieser Name kann zwar beliebig gewählt werden, sollte aber sinnvoll sein und aus folgenden Zeichen zusammengesetzt:

- `"a-z" Kleinbuchstaben`
- `"A-Z" Großbuchstaben`
- `"0-9" Ziffern`
- `"-"   Bindestrich`
- `"_"   Unterstrich`

Im Beispiel nehmen wir den Namen **ws001_lp**. Damit ist klar, dass dieser Drucker am Client **ws001** betrieben wird.

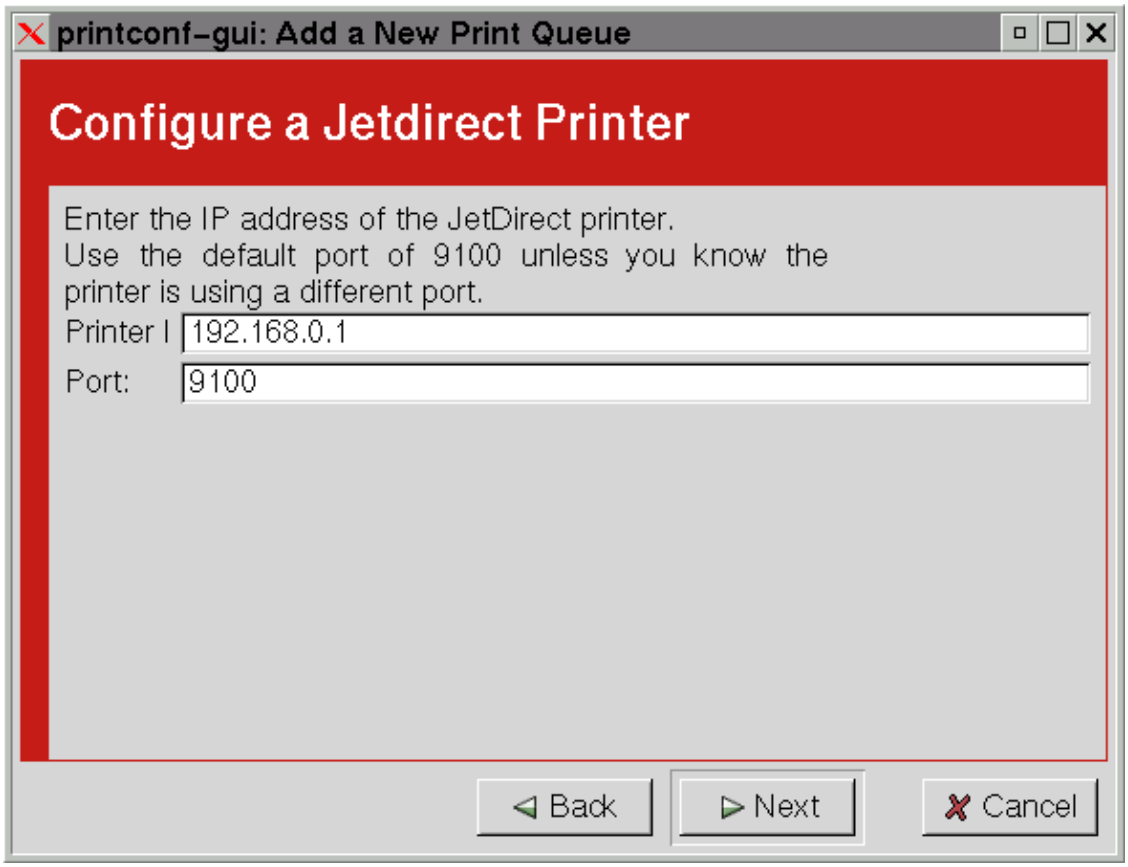

Abbildung 5−2. printconf−gui Detail Info

Zwei Felder sind auszufüllen:

1. IP−Adresse oder Name des Rechners, an den der Drucker angeschlossen ist.
2. Der TCP−Port des **lp_server** Prozesses.

   Für den ersten Drucker an einem Client ist das Port **9100**, für weitere die Ports **9101** und **9102**.

Kapitel 6. Fehlersuche

Für den Fall, dass der Client nicht bootet oder sonst etwas falsch läuft, hier einige Tipps zur Fehlersuche und −beseitigung.

Stellen Sie fest, bis wohin der Bootvorgang abläuft.

6.1. Fehlersuche beim Starten von Diskette

Einige Möglichkeiten, wie es beim Start von Diskette aussehen könnte:

```
loaded ROM segment 0x0800 length 0x4000 reloc 0x9400
Etherboot 5.0.1 (GPL) Tagged ELF for [LANCE/PCI]
Found AMD Lance/PCI at 0x1000, ROM address 0x0000
Probing...[LANCE/PCI] PCnet/PCI−II 79C970A base 0x1000, addr 00:50:56:81:00:01
Searching for server (DHCP)...
<sleep>
```

Wie das Beispiel zeigt, kann man den Startvorgang auf dem Monitor beobachten. Sieht man nichts, dann kann dies an einer fehlerhaften Diskette liegen oder daran, dass das Image nicht richtig abgelegt wurde.

Bei einer Meldung wie der folgenden kann man vermuten, dass der falsche Treiber auf die Diskette geschrieben wurde.

```
ROM segment 0x0800 length 0x8000 reloc 0x9400
Etherboot 5.0.2 (GPL) Tagged ELF for [Tulip]
Probing...[Tulip]No adapter found
<sleep>
<abort>
```

Wenn die Netzwerkkarte erkannt wird und die MAC−Adresse richtig angegeben wird, dann ist die Diskette wohl in Ordnung und der Fehler liegt woanders.

6.2. Fehlerursache DHCP

Sobald die Netzwerkkarte initialisiert wurde, schickt der Etherboot−Code eine DHCP−Anfrage per Broadcast ins lokale Netz.

Wenn der Client−Rechner eine Antwort bekommt, dann konfiguriert der Code die Netzwerkkarte. Wenn die IP−Adresse auf dem Monitor angezeigt wird, ist alles klar gegangen. Hier ein Beispiel, wie es aussehen sollte:

```
ROM segment 0x0800 length 0x4000 reloc 0x9400
Etherboot 5.0.1 (GPL) Tagged ELF for [LANCE/PCI]
Found AMD Lance/PCI at 0x1000, ROM address 0x0000
Probing...[LANCE/PCI] PCnet/PCI−II 79C970A base 0x1000, addr 00:50:56:81:00:01
Searching for server (DHCP)...
<sleep>
Me: 192.168.0.1, Server: 192.168.0.254, Gateway 192.168.0.254
```

Wenn Sie eine Zeile wie die letzte sehen, dann arbeitet DHCP richtig und der Fehler muss woanders liegen (TFTP).

Umgekehrt ist etwas falsch, wenn die folgende Meldung erscheint und danach jede Menge <sleep> Meldungen erscheinen. Normal sind ein bis zwei <sleep> Meldungen bis der DHCP−Server antwortet.

```
Searching for server (DHCP)...
```

Die Fehlerursache herauszufinden kann schwierig sein, hier ein paar Hinweise:

6.2.1. Verbindungen überprüfen

Ist der Client physikalisch richtig an dasselbe Netz wie der Server angeschlossen?

Kontrollieren Sie nach dem Einschalten des Clients, ob die 'link'−LEDs überall aufleuchten.

Bei einer Direktverbindung zwischen Server und Client muss ein cross−Kabel verwendet werden, bei Einsatz von Hub oder Switch ausschließlich normale Patchkabel.

6.2.2. Arbeitet dhcpd?

Stellen Sie fest, ob der **dhcpd** Prozess auf dem Server läuft. Dazu gibt es mehrere Möglichkeiten:

dhcpd läuft normalerweise im Hintergrund und wartet auf dem udp−Port 67 auf Anfragen. Führen Sie das Kommando **netstat** aus, um zu sehen, ob der Prozess vorhanden ist:

```
netstat −an | grep ":67 "
```

Es sollte etwa folgendes zu sehen sein:

```
udp     0    0   0.0.0.0:67         0.0.0.0:*
```

Die vierte Spalte zeigt, getrennt durch Doppelpunkt, die IP−Adresse und den Port an. überall Nullen zeigen an, dass auf allen Netzwerkkarten Anfragen entgegengenommen werden. Gibt es also etwa **eth0** und **eth1** dann wartet **dhcpd** auf beiden Interfaces auf Anfragen.

Die Tatsache alleine, dass auf Port 67 ein Prozess auf Anfragen wartet, heißes noch nicht, dass dies **dhcpd** ist, es könnte auch **bootpd** sein.

Um sicherzugehen, dass tatsächlich **dhcpd** läuft, führen Sie das Kommando **ps** aus.

```
ps aux | grep dhcpd
```

Dann sollte es etwa so aussehen:

```
root 23814 0.0 0.3 1676 820 ?      S 15:13 0:00 /usr/sbin/dhcpd
root 23834 0.0 0.2 1552 600 pts/0  S 15:52 0:00 grep dhcp
```

Die erste Zeile zeigt an, dass es der Fall ist.

Wenn Sie keine solche Zeile sehen, dann überprüfen Sie, ob das automatische Starten des dhcp−Servers im runlevel 5 eingestellt ist. Auf Redhat basierenden Systemen kann man dazu das Kommando **ntsysv** ausführen,

in der erscheinenden Liste nach unten scrollen und sich vergewissern, dass **dhcpd** angekreuzt ist. Wenn Sie SuSE einsetzen geht das entsprechend mit YAST2.

Sie können auch versuchen, den Prozess erst einmal von Hand zu starten. Kommando bei Redhat:

```
service dhcpd start
```

Und bei SuSE

```
rcdhcp start
```

Achten Sie auf eventuelle Fehlermeldungen.

6.2.3. Überprüfen Sie die Konfigurationsdatei

Gibt es in der Datei `/etc/dhcpd.conf` einen Eintrag für den Client?

überprüfen Sie insbesondere den 'fixed−address' Teil auf Übereinstimmung mit der Client−Hardware.

6.2.4. Blockieren ipchains oder iptables die Anfrage?

6.2.4.1. Checken von ipchains

Führen Sie das folgende Kommando aus und beobachten Sie die Ausgabe:

```
ipchains −L −v
```

Wenn Sie etwas wie dies sehen, dann liegt es nicht an ipchains:

```
Chain input (policy ACCEPT: 229714 packets, 115477216 bytes):
Chain forward (policy ACCEPT: 10 packets, 1794 bytes):
Chain output (policy ACCEPT: 188978 packets, 66087385 bytes):
```

6.2.4.2. Checken von iptables

Führen Sie das folgende Kommando aus und beobachten Sie die Ausgabe:

```
iptables −L −v
```

Wenn Sie etwas wie dies sehen, dann liegt es nicht an ipchains:

```
Chain INPUT (policy ACCEPT 18148 packets, 2623K bytes)
 pkts bytes target     prot opt in     out     source

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source

Chain OUTPUT (policy ACCEPT 17721 packets, 2732K bytes)
 pkts bytes target     prot opt in     out     source
```

6.2.5. Sendet der Client die Anfrage?

Beobachten Sie auf dem Server die Datei `/var/log/messages` während der Client bootet. Das geht so:

```
tail −f /var/log/messages
```

Damit blättert der Dateiinhalt vor sich hin, jeder neue Eintrag wird sofort angezeigt.

```
server dhcpd: DHCPDISCOVER from 00:50:56:81:00:01 via eth0
server dhcpd: no free leases on subnet WORKSTATIONS
server dhcpd: DHCPDISCOVER from 00:50:56:81:00:01 via eth0
server dhcpd: no free leases on subnet WORKSTATIONS
```

Eine solche Meldung besagt, das **dhcpd** läuft, aber nichts über die anfragende Maschine weiß. Kontrollieren Sie die Einträge für den Client in /etc/dhcpd.conf.

6.3. Fehlersuche TFTP

Etherboot benutzt TFTP, um den Linux−Kernel vom Server zu laden. Es handelt sich um ein ziemlich einfaches Protokoll; TFTP macht aber manchmal Probleme.

Wenn Sie auf dem Client beim Startprozess folgendes sehen:

```
Loading 192.168.0.254:/lts/vmlinuz−2.4.9−ltsp−5 |
```

und dabei das letzte Zeichen der Zeile (das '|') schnell zwischen den Zeichen '|', '\', '−' and '/' wechselt, dann zeigt dies an, dass der Kernel geladen wird. TFTP sollte damit richtig laufen.

Sieht man diesen Zeichenwechsel nicht, zeigt das ein Problem an. Möglicherweise ist dies falsch:

6.3.1. tftpd läuft nicht

Unter Redhat 7.1 wird tftp von **xinetd** gestartet. Es gibt ein Startscript namens `/etc/xinetd.d/tftp`, welches Informationen für den **tftpd** Prozess enthält.

6.3.2. Kernel nicht an der von tftpd erwarteten Stelle

Der Kernel muss an einer Stelle liegen, auf die **tftpd** zugreifen kann. Wenn zum Starten von **tftpd** die '−s' Option benutzt wird, dann müssen alle angegebenen Dateinamen relativ zu `/tftpboot` sein. Beispielsweise muss der Kernel `/tftpboot/lts/vmlinuz−2.4.9−ltsp−5` sein, wenn der **filename** Eintrag in der Datei `/etc/dhcpd.conf` auf `/lts/vmlinuz−2.4.9−ltsp−5` lautet.

6.4. Fehlersuche root−Dateisystem per NFS

Es gibt mehrere Dinge, die dem Mounten des root−Dateisystems im Wege stehen können:

6.4.1. Der init Prozess fehlt

Bei der folgenden Fehlermeldung:

```
Kernel panic: No init found.  Try passing init= option to kernel.
```

ist es sehr wahrscheinlich, dass entweder versucht wird, ein falsches root−Dateisystem zu mounten oder dass das 'richtige' Verzeichnis leer ist.

6.4.2. Der Server meldet 'error −13'

Die Fehlermeldung:

```
Root−NFS: Server returned error −13 while mounting /opt/ltsp/i386
```

zeigt an, dass entweder das Verzeichnis `/opt/ltsp/i386` nicht in der Datei `/etc/exports` enthalten ist oder dort fehlerhaft eingetragen wurde.

Sehen Sie in der Datei `/var/log/messages` nach irgendwelchen Hinweisen. Ein Eintrag wie dieser:

```
Jul 20 00:28:39 jamlap rpc.mountd: refused mount request from ws004
  for /opt/ltsp/i386 (/): no export entry
```

bestätigt z.B. den Verdacht, dass der Eintrag `/etc/exports` falsch ist.

6.4.3. NFS Daemon Probleme (portmap, nfsd & mountd)

Es kann schwierig sein, NFS−Fehler aufzuspüren. Wenn man versteht, was konfiguriert werden muss und welche Diagnosetools zur Verfügung stehen, wird es wohl etwas einfacher.

Auf dem Server müssen drei Prozesse laufen, damit NFS richtig funktioniert. Dies sind: **portmap**, **nfsd** und **mountd**.

6.4.3.1. Der Portmapper (portmap)

Bei diesen Meldungen:

```
Looking up port of RPC 100003/2 on 192.168.0.254
portmap: server 192.168.0.254 not responding, timed out
Root−NFS: Unable to get nfsd port number from server, using default
Looking up port of RPC 100005/2 on 192.168.0.254
portmap: server 192.168.0.254 not responding, timed out
Root−NFS: Unable to get mountd port number from server, using default
mount: server 192.168.0.254 not responding, timed out
Root−NFS: Server returned error −5 while mounting /opt/ltsp/i386
VFS: unable to mount root fs via NFS, trying floppy.
VFS: Cannot open root device "nfs" or 02:00
Please append a correct "root=" boot option
Kernel panic: VFS: Unable to mount root fs on 02:00
```

```
```

kann man vermuten, dass der Portmapper **portmap** nicht läuft. Feststellen lässt sich dies durch das **ps** Kommando:

```
ps −e | grep portmap
```

Bei laufendem Portmapper sieht man dies:

```
30455 ?        00:00:00 portmap
```

Eine weitere Testmöglichkeit bietet das **netstat** Kommando. Der Portmapper benutzt die TCP und UDP Ports 111. Führt man folgendes aus:

```
netstat −an | grep ":111 "
```

dann sollte dies zu sehen sein:

```
tcp   0   0 0.0.0.0:111       0.0.0.0:*           LISTEN
udp   0   0 0.0.0.0:111       0.0.0.0:*
```

Sieht man nichts, dann läuft der Portmapper auch nicht. Starten per Hand geht so:

```
/etc/rc.d/init.d/portmap   start
```

oder bei neuerer SuSE−Distribution:

```
rcportmap   start
```

Fall so der Portmapper ans Laufen gebracht werden kann, dann sollten Sie nun einstellen, dass er beim Hochfahren des Servers automatisch gestartet wird. Unter Redhat benutzen Sie dazu das Kommando **ntsysv**, unter SuSE YAST oder YAST2.

6.4.3.2. Die Prozesse NFS und MOUNT (nfsd & mountd)

Für NFS müssen zwei Prozesse laufen: **nfsd** und **mountd**. Beide werden durch das Script `/etc/rc.d/init.d/nfs` gestartet.

Mit dem Kommando **ps** können Sie feststellen, ob beide Prozesse laufen.

```
ps −e | grep nfs
ps −e | grep mountd
```

Falls einer oder beide nicht laufen, dann müssen diese nun gestartet werden.

Eigentlich sollte hier das **restart** Argument nutzbar sein, aber auf diese Weise wird kein Neustart von **nfsd** erfolgen (Bug?). Deshalb muss man folgendes tun:

```
/etc/rc.d/init.d/nfs  stop
/etc/rc.d/init.d/nfs  start
```

Vielleicht gibt es Fehlermeldungen des **stop** Kommandos, aber das ist normal. Das **start** sollte allerdings **o.k.** zurückgeben.

Wenn die Prozesse laufen, NFS aber immer noch nicht klappt, dann kann mit dem **rpcinfo** Kommando feststellen, ob sie sich auch beim Portmapper angemeldet haben.

```
rpcinfo −p localhost
```

Man sollte etwas wie dies sehen:

```
program vers proto   port
 100000    2   tcp    111  portmapper
 100000    2   udp    111  portmapper
 100011    1   udp    856  rquotad
 100011    2   udp    856  rquotad
 100005    1   udp   1104  mountd
 100005    1   tcp   2531  mountd
 100005    2   udp   1104  mountd
 100005    2   tcp   2531  mountd
 100003    2   udp   2049  nfs
```

Dies zeigt an, das **nfs** (nfsd) und **mountd** laufen und sich beim Portmapper registriert haben.

6.5. Fehlersuche X−Server

Teufel auch! Die schwierigste Einzelaufgabe ist wahrscheinlich das richtige Konfigurieren des X−Servers. Bei halbwegs neuer, von Linux unterstützter Graphikkarte und einem Monitor, der viele Frequenz− und Auflösungseinstellungen beherrscht ist es kein Problem. Falls es nicht klappt liegt es meist am falschen X−Server für die Karte.

Es lässt sich einfach feststellen, ob es der falsche X−Server ist: Entweder bricht der Start des Servers mit Fehlermeldungen ab oder die Anzeige sieht verzerrt oder irgendwie komisch aus.

Auf dem Client wird der X−Server durch das Script `/tmp/start_ws` gestartet. Dies geschieht in runlevel 5 als letzter Schritt des Startvorganges automatisch kann aber auch im runlevel 3 per Hand ausgeführt werden. Das Script startet den X−Server mit der **−query** Option, die auf einen Rechner verweist, auf dem ein Display−Manager wie **XDM**, **GDM** oder **KDM** läuft.

Da der X−Server durch das Script `/tmp/start_ws` gestartet wird, das Script selbst aber durch **init**, wird das **init** 10 mal versuchen, den X−Server zu starten bevor es aufgibt ("respawning too fast"). Wenn man bis dahin noch nicht den Client−Rechner abgeschaltet hat, wird man die Fehlermeldung des X−Servers auf dem Monitor finden.

10 mal zu sehen, wie der X−Server es nicht schafft kann ziemlich irritierend sein. Da startet man den Rechner doch lieber im runlevel 3, so dass kein automatischer Start des X−Servers erfolgt. Im runlevel 3 hat man nach dem Start einen shell−Prompt der **bash**. Der X−Server kann nun manuell gestartet werden:

```
/tmp/start_ws
```

Statt nach 10 Versuchen gibt der X−Server nun nach einem auf und die Fehlermeldungen lassen sich nachlesen.

6.6. Fehlersuche Display−Manager

Der Display−Manager ist ein auf dem Server laufender Prozess, der darauf wartet, von einem X−Server kontaktiert zu werden. Wenn dies passiert, dann schickt der Prozess ein Anmeldefenster auf den Bildschirm der Maschine, auf dem der anfragende X−Server läuft. Benutzer können sich nun zur Nutzung von Programmen des Servers anmelden.

Die drei am häufigsten benutzten Display−Manager sind:

- XDM − Den gibt es schon seit ewigen Zeiten. Er gehört zum X Window System.
- GDM − Der 'Gnome Display−Manager'. Dieser gehört zum gnome−Paket.
- KDM − Der 'KDE Display Manager'. Und dieser ist Teil des KDE−Desktop−Systems

Neuere GNU/Linux Distributionen haben alle drei im Angebot.

6.6.1. Grauer Bildschirm mit großem Cursor in Form eines X

Dies bedeutet, dass der X−Server läuft, aber keinen Kontakt zu einem Display−Manager herstellen konnte. Mögliche Gründe:

1. Auf dem angesprochenen Rechner läuft kein Display−Manager

   Neuere Versionen von Redhat (7.0 und höher), haben den Start des Display−Managers in den **init** Prozess integriert. In der Datei `/etc/inittab` gibt es eine Zeile, die so aussieht:

   ```
   x:5:respawn:/etc/X11/prefdm −nodaemon
   ```

   Das **prefdm** Script legt fest, welcher Display−Manager gestartet werden soll.

   Welches der voreingestellte Display−Manager ist, hängt von den installierten Paketen ab. Ist Gnome installiert, dann ist GDM voreingestellt. Falls Gnome nicht installiert ist, prüft das prefdm−Script ob KDE installiert ist. Im zutreffenden Fall wird KDM als Display−Manager voreingestellt. Ist KDE ebenfalls nicht installiert, dann wird GDM zum voreingestellten Display−Manager.

   Mittels des **netstat** Kommandos, lässt sich feststellen, welcher Display−Manager läuft. Auf dem Server gibt man folgenden Befehl ein:

   ```
   netstat −ap | grep xdmcp
   ```

   Es sollte zu sehen sein, dass ein Prozess auf dem xdmcp−Port (177) auf Anfragen wartet.

   ```
   udp     0   0 *:xdmcp           *:*               1493/gd
   ```

   Man sieht hier, dass **gdm** mit der PID 1493 läuft und auf dem xdmcp−Port bereit steht.

   Falls solch eine Zeile zu sehen ist, muss als nächstes nachgesehen werden, ob der Client die XDMCP−Anfrage an den richtigen Server stellt.

In der Datei `lts.conf` lässt sich die IP−Adresse des Servers angeben, dessen Display−Manager um ein graphisches Login angefragt werden soll. Der Eintrag ist optional. Falls er vorhanden ist, sollte er so aussehen:

```
XDM_SERVER  =  192.168.0.254
```

Die IP−Adresse kann in Ihrem Fall natürlich anders sein.

Wenn der Eintrag 'XDM_SERVER' nicht vorhanden ist, dann wird der Eintrag 'SERVER' benutzt. Ist auch dieser nicht vorhanden, wird **192.168.0.254** genommen.

Was auch immer angegeben wird, es muss sich um die richtige IP−Adresse des Rechners handeln, auf dem der Display−Manager läuft.

2. Möglicherweise ist der Display−Manager so konfiguriert, dass Anfragen anderer Rechner aus Sicherheitsgründen abgewiesen werden.

   Wenn Sie sicher sind, dass der Display−Manager läuft, dann prüfen Sie nun, ob der Display−Manager vielleicht so wie gerade erwähnt konfiguriert wurde.

   - **XDM**

     Bei Redhat ist voreingestellt, dass Anfragen von anderen Rechnern abgelehnt werden, bei neueren SuSE−Distributionen ebenfalls. Das weit oben erwähnte Script **ltsp_initialize** sollte eigentlich die notwendige Änderung an den Konfigurationsdateien vorgenommen haben. Wenn Sie keinen Zugang bekommen, sehen Sie aber besser einmal in der Datei `/etc/X11/xdm/xdm−config` nach. Suchen Sie nach einem Eintrag, der etwa so aussieht:

     ```
     DisplayManager.requestPort:     0
     ```

     Diese Zeile muss unbedingt auskommentiert werden, damit Anfragen anderer Rechner entgegengenommen werden.

     Damit XDM Anfragen anderer Rechner bedienen kann, ist noch der Inhalt einer weiteren Datei von Bedeutung: `/etc/X11/xdm/Xaccess`. In dieser Datei muss es unbedingt eine Zeile geben, die mit dem Zeichen '*' beginnt. Normalerweise gibt es diese Zeile, bei Redhat ist sie allerdings auskommentiert. Das **ltsp_initialize** Script sollte hier für den richtigen Eintrag gesorgt haben, aber sehen Sie bei Problemen auch in dieser Datei nach. Eigentlich sollte dort solch eine Zeile zu sehen sein:

     ```
     *        #any host can get a login window
     ```

   - **KDM**

     Neuere Versionen von KDM benutzen eine Datei namens **kdmrc**. Leider legen die verschiedenen Linux−Distributionen diese Konfigurationsdatei an unterschiedlichen Positionen ab; Redhat 7.2 z.B. unter **/etc/kde/kdm/kdmrc**. Am besten benutzen Sie bei anderen Distributionen das Kommando **locate kdmrc**, um die Datei zu lokalisieren.

Die Zugangskontrolle wird in dieser Datei in der Sektion **[Xdmcp]** festgelegt. Kontrollieren Sie, ob **Enable** auf **true** gesetzt ist.

ältere KDM−Versionen benutzen die XDM−Konfigurationsdateien, die im Verzeichnis `/etc/X11/xdm` zu finden sind.

- **GDM**

GDM benutzt andere Konfigurationsdateien. Diese befinden sich im Verzeichnis `/etc/X11/gdm`.

Kontrollieren Sie in der Hauptdatei `gdm.conf` den Abschnitt **[xdmcp]**. Dort sollte ein Eintrag 'Enable' zu sehen sein. Dieser muss − je nach GDM−Version − auf '1' oder 'true' gesetzt sein, wie in diesem Beispiel:

```
[xdmcp]
Enable=true
HonorIndirect=0
MaxPending=4
MaxPendingIndirect=4
MaxSessions=16
MaxWait=30
MaxWaitIndirect=30
Port=177
```

Beachten Sie die Zeile mit 'Enable=true'. ältere Versionen von GDM benutzen '0' und '1', neuere 'false' und 'true' für die Zugangskontrolle.

3. Wenn feststeht, dass der Display−Manager läuft und auch so konfiguriert ist, dass Anfragen fremder Rechner akzeptiert werden, dann kann die Fehlerursache noch folgende sein: Die Zuordnung von IP−Adressen zu Rechnernamen stimmt nicht. Der Name des Client−Rechners muss entweder in der Datei `/etc/hosts` enthalten sein oder einen gültigen Eintrag in den Nameserver−Dateien haben.

Kapitel 7. Kernel

Bei dem Kernel, der auf dem Client laufen soll, kann man einen der vorbereiteten Kernel benutzen, die per Download verfügbar sind, man kann aber auch einen eigenen Kernel konfigurieren und kompilieren. Außerdem hat man die Wahl, ob beim Booten nur Textmeldungen oder ein Bild mit Fortschrittsbalken angezeigt werden soll. Letztere Variante wird durch den **Linux Progress Patch (LPP)** ermöglicht, allerdings nicht bei sehr alten Grafikkarten, da diese nicht über den erforderlichen VESA Modus verfügen.

7.1. Standard LTSP Kernel

Das Kernel−Paket von LTSP 3.0 enthält 2 Kernel, einen mit Linux Progress Patch und einen ohne.

Beide Kernel enthalten bereits den Patch, der Swap über NFS ermöglicht.

7.2. Einen Kernel selbst kompilieren

Es gibt 2 Wege, einen Kernel für LTSP zu konfigurieren: Die übliche Methode ist die Verwendung einer 'Initial Ram Disk' abgekürzt:**initrd** . Das initrd Image ist ein kleines Dateisystem, welches an die Kerneldatei angehängt wird. Das initrd Dateisystem wird in den Arbeitsspeicher geladen und wenn ein Kernel gebootet hat, mountet er diese RAM−Disk als root−Dateisystem. initrd bietet einige Vorteile: Wir können die Netzwerkkarten−Treiber als Modul kompilieren und beim Booten den passenden Treiber laden. Dies ermöglicht einen einzigen Kernel zu haben, der prinzipiell alle Netzwerkkarten unterstützt. Der andere Vorteil besteht darin, dass wir den DHCP−Client als "user−space"−Programm laufen lassen können, statt im "Kernel−space". Das DHCP Client−Programm im "user−space" laufen zu lassen ermöglicht eine bessere Kontrolle durch DHCP−Optionen, die vom Client angefordert und vom Server gesendet werden. Außerdem macht es den Kernel etwas kleiner. Der andere Weg, den Kernel zu konfigurieren ist, ohne initrd. Ein Kernel ohne initrd erfordert, dass der Netzwerkkarte−Treiber fest in den Kernel einkompiliert wird. Außerdem werden die Kernel−config Optionen "IP−Autoconfig" und "Root filesystem on NFS" benötigt. Ein Kernel ohne initrd ist etwas kleiner und bootet ein wenig schneller. Nachdem der Client−PC gebootet ist, gibt es im laufenden Betrieb aber praktisch keinen Unterschied mehr.

Der Standard−Kernel des LTSP enthält eine initrd, welche die Erkennung der Netzwerkkarte übernimmt und eine "user−space" DHCP Anfrage sendet. Es war ein wichtiges Ziel, dieses initrd−Image so klein wie möglich zu machen. Daher haben wir "uClinux" als Ersatz für die libc−library und "busybox" für die Tools verwendet.

Wenn Sie eigene Kernel kompilieren wollen, sollten Sie das Paket ltsp_initrd_kit herunterladen. Es beinhaltet das root−filesystem und ein Script, um ein Image zu erzeugen.

7.2.1. Woher bekommt man die Kernel−Quellen?

Wenn man sich einen eigenen Kernel baut, sollte man mit frischen, unveränderten Kernel−Quellen von **ftp.kernel.org** beginnen. Der Grund dafür ist, dass die Distributionen, wie z.B. SuSE & RedHat, viele Patches zum Kernel hinzufügen, sodass deren Kernel−Quellen sich vom offiziellen Kernel unterscheiden.

Besorgen Sie sich das gewünschte Kernel−Paket und speichern Sie es unter `/usr/src` . Die Kernel findet man unter `/pub/linux/Kernel` auf ftp.kernel.org. Wir benötigen einen aktuellen 2.4.x Kernel, damit das **devfs** Dateisystem unterstützt wird.

Wenn Sie NFS swapping oder Linux Progress Patch (LPP) benötigen, müssen Sie sicherstellen, dass die Patches zur Kernelversion passen. zur Zeit ist 2.4.9 die neueste Kernel Version, welche diese Features unterstützt.

Für unser Beispiel werden wir den Kerlen 2.4.9 verwenden. Der Download−Pfad lautet:
`ftp://ftp.kernel.org/pub/linux/Kernel/v2.4/linux−2.4.9.tar.bz2`

Packen Sie das Paket in `/usr/src` aus. Vorsicht: nach dem Auspacken befinden sich die Quellen in einem Verzeichnis mit dem Namen `linux` . Eventuell existiert bereits ein gleichnamiges Verzeichnis mit anderem Inhalt, und wir wollen das ja nicht vermischen. Deshalb sollten Sie prüfen, ob es schon ein Verzeichnis `linux` oder einen entsprechenden symbolischen Link gibt und dieses gegebenenfalls umbenennen oder verschieben, bevor Sie die neuen Quellen ausparken.

Das Quellpaket wurde mit **bzip2** komprimiert. Deshalb müssen wir es erst dekomprimieren bevor wir es mit **tar** auspacken:

```
bunzip2 <linux−2.4.9.tar.bz2 | tar xf −
```

Nach dem Auspacken haben Sie ein Verzeichnis mit dem Namen `linux` welches den gesamten Sourcebaum enthält. Zu diesem Zeitpunkt gebe ich dem Verzeichnis üblicherweise einen aussagekräftigen Namen.

```
mv linux linux−2.4.9
```

Nachdem das Verzeichnis umbenannt wurde, wechseln Sie hinein:

```
cd linux−2.4.9
```

Üblicherweise ändere ich das `Makefile` bevor ich mit der Kernel−Konfiguration beginne. Ziemlich weit oben gibt es eine Variable mit dem Namen **EXTRAVERSION**. Diese stelle ich auf '−ltsp−1' , sodass die Versionsnummer des neuen Kernels '2.4.9−ltsp−1' sein wird. Dies macht später die Unterscheidung einfacher. Der Anfang des Makefiles sollte dann so aussehen:

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 9
EXTRAVERSION = −ltsp−1

KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
```

7.2.2. Kernel Patches

Nach dem Auspacken des Kernels haben Sie möglicherweise diverse Patches, die Sie einfügen wollen. Zum Beispiel den NFS−Swap Patch oder den Linux Progress Patch. Man muss die Kernel−Quellen patchen BEVOR man den Kernel konfiguriert.

7.2.2.1. NFS Swap Patch

Der NFS Swap Patch ermöglicht das Benutzen einer Swap−Datei, welche sich auf einem NFS−Server befindet. Üblicherweise wird empfohlen genug RAM einzubauen, damit der Client nicht swappen muss. Aber manchmal ist es schwierig mehr RAM einzubauen, besonders bei älteren Computern. In solchen Fällen macht das NFS−swapping aus einem unbrauchbaren Rechner einen brauchbaren Rechner.

Wenn das aktuelle Verzeichnis /usr/src/linux−2.4.9 ist und der Patch sich in /usr/src befindet, kann man mit dem folgenden Kommando den Patch testen:

```
patch −p1 −−dry−run <../linux−2.4.9−nfs−swap.diff
```

So können Sie herausfinden, ob der Patch zum Kernel passt. Wenn dieser Probelauf ohne Fehler endet, dann können Sie den Patch ohne die Option **−−dry−run** in die Quellen einfügen.

```
patch −p1 <../linux−2.4.9−nfs−swap.diff
```

7.2.2.2. Linux Progress Patch (LPP)

Der Linux Progress Patch (LPP) ermöglicht es, dass ein graphisches Logo während des Bootens angezeigt wird. Die üblichen Boot−Meldungen werden auf eine andere Konsole umgeleitet und spezielle Anweisungen in den boot−Scripts sorgen dafür, dass ein Fortschrittsbalken den Bootvorgang visualisiert.

Genau wie beim NFS−Swap−Patch kann man den LPP mit

```
patch −p1 −−dry−run <../lpp−2.4.9
```

testen. Wenn der Test erfolgreich war, wird der Patch durchgeführt:

```
patch −p1 <../lpp−2.4.9
```

7.2.3. Konfiguration von Kernel−Optionen

Nun können Sie das Kernel Konfigurationsprogramm ihrer Wahl starten. Zur Wahl stehen:

- make xconfig

  Dies ist die X Window Version des Kernel−Konfigurationstools

- make menuconfig

  Dies ist eine curses basierte Version.

- make config

  Das ist die einfache Zeile−für−Zeile Version des Tools.

7.2.3.1. Kernel mit initrd konfigurieren

Die Verwendung von initrd erfordert die folgenden Optionen:

- File systems −> /dev filesystem support

  "/dev file system support" muss angewählt werden. Es befindet sich in der Rubrik 'File systems' (Dateisysteme) 'Automatically mount at boot' darf NICHT eingeschaltet seit, denn das mounten wird durch das /linuxrc Script erledigt.

- Block devices −> RAM disk support

  LTSP Terminals benötigen RAM−Disk support. Dies Option finden Sie in der Rubrik 'Block devices'

- Block devices −> Initial RAM disk (initrd) support

  Auch diese Option muss aktiviert werden.

- Processor type and features −> Processor family

  Sie müssen sicherstellen, dass der Kernel auf der CPU des Clients läuft. Dies kann man in der Rubrik 'Processor type and features' einstellen. SMP−support sollten Sie abschalten, es sei denn, ihr Client hat tatsächlich mehrere CPUs.

- File systems −> Network file systems −> NFS Client support

  Der Client wird sein Dateisystem via NFS mounten. Also muss diese Option aktiviert werden.

Dies waren die benötigten Optionen. Sie können noch viele Optionen abschalten, um die Größe des Kernels zu reduzieren.

7.2.3.2. Kernel ohne initrd konfigurieren

Das Konfigurieren eines Kernel ohne initrd unterscheidet sich vom Kernel mit initrd in einigen Punkten:

- Block devices −> RAM disk support

  LTSP Terminals benötigen RAM−Disk support.

- Block devices −> Initial RAM disk (initrd) support

  Dies wird abgeschaltet

- Networking options −> IP:Kernel level autoconfiguration

  Diese Option muss eingeschaltet werden. Dadurch konfiguriert der Kernel beim Booten automatisch die Ethernet−Schnittstelle eth0 anhand von Kernel−Bootparametern.

  Es ist nicht nötig die DHCP, BOOTP oder RARP Optionen zu aktivieren, da der Etherboot−Code bereits eine DHCP oder BOOTP Anfrage gemacht hat und die IP−Parameter über die Kommandozeile an den Kernel übergibt. Deswegen muss der Kernel selbst keine eigene Anfrage mehr starten.

- Network Device support −> Ethernet (10 or 100Mbit)

  Wenn man auf initrd verzichtet, muss man einen bestimmten Netzwerkkarten−Treiber auswählen, der zur im Client verwendeten Netzwerkkarte passt. Dieser Treiber MUSS fest einkompiliert werden, weil die Ethernet Schnittstelle benötigt wird, bevor das root−Dateisystem gemountet wird. Dies ist der Unterschied zu einem Kernel mit initrd.

- File systems −> /dev filesystem support

Seit LTSP Version 2.09pre2 wird **devfs** benötigt, egal ob mit oder ohne initrd.

- File systems −> Automatically mount at boot

  Wenn KEINE initrd verwendet wird, dann muss das /dev Dateisystem beim Booten vom Kernel gemountet werden. Also muss diese Option aktiviert werden.

- File systems −> Network file systems −> NFS Client support

  Der Client wird sein root−Dateisystem per NFS mounten, also ist "NFS client support" nötig.

7.2.4. Den Kernel kompilieren

Um die Arbeit zu erleichtern, befindet sich eine Kopie der `.config` Datei im ltsp_initrd_kit Paket. Sie können diese ins Verzeichnis `/usr/src/linux−2.4.9` kopieren. Aber Vorsicht, überschreiben Sie nicht die Einstellungen, die Sie gerade selbst gemacht haben.

Nachdem Sie alle gewünschten Kernel−Optionen an− und abgewählt haben, können Sie den Kernel kompilieren:

```
make dep
make clean
make bzImage
make modules
make modules_install
```

Man kann diese Kommandos auch in einer Zeile zusammenfügen:

```
make dep && make clean && make bzImage && make modules && make modules_install
```

Das doppelte & bedeutet, dass das hintere Kommando erst gestartet wird, wenn das vordere erfolgreich ausgeführt wurde.

Nach dem Kompilieren des Kernels befindet sich der neue Kernel in `/usr/src/linux−2.4.7/arch/i386/boot/bzImage`.

7.2.5. Den Kernel für Etherboot markieren (Tagging)

Damit Etherboot mit dem Kernel umgehen kann, muss er präpariert werden. Das nennt man 'Kernel tagging'. Das Tagging fügt zusätzlichen Code zum Kernel hinzu, der ausgeführt wird, bevor die Kontrolle an den Kernel übergeben wird. Das Programm für's Kernel−Tagging heißt '**mknbi−linux**'.

Das Paket ltsp_initrd_kit enthält ein shell−script mit dem Namen **buildk**, welches alle Kommandos enthält, die man benötigt, um den Kernel für's Netzwerk−Booten vorzubereiten.

Kapitel 8. lts.conf Einträge

Als wir LTSP entwickelt haben, wussten wir, dass wir mit unterschiedlicher Hardware−Ausstattung bei den Clients rechen mussten. Egal welche Kombination aus Prozessor, Netzwerk− und Grafikkarte wir heute verwendeten, wir konnten fast sicher sein, dass diese Kombination in 3 Monaten, wenn wir mehr Clients hinzufügen wollen, nicht mehr erhältlich sein würde.

Also haben wir uns einen Weg ausgedacht, wie man die Konfiguration aller Clients zentral speichert. Diese Konfigurationsdatei heißt `lts.conf` und sie befindet sich in `/opt/ltsp/i386/etc` .

Das Format von lts.conf erlaubt Standardeinstellungen (default) und davon abweichende Einstellungen für einzelne Clients. Wenn alle ihre Clients identisch sind, dann genügt es, alle Einstellungen im Abschnitt [Default] einzutragen.

8.1. lts.conf Beispiel

```
[Default]
        SERVER              = 192.168.0.254
        X_MOUSE_PROTOCOL    = "PS/2"
        X_MOUSE_DEVICE      = "/dev/psaux"
        X_MOUSE_RESOLUTION  = 400
        X_MOUSE_BUTTONS     = 3
        USE_XFS             = N
        RUNLEVEL            = 5

[ws001]
        XSERVER             = auto
        X_MOUSE_PROTOCOL    = "Microsoft"
        X_MOUSE_DEVICE      = "/dev/ttyS1"
        X_MOUSE_RESOLUTION  = 50
        X_MOUSE_BUTTONS     = 3
        X_MOUSE_BAUD        = 1200

[ws002]
        XSERVER             = XF86_Mach64

[ws003]
        RUNLEVEL            = 3
```

8.2. Verfügbare Parameter in lts.conf

8.2.1. Allgemeine Parameter

Kommentare
: Kommentar−Zeilen beginnen mit dem '#' Zeichen

LTSP_BASEDIR
: Damit gibt man das LTSP Wurzelverzeichnis an. Der Defaultwert ist `/opt/ltsp`

SERVER
: Dies ist der Server, der als XDM_SERVER, TELNET_HOST, XFS_SERVER und SYSLOG_HOST, genommen wird, falls er nicht für einzelnen Dienste extra angegeben wird. Wenn Sie also einen

Rechner haben, der als Server für alles fungiert, dann können Sie diesen hier angeben und die anderen Server−Parameter weglassen. Wenn dieser Wert nicht gesetzt wird, wird der Default−Wert **192.168.0.254** benutzt.

SYSLOG_HOST
Wenn Sie log−Meldungen an eine andere Maschine als den default−Server senden wollen, dann können Sie diese hier angeben. Wenn dieser Wert NICHT gesetzt wird, dann wird der 'SERVER' Parameter genommen, wie oben beschrieben.

NFS_SERVER
Hier können Sie die IP−Adresse des NFS−Servers angeben der zum Mounten des `/home` Verzeichnisses benutzt werden soll. Wenn dieser Wert nicht gesetzt wird, dann wird der **SERVER** Parameter genommen.

USE_NFS_SWAP
Setzen Sie diese Option auf **Y** wenn Sie NFS−Swap verwenden wollen. Der Defaultwert ist **N**

SWAPFILE_SIZE
Hier können Sie die Größe der Swap−Datei angeben. Der Defaultwert ist **64m**.

SWAP_SERVER
Die Swap−Datei kann auf einem beliebigen NFS−Server liegen. Hier können Sie dessen IP−Adresse angeben. Der Defaultwert entspricht dem Wert, der mit NFS_SERVER angegeben wird.

NFS_SWAPDIR
Der Pfad, der vom NFS−Server für Swap−Dateien freigegeben wurde. Der Defaultwert ist `/var/opt/ltsp/swapfiles`. Achten Sie darauf, dass dieses Verzeichnis beim NFS−Server in `/etc/exports` eingetragen ist.

TELNET_HOST
Wenn der Client als Text−Terminal genutzt wird, dann kann man hier angeben, zu welchem Server eine Telnet−Verbindung aufgebaut werden soll. Wenn dieser Wert nicht gesetzt ist, wird die Angabe von **SERVER** genommen.

DNS_SERVER
Anhand dieser Angabe wird die Datei resolv.conf des Clients generiert.

SEARCH_DOMAIN
Auch dieser Wert landet in resolv.conf des Clients.

MODULE_01 *bis* ***MODULE_10***
Bis zu 10 Kernel−Module können durch diese Einträge geladen werden. Die Einträge entsprechen dem, was man hinter insmod angeben würde. Beispiel:

```
MODULE_01   = uart401.o
MODULE_02   = sb.o io=0x220 irq=5 dma=1
MODULE_03   = opl3.o
```

Wenn man hier einen absoluten Pfad angibt, wird **insmod** benutzt um das Modul zu laden. Ansonsten wird **modprobe** verwendet.

RAMDISK_SIZE

Wenn ein Client bootet, erzeugt er eine RAM−Disk und mountet sie nach /tmp. Die Größe dieser RAM−Disk kann man mit diesem Parameter beeinflussen. Angaben werden in kByte(1024Bytes)−Einheiten gemacht. Um eine 1MB RAM−Disk zu erzeugen, schreiben Sie **RAMDISK_SIZE = 1024**.

Wenn Sie die Größe der RAM−Disk hier ändern, müssen Sie sie auch im Kernel verändern. Entweder fest einkompiliert oder beim Einsatz von Etherboot oder Netboot kann man die Größe angeben, wenn man den Kernel mit mknbi−linux 'taggt'.

Der Defaultwert ist 1024 (= 1 MB)

RCFILE_01 *bis* ***RCFILE_10***

Das rc.local Script kann zusätzliche Startscripte ausführen. Legen Sie diese Scripte einfach ins etc/rc.d Verzeichnis und geben Sie hier den Namen ihres Scripts an.

SOUND

Wenn das LTSP Sound Paket installiert wurde, dann muss dieser Wert auf **Y** stehen, damit das Script **rc.sound** gestartet wird. rc.sound konfiguriert die Soundkarte und den Sound−Deamon (Sound−Server). Der Defaultwert ist **N**.

8.2.2. X Window Einstellungen

XDM_SERVER

Wenn die XDM Anfrage an einen anderen als den Standard−Server gesendet werden soll, dann kann man diesen hier angeben. Lässt man diese Parameter weg, dann wird der Wert von 'SERVER' genommen.

XSERVER

Hier kann man den X Server angeben, der auf dem Client verwendet werden soll. Bei PCI und AGP Karten sollte dieser Parameter nicht nötig sein. Das rc.local Script ist üblicherweise in der Lage, die Grafikkarte automatisch zu erkennen. Um dies zu verdeutlichen, kann man den Wert auch auf **auto** stellen.

Für ISA und VLB Grafikkarten oder um einen speziellen Treiber zu wählen, kann man hier das XFree86 4.x Treiber−Modul oder den Xfree86 3.3.6 X−Server angeben.

Wenn der Wert mit **XF86_** beginnt wird XFree86 3.3.6 verwendet. Andernfalls wird XFree86 4.x.x genommen. Der Defaultwert ist **auto**.

X_MODE_0 *bis* ***X_MODE_2***

Bis zu 3 Modelines oder Bildschirm−Auflösungen kann man hier angeben. Für jeden Mode kann man entweder eine Auflösung oder eine komplette Modeline angeben.

```
X_MODE_0 = 800x600

   oder

X_MODE_0 = 800x600 60.75 800 864 928 1088 600 616 621 657 −HSync −VSync
```

Wenn keine X_MODE_x Angaben gemacht werden, dann werden Standard−Modelines und die Auflösungen 1024x768, 800x600 und 640x480 verwendet.

Wenn eine oder mehrere X_MODE_x Angaben gemacht werden, dann wird keine der Standard−Modelines mehr verwendet.

X_MOUSE_PROTOCOL

Jeder Wert, der von XFree86 als Pointer Protocol bekannt ist, kann hier verwendet werden. Typische Werte sind z.B. "Microsoft" und "PS/2" oder "IMPS/2" für PS/2 Wheelmäuse (Intellimouse). Der Defaultwert ist **"PS/2"**.

X_MOUSE_DEVICE

Dies ist die Gerätedatei für den Maus−Anschluss. Bei seriellen Mäusen z.B. **/dev/ttyS0** oder **/dev/ttyS1**. Beim PS/2 Anschluss ist es **/dev/psaux**. Der Defaultwert ist **/dev/psaux**.

X_MOUSE_RESOLUTION

Dies ist der 'Resolution'(Auflösung)−Wert für die Maus in der **XF86Config** Datei. Ein typischer Wert für serielle Mäuse ist **50** und für eine PS/2 Maus **400**. Der Defaultwert ist **400**.

X_BUTTONS

Hier kann man die Anzahl der Mausknöpfe angeben. Üblicherweise **2** oder **3**, bei Wheelmäusen **5**. Der Defaultwert ist **3**.

X_MOUSE_EMULATE3BTN

Durch den Wert **Y** kann man die mittlere Maustaste bei 2−Tasten−Mäusen simulieren, indem man beide Tasten gleichzeitig drückt. Der Defaultwert ist **N**.

X_MOUSE_BAUD

Hier kann man die Baudrate für serielle Mäuse angeben Der Defaultwert ist **1200**.

X_COLOR_DEPTH

Damit kann man die Farbtiefe angeben. Mögliche Werte sind **8** = 256 Farben, **15**, **16** = 65536 Farben , **24** und **32** = 4,2 Milliarden Farben . Nicht alle X−Server bieten alle Farbtiefen. Der Defaultwert ist **16**

USE_XFS

Bei den Fonts (Zeichensätzen) kann man entweder einen Font−Server verwenden oder die Zeichensätze per NFS lesen. Der Font−Server soll die Verwaltung von Fonts an einer zentralen Stelle erleichtern. Allerdings gab es Probleme bei mehr als 40 Clients. Die 2 möglichen Werte lauten hier **Y** und **N**. Der Defaultwert ist **N**. Wenn Sie einen Fontserver verwenden wollen, dann können Sie mit **XFS_SERVER** die IP des Fontservers angeben.

XFS_SERVER

Wenn man einen X Fontserver benutzt, dann kann man hier seine IP Adresse angaben. Wenn diese Variable nicht gesetzt ist, wird der Wert von **SERVER** (s.o.) genommen.

X_HORZSYNC

Hier kann man den **HorizSync** Parameter für Xfree86 bzw. den Monitor angeben. Der Defaultwert ist **"31−62"**.

X_VERTREFRESH

Hier kann man den **VertRefresh** Parameter für Xfree86 bzw. den Monitor angeben. Der Defaultwert ist **"55−90"**.

XF86CONFIG_FILE
Wenn Sie eine eigenen komplette XF86Config Datei vorbereitet haben, dann können Sie diese im Verzeichnis **/opt/ltsp/i386/etc** ablegen. Geben Sie der Datei einen aussagekräftigen Namen, z.B. XF86Config.ws004.

```
XF86CONFIG_FILE = XF86Config.ws004
```

8.2.3. Touch−Screen Parameter

USE_TOUCH
Wenn der Client über einen Touch Screen verfügt, können Sie dies hier durch **Y** aktivieren. Der Defaultwert ist **N**.

X_TOUCH_DEVICE
Ein TouchScreen arbeitet ähnlich wie eine Maus und wird üblicherweise an einem seriellen Port angeschlossen. Diesen Anschluss können Sie hier angeben, z.B. **/dev/ttyS0**. Für diesen Eintrag existiert kein Defaultwert.

X_TOUCH_MINX
Kalibrierungswert für einen EloTouch Touch−Screen. Defaultwert: **433**.

X_TOUCH_MAXX
Kalibrierungswert für einen EloTouch Touch−Screen. Defaultwert: **3588**.

X_TOUCH_MINY
Kalibrierungswert für einen EloTouch Touch−Screen. Defaultwert: **569**.

X_TOUCH_MAXY
Kalibrierungswert für einen EloTouch Touch−Screen. Defaultwert: **3526**.

X_TOUCH_UNDELAY
Kalibrierungswert für einen EloTouch Touch−Screen. Defaultwert: **10**.

X_TOUCH_RPTDELAY
Kalibrierungswert für einen EloTouch Touch−Screen. Defaultwert: **10**.

8.2.4. Parameter für lokale Anwendungen

LOCAL_APPS
Wenn Anwendungen lokal auf einem Client laufen sollen, stellen Sie diesen Wert auf **Y**. Diverse weitere Einstellungen müssen am Server vollzogen werden, um Client−lokale Anwendungen zu ermöglichen. Lesen Sie hierzu das Kapitel 'Lokale Anwendungen' im LTSP Handbuch. Der Defaultwert ist **N**.

NIS_DOMAIN

Wenn Sie (Client−)lokale Anwendungen nutzen wollen, benötigen Sie einen NIS−Server in ihrem Netz. Hier können Sie den Namen der NIS−Domain angeben. Der Name muss mit dem Namen, der im NIS−Server eingestellt ist übereinstimmen. Eine NIS−Domain ist nicht dasselbe wie eine Internet−Domain. Der Defaultwert ist **ltsp**.

NIS_SERVER
Geben Sie hier die IP−Adresse des NIS−Servers an, wenn der Client keinen Broadcast senden soll, um einen NIS−Server zufinden.

8.2.5. Tastatur Einstellungen

Alle Dateien zur (internationalen) Tastatur−Unterstützung sind im LTSP−Verzeichnisbaum vorhanden. Deshalb ist die Konfiguration der Tastatur nur noch eine Sache von Einträgen in der XFree86−Konfiguration. Hier stehen diverse Parameter zur Verfügung.

Die Namen und möglichen Werte der folgenden Parameter stammen aus der Dokumentation von XFree86. Alle für XFree86 gültigen Werte sind auch hier gültig.

Wir würden in Zukunft gerne weitere Abschnitte hinzufügen die auflisten, welche konkreten Einstellungen für die einzelnen internationalen Tastaturen benötigt werden. Wenn es ihnen gelingt, ihre länderspezifische Tastatur einzustellen, dann informieren Sie bitte das LTSP Entwickler−Team.

XkbTypes
Der Defaultwert ist das Wort '**default**'.

XkbCompat
Der Defaultwert ist das Wort '**default**'.

XkbSymbols
Der Defaultwert ist '**us(pc101)**'. Dieser Defaultwert ist auch für deutsche Tastaturen o.k.

XkbModel
Der Defaultwert ist '**pc101**'. Dieser Defaultwert ist auch für deutsche Tastaturen o.k.

XkbLayout
Der Defaultwert ist '**us**'. Der Wert für eine deutsche Tastatur lautet '**de**'.

8.2.6. Drucker Konfiguration

Maximal 3 Drucker können an einem Client angeschlossen werden. Eine Kombination aus seriellen und parallelen Druckern kann mit den folgenden Einträgen in **lts.conf** konfiguriert werden.

PRINTER_0_DEVICE
Das Device (Anschluss) des ersten Druckers. Beispiele: **/dev/lp0 /dev/ttyS0 /dev/ttyS1**

PRINTER_0_TYPE
Der Druckertyp. Gültige Werte sind '**P**' für Parallel−Port Drucker, und '**S**' für Serielle Drucker.

PRINTER_0_PORT

Die TCP/IP Port−Nummer für den Druck−Server−Prozess auf dem Client (HP JetDirect Protokoll). Defaultwert ist '**9100**'

PRINTER_0_SPEED
Für serielle Drucker kann man hier die Baudrate einstellen. Defaultwert ist '**9600**' .

PRINTER_0_FLOWCTRL
Für serielle Drucker kann man hier die Art der Flusskontrolle angeben. Entweder '**S**' für Software (XON/XOFF) Flusskontrolle , oder '**H**' für Hardware (CTS/RTS) Flusskontrolle. Wenn nichts angegeben wird, gilt '**S**' als Defaultwert.

PRINTER_0_PARITY
Für serielle Drucker kann man hier die Parität angeben. Zur Auswahl stehen: '**E**'−Even(Gerade) '**O**'−Odd(Ungerade) '**N**'−None(keine Parität) Wenn nichts angegeben wird, gilt '**N**' als Defaultwert.

PRINTER_0_DATABITS
Für serielle Drucker kann man hier die Anzahl der Datenbits angeben. Zur Auswahl stehen: '**5**', '**6**', '**7**' und '**8**'. Wenn nichts angegeben wird, gilt '**8**' als Defaultwert.

PRINTER_1_DEVICE
Device (Anschluss) des zweiten Druckers

PRINTER_1_TYPE
Typ des zweiten Druckers

PRINTER_1_PORT
TCP/IP−Port des zweiten Druckers

PRINTER_1_SPEED
Baudrate des zweiten Druckers (seriell)

PRINTER_1_FLOWCTRL
Flusskontrolle des zweiten Druckers(seriell)

PRINTER_1_PARITY
Parität des zweiten Druckers(seriell)

PRINTER_1_DATABITS
Anzahl Datenbits des zweiten Druckers(seriell)

PRINTER_2_DEVICE
Device−Name des dritten Druckers

PRINTER_2_TYPE
Typ des dritten Druckers

PRINTER_2_PORT
TCP/IP−Port des dritten Druckers

PRINTER_2_SPEED
Baudrate des dritten Druckers (seriell)

PRINTER_2_FLOWCTRL
Flusskontrolle des dritten Druckers (seriell)

PRINTER_2_PARITY
Parität des dritten Druckers (seriell)

PRINTER_2_DATABITS
Anzahl Datenbits des dritten Druckers (seriell)

Kapitel 9. Lokale Anwendungen

In einer LTSP−Umgebung hat man die Wahl, ob die Programme lokal auf dem Client oder remote auf dem Server laufen sollen.

Die Anwendungen auf dem Server laufen zu lassen ist die einfachere Variante. Dabei läuft die Anwendung auf dem Prozessor und im Arbeitsspeicher des Servers, die Interaktion mit dem Benutzer, also Bildschirmausgabe und Tastatur− und Mauseingabe, erfolgt beim Client.

Dies ist eine eingebaute Fähigkeit des X Window Systems. Der Client funktioniert wie ein X Window Terminal.

Wenn eine Anwendung auf dem Client laufen soll, dann benötigt der Client−PC einige Informationen über den Benutzer:

- Benutzer ID (UID)
- Primäre Benutzergruppe (GID)
- Das Heimatverzeichenis des Benutzers

LTSP benutzt Network Information Service − NIS (früher auch ***Yellow Pages*** genannt), um den Clients diese Informationen zur Verfügung zu stellen.

9.1. Vorteile von lokalen Anwendungen

- Reduzierte Server−Last. In großen Netzen mit speicherintensiven Anwendungen, wie z.B. Netscape, kann es die Performance verbessern, solange der Client über genung CPU−Leistung und RAM verfügt um die Anwendung(en) zu bewältigen.
- Sog. "runaway" Prozesse, also Prozesse, die aufgrund von Fehlern viel CPU−Leistung und RAM an sich reißen, wirken sich nicht störend auf andere Benutzer aus.
- Soundunterstützung ist viel einfacher zu konfigurieren, wenn die Anwendung, die den Sound erzeugt, lokal auf dem Client läuft.

9.2. Dinge, die man beim Einsatz von lokalen Anwendungen beachten sollte

Lokale Anwendungen erfordern deutlich mehr Konfigurationsaufwand.

- Lokale Anwendungen erfordern mehr Leistung von der Client Hardware. Man benötigt mehr RAM und eine schnellere CPU. Empfehlung: 64MB RAM oder mehr.
- NIS − um Anwendungen auf dem Client laufen zu lassen, muss man sich zuerst gegenüber dem Client identifizieren. Der Client muss wissen, wer Sie sind. Dazu wird eine Art von Passwort−Authentifizierung benötigt. Wir haben NIS gewählt um Authentifizierung übers Netzwerk zu ermöglichen.
- Für lokale Anwendungen müssen zusätzliche Verzeichnisse vom Server per NFS exportiert werden.
- Anwendungen starten langsamer, denn sie müssen via NFS gelesen werden, was außerdem auch mehr Netzwerk−Traffic verursacht. Weil jede Kopie eines Programms auf seiner eigenen CPU läuft, verzichtet man auf den Vorteil von gemeinsamen (shared) Code−Segmenten im RAM. Gemeinsame Code−Segmente sorgen dafür, dass weitere Instanzen eines Programms deutlich schneller starten.

9.3. Server Konfiguration für lokale Anwendungen

9.3.1. Einträge in lts.conf

In der `lts.conf` Datei müssen einige Einträge gemacht werden:

LOCAL_APPS
: Dieser Wert muss auf **Y** stehen. Dadurch werden folgende Vorgänge beim Booten des Clients ausgelöst:

1. Das `/home` Verzeichnis des Servers wird per NFS gemountet.
2. Die Datei `/var/yp/nicknames` wird auf dem Client angelegt.
3. Der **portmapper** wird auf dem Client gestartet.
4. **xinetd** wird auf dem Client gestartet.
5. Die Datei `/etc/yp.conf` wird auf dem Client erzeugt.
6. Das **domainname** Kommando wird mit dem Wert von **NIS_DOMAIN** aus lts.conf ausgeführt.
7. Das **ypbind** wird ausgeführt.

NIS_DOMAIN
: NIS−Server und alle zugehörigen NIS−Clients müssen zu derselben NIS−Domain gehören. (NIS−Domains haben nichts mit DNS/Internet Domains zu tun)

NIS_SERVER
: Ein NIS−Client versucht entweder eine Verbindung mit einem bestimmten NIS−Server herzustellen, oder er sendet ein Broadcast ins Netz und wartet, dass sich ein Server meldet. Wenn Sie einen NIS−Server angeben wollen , dann tragen Sie seine IP−Adresse hier ein.

9.3.2. Network Information Service − NIS

NIS ist ein Client/Server Dienst. Auf dem Server läuft ein daemon (Hintergrundprozess/Dienst), welcher Anfragen von den Clients entgegen nimmt. Dieser Prozess heißt **ypserv**.

Auf dem Client gibt es den Prozess **ypbind**. Wenn der Client Informationen über den User benötigt, z.B. zur Passwort−Überprüfung, oder den Namen des Heimatverzeichnisses, dann wird ypbind benutzt, um eine Verbindung mit ypserv auf dem Server herzustellen.

Wenn Sie bereits NIS in ihrem Netzwerk verwenden, ist es nicht nötig den LTSP−Server als zusätzlichen NIS−Server zu betreiben. Tragen Sie einfach in lts.conf für NIS_DOMAINNAME und NIS_SERVER Werte ein, die zu ihrem Netz passen.

Wenn Sie in ihrem Netz bisher noch kein NIS verwenden, müssen Sie **ypserv** auf dem Server konfigurieren & starten.

Umfassende Informationen über die Installation eines NIS−Servers finden Sie im ***The Linux NIS(YP)/NYS/NIS+ HOWTO*** des LDP (Linux Documentation Project). Siehe "Weitere Informationsquellen" am Ende dieses Handbuchs.

9.4. Konfiguration von Anwendungen

Damit eine Anwendung auf dem Client laufen kann, müssen Sie alle Teile der Anwendung dahin kopieren, wo der Client Sie auch finden kann.

Bei älteren LTSP Versionen (2.08 und früher) wurden viele Verzeichnisse des Servers per NFS exportiert und vom Client gemountet. Verzeichnisse wie z.B. `/bin`, `/usr/bin`, `/lib` und `/usr` wurden exportiert.

Das Problem bei dieser Vorgehensweise ist, dass es nur funktioniert, wenn Client und Server die gleiche Hardware−Architektur verwenden. Sogar der Unterschied zwischen einem Server mit Pentium II (i686) und einem Client mit Pentium I (i586) kann zu einem Problem werden, da der Server wahrscheinlich i686 Libraries (Programm−Bibliotheken) hat und keine i386, i486 oder i586 libraries.

Also ist der sauberste Weg, einen kompletten Verzeichnisbaum zu haben, der alle Programme und Libraries enthält, den der Client benötigt, unabhängig von den Programme und Libraries des Servers.

Alle Teile , die eine (Client−lokale) Anwendung benötigt, müssen in diesem Verzeichnisbaum abgelegt werden. Eines der LTSP−Pakete, die auf der LTSP−Website zum Download zu Verfügung stehen, ist das Paket local_netscape. Es installiert viele Dateien ins `/opt/ltsp/i386/usr/local/netscape` Verzeichnis. Darin enthalten sind Java−Klassen, Hilfe−Texte, Programmdateien und Scripte.

Netscape benötigt keine zusätzlichen System−Libraries, deswegen muss in `/opt/ltsp/i386/lib` nichts hinzugefügt werden. Viele andere Anwendungen benötigen jedoch zusätzliche Libraries.

Also, wie finden wir heraus, welche Libraries benötigt werden ? Dazu lässt sich das **ldd** Kommando gut verwenden.

Nehmen wir an, Sie wollen eine bestimmte Anwendung als Client−lokale Anwendung einrichten. Als Beispiel nehmen wir das Programm **gaim**. **gaim** ist ein AOL Instant Messenger Client, er ermöglicht die Kommunikation mit anderen AIM Benutzern im Internet.

Als erstes müssen Sie die **gaim** Binärdatei finden. Bei RedHat 7.2 liegt sie im `/usr/bin` Verzeichnis. Die Tools **type** und **which** sind für eine solche Suche hilfreich.

Wenn Sie die **gaim**Binärdatei gefunden haben, können Sie mit **ldd** die benötigten Libraries herausfinden.

```
[jam@server /]$ ldd /usr/bin/gaim
        libaudiofile.so.0    => /usr/lib/libaudiofile.so.0 (0x40033000)
        libm.so.6            => /lib/i686/libm.so.6 (0x40051000)
        libnsl.so.1          => /lib/libnsl.so.1 (0x40074000)
        libgnomeui.so.32     => /usr/lib/libgnomeui.so.32 (0x4008a000)
        libart_lgpl.so.2     => /usr/lib/libart_lgpl.so.2 (0x4015d000)
        libgdk_imlib.so.1    => /usr/lib/libgdk_imlib.so.1 (0x4016c000)
        libSM.so.6           => /usr/X11R6/lib/libSM.so.6 (0x40191000)
        libICE.so.6          => /usr/X11R6/lib/libICE.so.6 (0x4019a000)
        libgtk-1.2.so.0      => /usr/lib/libgtk-1.2.so.0 (0x401b1000)
        libdl.so.2           => /lib/libdl.so.2 (0x402df000)
        libgdk-1.2.so.0      => /usr/lib/libgdk-1.2.so.0 (0x402e3000)
        libgmodule-1.2.so.0  => /usr/lib/libgmodule-1.2.so.0 (0x40319000)
        libXi.so.6           => /usr/X11R6/lib/libXi.so.6 (0x4031d000)
        libXext.so.6         => /usr/X11R6/lib/libXext.so.6 (0x40325000)
        libX11.so.6          => /usr/X11R6/lib/libX11.so.6 (0x40333000)
        libgnome.so.32       => /usr/lib/libgnome.so.32 (0x40411000)
```

```
        libgnomesupport.so.0 => /usr/lib/libgnomesupport.so.0 (0x40429000)
        libesd.so.0          => /usr/lib/libesd.so.0 (0x4042e000)
        libdb.so.2           => /usr/lib/libdb.so.2 (0x40436000)
        libglib-1.2.so.0     => /usr/lib/libglib-1.2.so.0 (0x40444000)
        libcrypt.so.1        => /lib/libcrypt.so.1 (0x40468000)
        libc.so.6            => /lib/i686/libc.so.6 (0x40495000)
        libz.so.1            => /usr/lib/libz.so.1 (0x405d1000)
        /lib/ld-linux.so.2   => /lib/ld-linux.so.2 (0x40000000)
```

Das obere Listing zeigt alle dynamisch gelinkten Libraries, die von **gaim** benötigt werden.

Die meisten Programme, die shared Libraries verwenden, benötigen den dynamischen Loader **ld−linux** um jede der Libraries zu finden und zu laden. Manche Programme laden die Libraries jedoch manuell mit dem **dlopen()** Systemaufruf. Bei solchen Anwendungen wird **ldd** die benötigten Libraries nicht anzeigen. Für solche Fälle kann man das **strace** Kommando verwenden. strace wird alle **dlopen()** Aufrufe mit den zugehörigen Libraries anzeigen.

Hat man die Liste der benötigten Libraries ermittelt, müssen diese an die richtigen Stellen in den `/opt/ltsp/i386` Baum kopiert werden.

9.5. Lokale Anwendungen starten

X Window Programme starten üblicherweise dort, wo der Fenstermanager läuft. Wenn der Fenstermanager auf dem Server läuft und seine Ausgaben auf den Client schickt, dann wird jede gestartete Anwendung ebenfalls auf dem Server laufen und die Ausgaben zum Client schicken.

Der Trick bei lokalen Anwendungen besteht darin, dass der Server dem Client sagt, ein Programm zu starten. Dies wird üblicherweise mit dem **rsh** Kommando gemacht.

Hier ein Beispiel wie man **gaim** auf dem Client startet:

```
HOST=`echo $DISPLAY | awk -F: '{ print $1 }'`
rsh ${HOST} /usr/bin/gaim -display ${DISPLAY}
```

Das obige Beispiel kann man in einem **xterm** Fenster eingeben, oder in ein Shell−Script schreiben, welches dann durch den Klick auf ein Icon ausgelöst werden kann.

Der lokale Start von Netscape läuft ähnlich ab, allerdings muss vor dem Start eine Umgebungs−Variable gesetzt werden.

```
HOST=`echo $DISPLAY | awk -F: '{ print $1 }'`
rsh ${HOST} MOZILLA_HOME=/usr/local/netscape \
       /usr/local/netscape/netscape -display ${DISPLAY}
```

Kapitel 10. Konfigurations−Beispiele

Fast alle Eigenschaften des Clients können durch einen Eintrag in der Datei `lts.conf` festgelegt werden. Diese Datei liegt gewöhnlich im Verzeichnis `/opt/ltsp/i386/etc`.

10.1. Serielle Maus

Beispieleintrag in `lts.conf` für eine serielle Standard−Maus mit 2 Tasten:

```
X_MOUSE_PROTOCOL    = "Microsoft"
X_MOUSE_DEVICE      = "/dev/ttyS0"
X_MOUSE_RESOLUTION  = 400
X_MOUSE_BUTTONS     = 2
X_MOUSE_EMULATE3BTN = Y
```

10.2. PS/2 Wheel−Maus

Beispieleintrag in `lts.conf` für eine Intellimouse (auch für Logitech M−BA47 o.ä.):

```
X_MOUSE_PROTOCOL    = "IMPS/2"
X_MOUSE_DEVICE      = "/dev/psaux"
X_MOUSE_RESOLUTION  = 400
X_MOUSE_BUTTONS     = 5
X_ZAxisMapping      = "4 5"
```

10.3. USB Drucker an ThinkNic

Der ThinkNIC−Client hat einen USB−Anschluss, der für einen Drucker benutzt werden kann. Beispieleintragungen für solch einen Drucker in `lts.conf`:

```
MODULE_01           = usb−ohci
MODULE_02           = printer
PRINTER_0_DEVICE    = /dev/usb/lp0
PRINTER_0_TYPE      = S
```

10.4. Einen passenden Xserver von XFree86 3.3.6 konfigurieren

Per Voreinstellung wird XFree86 4.1.0 für den Client eingerichtet. Für ältere Graphikkarten, die von der Version 4.1.0 nicht mehr unterstützt werden, muss zunächst das richtige Xserver−Paket der Version 3.3.6 heruntergeladen und installiert werden. Dann muss man noch die richtigen Einträge in der Datei `lts.conf` vornehmen. Beispieleinträge für den Start des **SVGA**− Xservers (dieser unterstützt sehr viele alte Karten):

```
XSERVER             = XF86_SVGA
```

Kapitel 11. Weitere Informationsquellen

11.1. Im Internet

1. Die LTSP Home−Page

   www.LTSP.org

2. Diskless−Nodes HOW−TO für Linux

   www.linuxdoc.org/HOWTO/Diskless−HOWTO.html

3. Etherboot Home−Page

   etherboot.sourceforge.net

4. The Rom−O−Matic Site

   www.Rom−O−Matic.net

5. XFree86 Maus−Unterstützung

   www.xfree86.org/current/mouse.html

6. XFree86−Video−Timings−HOWTO

   www.linuxdoc.org/HOWTO/XFree86−Video−Timings−HOWTO.html

7. Das Linux NIS(YP)/NYS/NIS+ HOWTO

   www.linuxdoc.org/HOWTO/NIS−HOWTO.html

11.2. Gedruckte Publikationen

1. Managing NFS and NIS
   Hal Stern
   O'Reilly & Associates, Inc.
   1991
   ISBN 0−937175−75−7

2. TCP/IP Illustrated, Volume 1
   W. Richard Stevens
   Addison−Wesley
   1994
   ISBN 0−201−63346−9

3.

X Window System Administrator's Guide
Linda Mui and Eric Pearce
O'Reilly & Associates, Inc.
1993
ISBN 0−937175−83−8
(Volume 8 of the The Definitive Guides to the X Window System)