

Minix Neo X5 Projektseite

PROJEKTSEITE IST NOCH IM AUFBAU: REIHENFOLGE STIMMT NOCH NICHT

Projektziel

Basierend auf dem Artikel aus c't 2014 Heft 4 (S.166ff) soll ein Minix Neo X5 (im Folgenden nur noch Minix genannt) mit einem Debian-System bestückt werden. Auf dieser Basis soll der Minix als Mini-Server im lokalen Netz dienen und folgende Dienste bereitstellen:

- owncloud (zum Sync von Adressen/Kalender mit Android-Geräten und Thunderbird-Clients unter Windows/Linux)
- Dateifreigaben (mit samba)

Die Vorteile des Gerätes gegenüber einem Raspberry Pi sind:

- ähnlicher Energieanforderungen
- keine Lüfter
- integrierter Flash (16GB s.u.)
- relativ viel RAM (1GB s.u.)
- relativ hohe Rechenpower (1.4GHz Dual Core)

Nachteil: keine Erweiterbarkeit (ohne das Gehäuse zu modifizieren). Für den Einsatzbereich ist dies aber kein echter Nachteil.

Vorbereitung oder wie läuft das Ganze ab?

Da das Projekt nicht „plug 'n' play“ ist, sondern noch ein paar Schritte per Hand druchgeführt werden müssen, ist es sinnvoll sich vorab mit dem Prozedere auseinander zu setzen. Im Folgenden werden die einzelnen Schritte grob zusammengefasst, um einen Überblick zu geben.

1. Debian-VM aufsetzen, in der alle Arbeiten durchgeführt werden.¹⁾
2. root-FS(Minix-FS)²⁾ erzeugen, dies wird später das filesystem des Minix (offen)
3. Grundkonfigurationen innerhalb des Minix-FS vornehmen (hier kann das Script aus dem c't Artikel helfen) (offen)
4. kopieren des Minix-FS auf eine SD-Card kopieren (offen)
5. initramfs erzeugen, das als Boot-Loader fungiert und später den Kernel lädt (offen)
6. Kernel für Minix bauen (hört sich komplizierter an, als es ist 😊) (offen)
7. aus dem initramfs und dem neuen Kernel in ein Kernel-Image generieren. (offen)
8. Kernel-Image auf den Neo X5 flashen (offen)

Nun geht's aber endlich los.

Installation der Debian-VM

Für alle weiteren Arbeiten und um das Host-System nicht „zu verschmutzen“ wird empfohlen eine

virtuelle Maschine zu verwenden. Wem dies zu kompliziert erscheint oder wem es egal ist, Pakete zu installieren, die nur für dieses Projekt benötigt werden, der kann auch auf einem Debian-basierten Host-System (z.B. Ubuntu) alle weiteren Arbeiten durchführen.

Windows-User kommen um diesen Schritt nicht herum, da die genutzten Werkzeuge nicht ohne weiteres unter Windows funktionieren.

Man benötigt zunächst ein Debian-Image (wheezy), welches man im [Debian-Download-Bereich](#) herunterladen kann. Hier wird von einer xfce-dasierten Debian-Version (debian-7.4.0-amd64-xfce-CD-1.iso) ausgegangen. Eine einfache Netzwerk-Installation (kleinste Variante) sollte auch ausreichen. Ggf. müssen dann noch einige zusätzliche Pakete geladen werden.

Das Image wird als Installationsmedium in einer neuen Virtual-Box-Maschine eingebunden. Hier die verwendeten Eckdaten:

- Linux/64-bit Debian
- 384 MB RAM
- 8GB HDD

Also im wesentlichen die minimalen vorgeschlagenen Werte. Die Installation des OS ist weitgehend selbsterklärend. Man muss lediglich am Anfang die gewünschte Sprache auswählen. Alle Netzwerk-Fragen können beliebig beantwortet werden, da keine echte Netzwerkeinbindung benötigt wird. Die beiden User (root und Hauptnutzer) sowie deren Passwörter sollten notiert werden, da diese später benötigt werden.

Installation benötigter Pakete für die VM

Damit das Debian-System eine ARM-Architektur bearbeiten kann (root-FS anlegen, ARM-Kernel kompilieren, etc.) werden diverse Pakete benötigt. Ich spare mir diese jeweils an den benötigten Stellen zu installieren und mache dies hier gesammelt.

Damit alle Pakete gefunden werden können, benötigt die `/etc/sources.list` eine Erweiterung, die man am besten am Ende der Datei anhängt (mit root-Rechten editieren!):

```
deb http://www.emdebian.org/debian/ sid main
```

Für die Installation wechselt man in die Debian-VM und startet dort ein Terminal. Innerhalb des Terminals wechselt man mit `su` zum `root`-User, der alle Rechte hat. Das Folge Skript muss demnach als `root` ausgeführt werden. Alternativ kann man die Befehle einzeln im Terminal ausführen.

```
#!/bin/bash

# Quellen-Datenbank aktualisiersen, damit die Versionen stimmen.
apt-get update

# Installation zur Unterstützung anderer CPU-Typen
apt-get install qemu-user-static binfmt-support debootstrap

# Installation aller Tools für die Kernel-Erzeugung
# Schlüssel des zusätzlichen Repositories
```

```
apt-get install emdebian-archive-keyring
apt-get update
# Compiler für ARM-Linux
apt-get install gcc-4.7-arm-linux-gnueabi build-essential git sharutils
```

Damit sollten die Vorarbeiten abgeschlossen sein.

Anlegen des neuen root-FS

Wir benötigen einen Unterordner in dem das zukünftige root - FS, welches später das Basis-Filesystem des Minix wird angelegt wird. Um viel Schreiarbeit zu sparen, legen wir wie im c't-Artikel vorgeschlagen folgenden Ordner an und wechseln in diesen Ordner:

```
mkdir /home/neo-rootfs
cd /home/neo-rootfs
```

Man sollte diesen Ordner nicht unter /tmp anlegen, da dieser Ordner standardmäßig bei jedem Reboot gelöscht wird!

Mit dem folgenden Befehl wird das Grundsystem angelegt. Dies kann einige Zeit in Anspruch nehmen. Also Geduld!

```
qemu-debootstrap --verbose --variant=minbase --include=nano,ifupdown,netbase
--arch=armhf wheezy /home/neo-rootfs http://ftp.de.debian.org/debian
```

Kurze Erläuterung der Parameter:

| Parameter | Bemerkung |
|-----------|--|
| --verbose | Gibt alle Schritte im Terminal aus; Hilft Fehler zu finden |
| --variant | Gibt an, dass die kleinste Debian-Variante verwendet werden soll; hier Minimal Basis andere sind möglich, blähen das System aber auf |
| --include | Zusätzliche Software, die wir später auf dem Minix benutzen wollen; hier der Editor nano, Schnittstellen-Tools ifupdown Basis-Netzwerkdienste netbase |
| --arch | Zielarchitektur; hier ARM-Basiertes-System |

Die restlichen Parameter geben die Debian-Version (wheezy) sowie das Ziel (/home/neo-rootfs) und die Quelle (http://ftp.de.debian...) an.

Wenn qemu erfolgreich war, dann sollte sinngemäß die folgende Zeile erscheinen:

```
I: Base system installed successfully.
```

Im Ordner /home/neo-rootfs sollte sich nun eine neue Linux-Ordnerstruktur befinden. Hier finden die nächsten Anpassungen statt.

Minix-FS auf SD-Card kopieren

Zunächst muss der Name der SD-Card gefunden werden. Mit `lsblk` kann man den Namen finden oder man benutzt `dmesg` kurz nach dem einlegen der Karte. Man erhält eine ähnliche Ausgabe:

```
[ 6502.829511] sdb: sdb1
[ 6502.839812] sd 10:0:0:0: [sdb] No Caching mode page found
[ 6502.839838] sd 10:0:0:0: [sdb] Assuming drive cache: write through
[ 6502.839850] sd 10:0:0:0: [sdb] Attached SCSI removable disk
[ 6503.344347] EXT4-fs (sdb1): mounted filesystem with ordered data mode.
Opts: (null)
```

Bei mir wurde die SD-Card als `sdb` eingehängt.

Damit es nicht zu Fehlermeldung kommt sollte die SD-Karte zunächst ausgeworfen werden, falls diese bereits automatisch eingebunden wurde.

```
umount /media/USER/SDCARD
```

Für `USER` und `SDCARD` müssen selbstverständlich die entsprechenden Werte des eigenen Systems verwendet werden. (ggf. mit weiteren Partitionen auf der SD-Card wiederholen)

Partitionieren

```
sudo fdisk /dev/sdb
```

Mit `p`³⁾ werden alle Partionen angezeigt. So kann man überprüfen, ob das richtige Gerät ausgewählt wurde. Ansonsten zerstört man sich u.U. sein Betriebssystem!! Mit `d`⁴⁾ (VORSICHT!!) können bereits vorhandene Partitionen gelöscht werden. Nach `d` muss man per Ziffer die entsprechende Nummer der Partition angeben (s. `p`). Mit `n`⁵⁾ kann eine neue Partition angelegt werden. Man wird gefragt, ob die Partition `p` (primär) oder `e` (extended) sein soll. Wir benötigen eine primäre Partition: also `p`. Falls noch weitere Partitionen benötigt werden, sollte bei den nächsten Fragen entsprechender Platz freigelassen werden. Zu ersten Testzwecken ist es sinnvoll die gesamte SD-Card zu verwenden. Wir wählen demnach die Vorgaben als Parameter für den ersten und letzten Sektor.

Filesystem anlegen

Ist das Partitionieren erledigt, dann wird das Filesystem eingerichtet. Es wird ein EXT4 benötigt. Als Name wird `linuxroot` vorausgesetzt.

```
sudo mkfs.ext4 -L linuxroot /dev/sdb1
```

SD-Card mit VM verbinden

Dieser Schritt kann je nach System kompliziert oder sehr simpel werden. Der einfachste Weg ist es

die SD-Card als USB-Gerät in die VM einzubinden. Fertig!

Der harte Weg: Die SD-Card wird **NICHT** als USB-Gerät im Host-System eingebunden, sondern als eigene Partition. Dann wird es komplizierter. Als Erstes muss man herausfinden, unter welchem Namen die SD-Card eingebunden wurde.

```
lsblk
```

Eine mögliche Ausgabe sieht so aus:

```
NAME                MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                  8:0    0 465,8G  0 disk
├─sda1                8:1    0 310,3G  0 part
├─sda2                8:2    0   143G  0 part
├─sda3                8:3    0     1K  0 part
├─sda5                8:5    0   10,6G  0 part
└─sda6                8:6    0    1,9G  0 part [SWAP]
sdb                  8:16   0   29,8G  0 disk
└─sdb1                8:17   0   29,8G  0 part
```

HINWEIS: Über die Speichergröße lässt sich die SD-Card normalerweise leicht ermitteln.

Die SD-Card könnte demnach als `/dev/sdb` eingebunden sein. Es muss eine VDMK-Datei für VirtualBox erstellt werden und das Device schreibberechtigt werden (z.B. `USER:USER`)

```
sudo chmod 666 /dev/sdb
chown USER:USER sdcard.vmdk
sudo VBoxManage internalcommands createrawvmdk -filename sdcard.vmdk -
rawdisk /dev/sdb
```

Leider geht nach jedem Reboot die Schreibberechtigung verloren, so dass der `chmod` Befehl nach jedem Neustart wiederholt werden muss.

Danach kann die kann man in VirtualBox eine neue Festplatte zu der existierenden Einbinden. Als „Image“-Datei wird die erzeugte `sdcard.vmdk` verwendet.

Anpassungen des rootFS

Das neu erzeugte rootFS muss nun noch angepasst werden. Zunächst muss aber erst in das rootFS gewechselt werden. Dazu wird das neu erzeugte rootFS in Teilen gemounten und im Anschluss mit `chroot` zum Wurzelverzeichnis gemacht.

```
mount -t proc proc ${ROOTFS_DIR}/proc
mount -t sysfs sysfs ${ROOTFS_DIR}/sys
mount -o bind /dev ${ROOTFS_DIR}/dev
mount -t devpts devpts ${ROOTFS_DIR}/dev/pts
echo -e "${GREEN}Wechsele in ${ROOTFS_DIR}${NC}."
echo -e "Bitte nach dem Wechsel ${RED}install_tools.sh${NC} und
${RED}config_keyboard.sh${NC} aufrufen."
```

```
echo -e "Mit ${RED}exit${NC} kann ins Hauptsystem zurückgewechselt werden."  
chroot ${ROOTFSDIR}  
# mountpoints wieder entfernen  
umount ${ROOTFSDIR}/proc  
umount ${ROOTFSDIR}/sys  
umount ${ROOTFSDIR}/dev/pts  
umount ${ROOTFSDIR}/dev  
echo -e "Willkommen zurück im ${RED}Hauptsystem${NC}."
```

Im rootFS selbst müssen noch einige Pakete nachinstalliert werden und das Tastatur-Layout ausgewählt werden. Das Basis System ist zunächst mit englischem Tastaturlayout und falscher Zeitzone konfiguriert. Mit den folgenden Befehlen wird das rootFS auf deutsche Parameter eingestellt.

```
# Tastatur-Layout auf deutsch ändern:  
dpkg-reconfigure locales
```

Es werden einige perl-Fehler ausgeworfen, die ignoriert werden können. Als Parameter können die folgenden Werte dienen:

| Parameter | Wert |
|-----------|--|
| Timezone: | 8. Europe/ 6. Berlin |
| Keymap: | 155. pc / qwertz / German / Standard / latin1 - no dead keys |
| Keyboard: | 98. de_DE.UTF-8 UTF-8 |

Weiterhin sollte im rootFS ein Passwort für den Admin (root) gesetzt werden, da ansonsten kein Zugriff möglich ist:

```
passwd
```

SD-Card partitionieren und formatieren

Zunächst werden alle Partitionen auf der SD-Card gelöscht. # SD-Card mit neuer Filesystem formatieren. Die SD-Card wurde als /dev/sdb eingehängt. mkfs.ext4 -L linuxroot /dev/sdb1 # Formatierte SD-Card einhängen unter /mnt/sdcard (vorher mit mkdir /mnt/sdcard Einhängpunkt erzeugen) mount -t ext4 /dev/sdb1 /mnt/sdcard/

root-FS auf SD-Card kopieren

neues root-FS auf SD-Card kopieren -a für alle Links -v für zusätzliche Ausgabe während des Kopiervorgangs

```
cp -av /home/neo-rootfs/* /mnt/sdcard/
```

Technische Daten des Minix Neo X5

Hier die technischen Eckdaten des Minix Neo X5: (Auszug aus dem offiziellen Datenblatt):

| Type | Beschreibung |
|---------------------|--|
| Prozessor | Rockchip RK3066 Dual Core Cortex A9 1.4GHz (max. 1.6GHz) |
| GPU | Quad Core Mali 400 (OpenGL ES 2.0/1.1, Opven VG1.1, Flash 11.1) |
| RAM | 1GB DDR3 |
| Int. Speicher | 16GB NAND Flash |
| Funkschnittstellen | 802.11 b/g/n WiFi, Bluetooth, 3G über USB-Dongle (nicht enthalten) |
| OS | Android 4.1.1 Jelly Bean |
| Video Output | HDMI 1.4a, Full HD 1080p, 3D Filme unterstützt |
| Audio Output | HDMI 1.4a, optisch, S/PDIF, analog (Kopfhörer, Klinke) |
| Sonstige Anschlüsse | |
| | RJ-45 Ethernet (10/100 Mbit/s) |
| | SD/MMC Slot (SD 3.0, MMC 4.41) |
| | 3x USB 2.0 Ports |
| | Micro USB OTG Port |
| | Infrarot-Empfänger (Fernbedienung ist enthalten) |
| | Schlitz für Kensington Schloss |
| Stromversorgung | 5V, 3A Adpater (enthalten), Gerät benötigt laut Hersteller weniger als 1A |
| Videoformate | AVI/RM/RMVB/MKV/WMV/MOV/MP4/WEBM/DAT(VCD format)/VOB/MPEG/MPG/FLV/ASF/TS/TP/3GP u.a. |

Info-Sammlung

Unsortierte Infosammlung zum Thema:

Fast perfektes Skript um Ubuntu 12.04 vorzubereiten:

<http://www.myria.de/computer/864-linux-auf-dem-minix-x5> Lehnt sich stark an das c't-Skript an, ist aber ausgereifter. Einige Probleme sind allerdings immer noch enthalten.

- gcc-Symlink fehlt
- Variablen sind nicht konsequent umgesetzt

Linux permanent Boot statt nur im Recovery-Modus:

<http://minixforums.com/threads/linux-on-minix-x5.1388/page-16>

1)

kann entfallen, wenn man ohnehin unter Debian arbeitet

2)

FS: filesystem

3)

p: print

4)

d: delete

5)

n: new

From:
<http://www.kopfload.de/> - **kopfload - Lad Dein Hirn auf!**

Permanent link:
http://www.kopfload.de/doku.php?id=allgemein:minix:minix_debian&rev=1398956102

Last update: **2025/11/19 16:13**

