Thin Clients

Michael Hartmann
<michael.hartmann@as-netz.de>

Augsburger Linux–Infotag 2006

25. März 2006

Inhalt

Definition

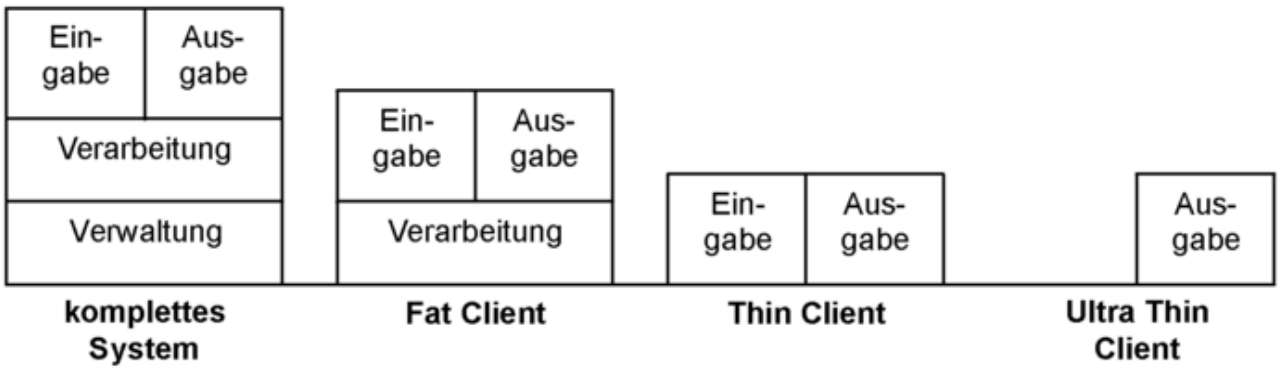
Definition

Thin Client: Endgerät eines Netzwerkes, dessen funktionale Ausstattung (größtenteils) auf Ein- und Ausgabe beschränkt ist.

Ultra Thin, Thin und Fat Clients

- Ultra Thin Client:
  - Ausgabe
- Thin Client:
  - Eingabe
  - Ausgabe
- Fat Client:
  - Eingabe
  - Ausgabe
  - Verarbeitung

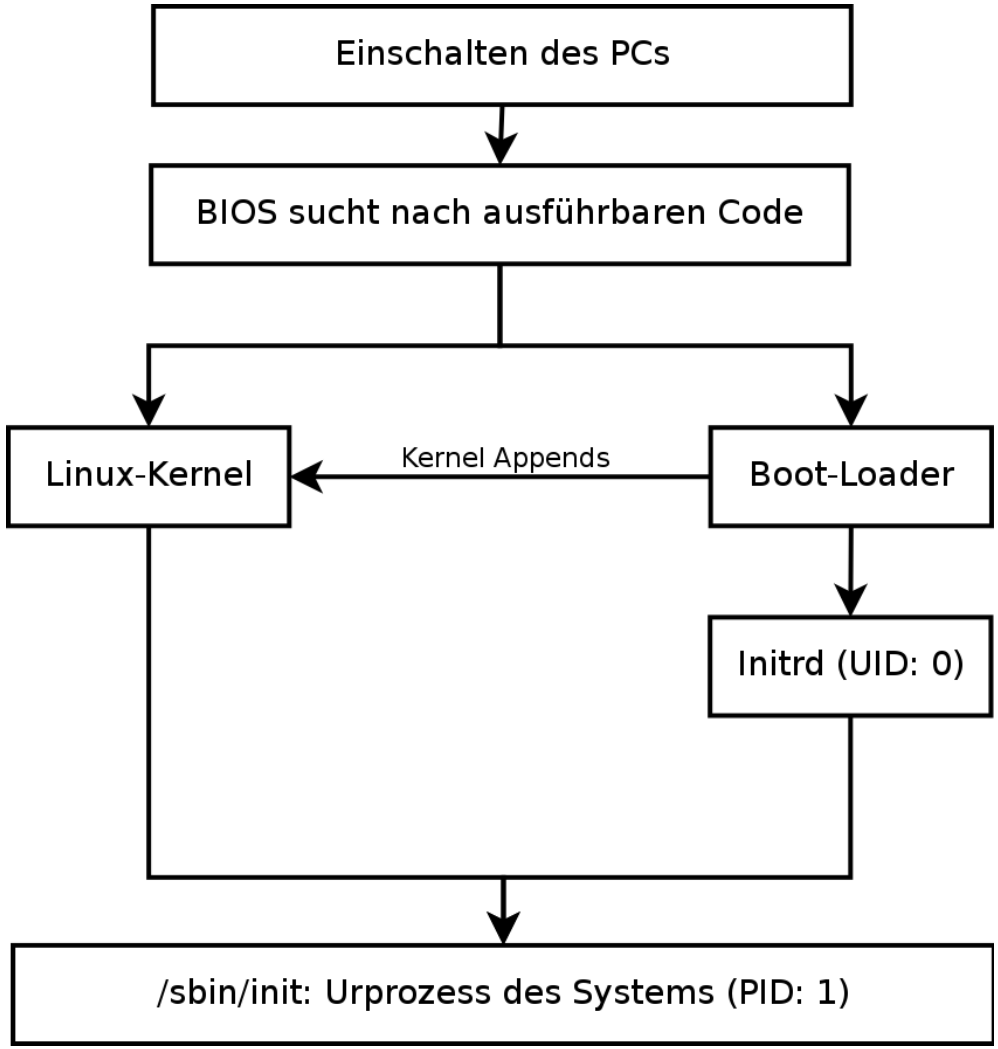
- Komplettsystem:
  - Eingabe
  - Ausgabe
  - Verarbeitung
  - Verwaltung
- weitere Schattierungen denkbar



Vor- und Nachteile von Thin/Fat Clients und Komplettsystemen

	Thin Client	Fat Client	Komplettsystem
Hardwareanforderung	niedrig	mittel	hoch
Netzwerklast	hoch	mittel	niedrig
Speicherung von Daten	nein	nein	ja
„Sicherheit“	hoch	hoch	niedrig
Einrichtungsaufwand	hoch	hoch	niedrig
Flexibilität	niedrig	mittel	hoch
Skalierbarkeit	hoch	mittel	niedrig

Bootvorgang eines Linux-Systems allgemein



Bootvorgang eines Linux-Systems mit initrd

1. Einschalten des PCs
2. BIOS sucht nach ausführbaren Code
3. Boot-Loader lädt Kernel und Initrd
4. Kernel mountet Initrd als Ramdisk
5. /linuxrc wird ausgeführt (mit UID 0), ...
6. ...mountet das „echte“ root-Dateisystem ...
7. ...und hängt es nach / um (pivot_root)
8. init (/sbin/init) wird ausgeführt (PID 1)
9. initrd wird unmounted

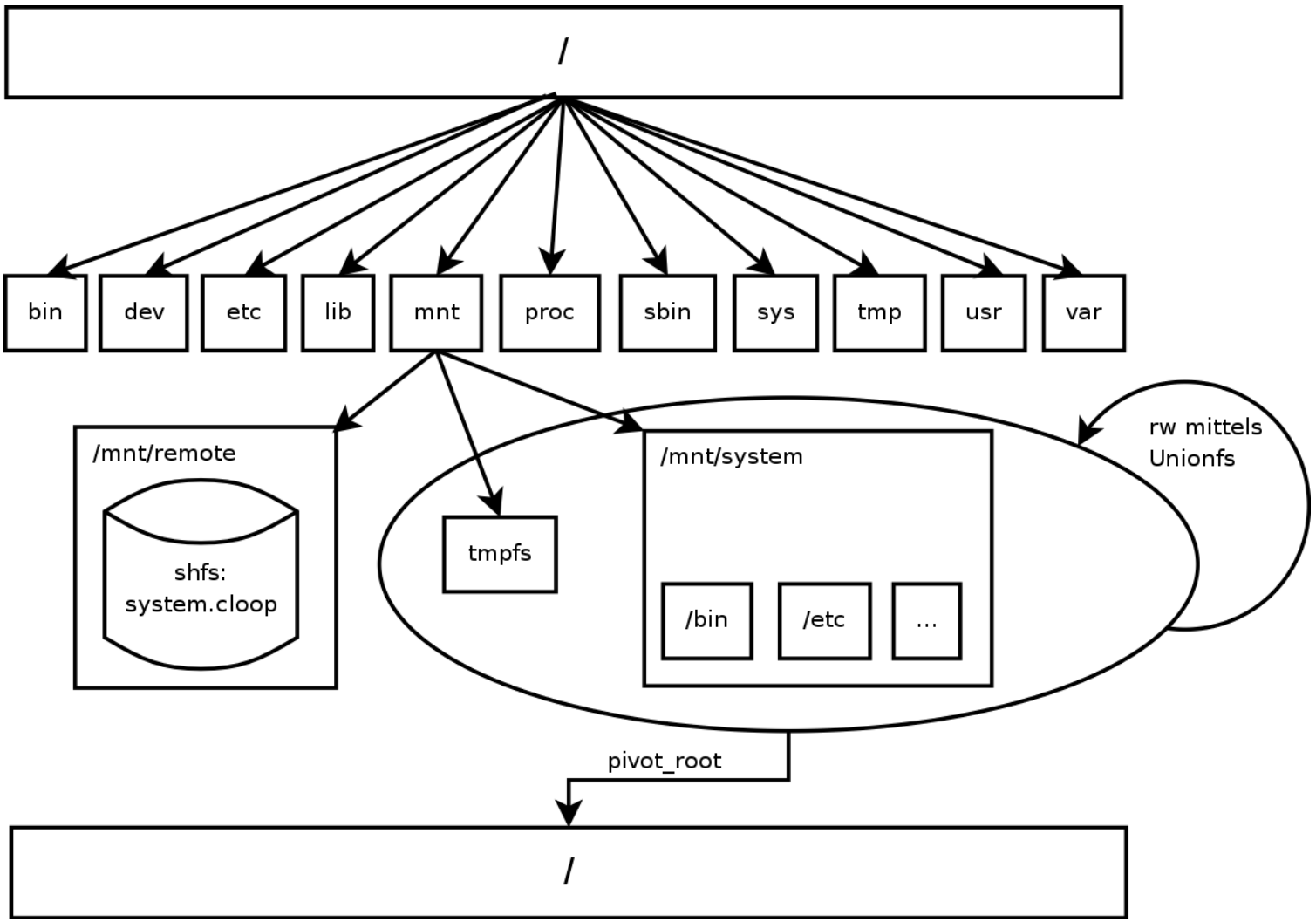
Ziele

- festplattenlos („Diskless-Client“):
  - Netzwerk als Datenspeicher
- flexibel:
  - Auswahl zwischen Arbeiten auf lokalem oder entfernten Rechner
  - automatische Serversuche
  - möglichst hardwareunabhängig (allerdings: x86-Plattform)
- einfach. . .
  - . . . anzupassen
  - . . . einzurichten
  - . . . benutzbar

Idee

1. Kernel und Initrd werden über das Netzwerk geladen (z.B. durch PXE)
2. Root-Dateisystem ist Ramdisk (cramfs) mit Minisystem (busybox)
3. /sbin/init wird gestartet:
   1. lädt notwendige Netzwerkmodule (im Moment nur für Netzwerkkarten (Ethernet))
   2. statische Netzwerkkonfiguration oder DHCP (udhcpc)
   3. automatische Suche nach Server mittels Multicast (ncp) oder über Kernel-Kommandozeile (Kernel-Append)
   4. Mounten des Ordners mit Thin Client Dateien (cloop-Image) des Servers mittels SSH (shfs)
   5. Mounten des „cloop-Dateisystems“, das späteres Root-Dateisystem enthält
   6. späteres Root-Dateisystem durch unionfs schreibbar machen . . .
   7. . . . und mittels pivot_root nach / umhängen
   8. /sbin/init des Thin Client Systems starten

Änderung der Dateisystemshierarchie



Erstellen der Initrd

- cramfs als Dateisystem der Ramdisk:
  - klein, da komprimiert ($\leq$ 4 MiB)
  - allerdings nicht schreibbar
- Erstellen eines „Mini-Linux-System“ mit ...
  - wichtigsten Systemtools (u.a.: mount, shell, ssh)
  - benötigten Bibiliotheken
  - Kernel-Modulen (u.a.: Netzwerkkarten, unionfs, shfs, cloop)
  - statischen Gerätedateien (/dev)
  - Mountpunkte (/proc, /sys, /mnt)
- busybox als platzsparende Alternative:
  - beinhaltet wichtigste UNIX-Tools (z.B.: shell, pivot_root, ls ...)
  - nur ein Binary: sehr geringer Overhead
  - einfach und schnell anpassbar

Erstellen des cloop-Dateisystems

1. Installieren eines Debian-Systems
2. Installieren der benötigten Pakete
3. Konfiguration des Systems
4. evtl. automatische Konfiguration des Systems (X-Autokonfiguration)
5. Ändern von /etc/fstab
6. Aufräumen des Systems:
   - alte Logdateien
   - Verlaufsdateien (z.B.: .bash_history)
   - heruntergeladene Dateien/Pakete
7. Speichern des Debian-Systems in einem ext2-Dateisystem
8. Kompression des ext2-Dateisystems mittels cloop (create_compressed_fs)

proc und sysfs mounten

/sbin/init

```
echo -n "Mounting proc (/proc)... "
mount -nt proc none /proc
check $?

echo -n "Mounting sysfs (/sys)... "
mount -nt sysfs none /sys
check $?
```

- proc und sysfs für einige Programme dringend erforderlich

Fehlerverwaltung

/sbin/init

```
# Prüfen, ob letztes Kommando erfolgreich war
check ( ) {
  if [ "$1" = 0 ]; then
    echo "done";
  else
    echo "failed";
    echo "Starting /bin/sh (PID 1)..."
    exec /bin/busybox sh; # wir brauchen PID 1...
  fi
}
```

- einfach, zweckmäßig
- nicht „schön“, keine Fehlermeldungen
- praktisch beim Debuggen

Auslesen der Kernel-Kommandozeile

/sbin/init

```
get_var ( ) { # Kommandozeile parsen
  echo ¨`tr ' ' '\n' < /proc/cmdline | /bin/grep ¨$1 = 0¨ | \
  cut -d= -f2-`¨ ;
}

# Variablen setzen
MODULE=¨`get_var MODULE`¨ # nur dieses Modul laden
IP=¨`get_var IP`¨         # diese IP benutzen...
ROUTE=¨`get_var ROUTE`¨   # mit diesem GW...
DEV=¨`get_var DEV`¨       # und diesem Gerät
DIR=¨`get_var DIR`¨       # Pfad auf Server
SERVER=¨`get_var SERVER`¨ # IP des Servers
```

Übersicht: Kernel-Kommandozeilen Optionen

Option	Beschreibung
MODULE=[*Modulname*]	Netzwerkkartenmodul laden (anstatt aller Module)
IP=[*IP-Adresse*]	statische IP verwenden (DHCP überspringen) . . .
ROUTE=[*IP-Adresse*]	. . . mit dieser Standardroute . . .
DEV=[*ethX*]	. . . für dieses Gerät
DIR=[*/Pfad/zum/Ordner*]	Verzeichnis mit cloop-Image auf Server
SERVER=[*Adresse*]	diese Adresse für Server benutzen

Laden benötigter Module

/sbin/init

```
if [ $MODULE ]; then
  echo -n "Loading $MODULE... "
  xmodprobe -q $MODULE
  check $?
else
  # alle Module laden
  echo "Loading modules for network cards..."
  for netmodule in \
  /lib/modules/`uname -r`/kernel/drivers/net/*.ko; do
    netmodule="`basename $netmodule .ko`"
    echo "  Loading $netmodule..."
    xmodprobe $netmodule 2> /dev/null # keine Fehler, kein Test
  done
fi
```

(automatische) Konfiguration des Netzwerks

/sbin/init

```
if [ $IP ]; then
  [ ! $DEV ] && DEV="eth0"
  echo -n "Setting up $DEV (IP: $IP)... "
  ifconfig $DEV $IP up
  check $?

  if [ $ROUTE ]; then
    echo -n "Setting default gateway ($ROUTE)... "
    route add default gw $ROUTE
    check $?
  fi
else
  echo -n "Configuring network interface(s) using DHCP... "
  udhcpc -q 2> /dev/null # wir wollen keinen Daemon
  check $?
fi
```

automatische Erkennung des Servers

/sbin/init

```
if [ ! $SERVER ]; then
  echo -n "Looking for server... "
  mount -t tmpfs tmpfs /tmp && cd /tmp/
  npoll 2> /dev/null
  SERVER="`cat /tmp/server_ip`"
  check $?
  cd / && umount /tmp
fi
```

- „quick 'n dirty“: Multicast mit ncp
- allerdings: schreibbares Verzeichnis benötigt → tmpfs

Mounten des späteren Root-Dateisystems

/sbin/init

```
# Standardordner
[ ! $DIR ] && DIR="/opt/thinclient/system"

echo -n "Mounting remote filesystem from $SERVER using shfs... "
shfsmount --nomtab root@$SERVER:$DIR /mnt/remote 2> /dev/null
check $?

echo -n "Loading cloop module... "
xmodprobe cloop file=/mnt/remote/system.cloop
check $?

echo -n "Mouning ext2-filesystem containing /... "
mount -o ro -t ext2 /dev/cloop /mnt/system
check $?
```

Mounten des späteren Root-Dateisystems: Probleme

- ssh:
  - keine Passworteingabe:
    Deaktivieren des Tastatur-Echos nicht möglich → Public-Key
  - „*You don´t exist – go away!*“:
    libnss fehlt → in die Initrd integrieren
  - known_hosts nicht schreibbar (cramfs!):
    Symlink /.ssh/known_hosts → /dev/null
  - ssh greift auf Ramdisk zu:
    nicht aushängbar, da „Device busy“ → RAM-Verschwendung
- shfsmount:
  - mtab nicht schreibbar (cramfs!):
    shfsmount-Option: nomtab

Schreibbar machen des späteren Root-Dateisystems

/sbin/init

```
echo -n "Mounting tmpfs on /mnt/tmpfs... "
mount -t tmpfs none /mnt/tmpfs
check $?

echo -n "Using unionfs for rw-/... "
mount -t unionfs -o dirs=/mnt/tmpfs=rw:/mnt/system=ro unionfs \
/mnt/system
check $?
```

Starten des (eigentlichen) Thin-Client Systems

/sbin/init

```
echo "Starting /sbin/init..."
cd /mnt/system && pivot_root . initrd/ && exec /sbin/init 2
```

- pivot_root: Root-Dateisystem wechseln: /mnt/system → /
- exec: /sbin/init benötigt PID 1
- /sbin/init übernimmt weiteren Bootvorgang

Fragen

Fragen?

dann eMail an michael.hartmann@as-netz.de

LUGA-Treffen

Treffen der LUGA

- wann? – jeden ersten Mittwoch im Monat um 19 Uhr
- wo? – in den Räumen des ACF (Fröhlichstraße 6)
- was? – Newsflash, Diskussionen, Vorträge, Hilfe uvm.
- weitere Informationen: `http://www.luga.de/`

Bonus Slide: XDMCP

- XDMCP: X display manager control protocol
- Netzwerkprotokoll: X-Terminal $\longleftrightarrow$ X-Server
- erlaubt (ohne großen Aufwand) Arbeiten auf entfernten X-Server

---

kdmrc (normalerweise: /etc/kde3/kdm/kdmrc)

```
[Xdmcp]
# Whether KDM should listen to incoming XDMCP requests.
# Default is true
Enable=true # auf true setzen oder auskommentieren, um Xdmcp zu
aktivieren
```

Bonus Slide: PXE

- PXE: Preboot Execution Environment
- Betriebssystem unabhängige Umgebung für Netzwerkboot
- auf sehr vielen Geräten vorhanden
- Ablauf:
  1. PXE boot ROM wird gestartet
  2. PXE boot ROM schickt DHCP request
  3. DHCP-Server antwortet mit einer zusätzlichen ¨filename¨ Option
  4. PXE lädt die angegebene Datei über TFTP (z.B.: pxelinux)
  5. Datei wird ausgeführt
  6. Datei übernimmt nun weiteren Bootvorgang

Bonus Slide: PXE einrichten

- TFTPD
  - `INITFTPD_PATH` in Konfiguration anpassen
  - TFTPd starten
- DHCPd:
  - `filename "pxelinux.0"` zur Konfiguration hinzufügen
  - evtl. `next-server` *IP* zur Konfiguration hinzufügen
  - DHCP-Server starten
- syslinux/pxelinux
  - pxelinux.0 in INITFTPD_PATH kopieren
  - pxelinux.cfg in INITFTPD_PATH erstellen mit Angaben zu Kernel, Initrd und Appends