Minix Neo X5 build_mini_x5_sys_v2.sh

Es wird keine Haftung für Schäden, die durch hier veröffentlichte Programme verursacht werden, übernommen. Nutzung auf eigene Gefahr!

Das folgende Skript basiert auf verschiedenen Quellen. Besonders hervorzuheben ist die Seite myria.de. Diese Seite legt die Basis für diese Arbeit. Nochmals vielen Dank an dieser Stelle!! Da auch ich nicht davon ausgehen kann keine Fehler mehr im Code zu haben, bitte im mir eine kurze Nachricht zu kommen zu lassen. Dazu kann das Ergänzungen-Formular am Ende der Seite verwendet werden.

Movitation

Die Basis erzeugte einige Fehler und mich störte, dass das Flash-Tool `rkflashtool` mit einem Skript ausgeliefert wird, dass die Hardware potentiell fehlerhaft beschreiben könnte (s. change notes). Weiterhin wollte ich statt Ubuntu 12.04 LTS (Precise Pangolin) die aktuelle Debian 7 (Wheezy) Distribution nutzen.

Bedienungsanleitung

Im wesentlich funktioniert das Skript auf der myria-Seite beschrieben. Damit es nicht mit dem Original verwechselt wird habe ich es **`build_minix_x5_sys_v2.sh`** genannt.

Das Skript muss mit `root`-Rechten aufgerufen werden:

```
build_minix_x5_sys_v2.sh [COMMAND]
```

Als `COMMAND` können folgende Parameter genutzt werden.

Als Hilfe wird folgender Text angezeigt:

```
Optionen:
prepare .... System vorbereiten, nötige Pakete installieren
bootstrap .. System vorbereiten (prepare) und Dateien für rootfs
herunterladen (bootstrap)
kernel ..... Kernel herunterladen und compilieren
chrootfs ... mit chroot in das rootfs wechseln und das minix-system
vorkonfigurieren
mksystem ... führt prepare, bootstrap, chrootfs und kernel nacheinander aus

copy2sd .... System auf SD-Karte kopieren
flash2minix. recovery.img in minix Speicher flashen mit flash2minix.sh

packen ..... $WORKDIR für Backup in Datei minix.tar.bz2 packen
auspacken .. Backup minix.tar.bz2 in den Ordner $WORKDIR auspacken
adb ........ Android-SDK installieren
```

```
help ....... diese Hilfe anzeigen
```

Die Reihenfolge der wesentlichen Befehle, um ein Linux-basiertes Minix Neo X5-System aufzusetzen.

Position	Befehl	Beschreibung
1	`prepare`	Es werden die ggf. fehlenden Pakete nachinstalliert, damit die nachfolgenden Befehle ausgeführt werden können.
2	`bootstrap`	Es wird das `rootFS` gemäß der Distributionsauswahl heruntergeladen und unter `minix/minix-rootfs` abgelegt
3	`chrootfs`	Wechseln in das neue `rootFS`. Dort müssen noch die beiden Skripte `install_tools.sh` (Nachinstallation einiger Pakete, Mount-Points festlegen und Netzwerkkonfiguration) und `config_keyboard.sh` (Tastaturlayout festlegen; immer noch etwas buggy unter Debian).
4	`kernel`	Die Kernel-Quellen werden heruntergeladen und der neue Minix-Neo-Kernel wird generiert. Es wird dabei die Default-Konfiguration verwendet

Nachdem der Kernel (recovery.img) und das rootFS erstellt wurden müssen diese noch auf den Minix gebracht werden. Das rootFS wird dazu auf eine SDCARD kopiert und Kernel wird mit dem neuen Skript `flash2minix.sh` im Ordner `minix/minix-kernel/rkflashtool` in den `recovery`-Bereich des Minix kopiert. Die Reihenfolge dieser beiden Schritte ist unabhängig, da sie auf unterschiedliche Medien zugreifen.

Position	Befehl	Beschreibung
5	`copy2sd`	Alle Daten aus dem `rootFS` werden werden auf eine SDCARD, die als `/media/USERNAME/linuxroot` eingebunden wurde, kopiert. WICHTIG: Der Name `linuxroot` ist wichtig, da hierüber die SDCARD vom Image gemountet wird.
6	`flash2minix`	Das `recovery.img`-Image wird in den recovery-Bereich des Minix geflasht.

Abschließend gibt es noch eine Reihe administrative COMMANDS wie `packen` (backup), `auspacken` (restore) und `adb` (Installation der `adb`-[1)]Tools)um die Sache abzurunden. Diese Befehle sind optional und müssen nicht zwingend verwendet werden.

Befehl	Beschreibung
`packen`	Alle Daten aus dem Arbeitsverzeichnis werden in das Archive `minix.tar.bz2` gepackt. Inklusive des build-Skript selbst.
`auspacken`	Auspacken von `minix.tar.bz2` in den aktuellen Ordner. Alternativ kann der Befehl `tar -xvjf minix.tar.bz2` verwendet werden. ACHTUNG: Sollte dies im Arbeitsordner getan werden, so wird der Stand überschrieben!
`adb`	Laden und einrichten der `adb`-Tools. Damit kann der minix per `adb reboot recovery` veranlasst werden in gezielt den recovery-Mode zu starten.

Das eigentliche Script build_minix_x5_sys_v2_1.sh

Falls das Script über die Konsole laden werden soll, wenn z.B. kein Browser zur Hand ist, dann kann dieser Befehl genutzt werden:

```
wget -O skript.sh
"http://www.kopfload.de/doku.php?do=export_code&id=allgemein:minix:minix_scr
ipt&codeblock=2"
```

Und hier nun das Script:

build_minix_x5_sys_v2.sh

```
#!/bin/bash
#SDCARDDIR=/media/linuxroot
SDCARDDIR=/media/$USERNAME/linuxroot
RED='\e[1;31m'
GREEN='\e[1;32m'
CYAN='\e[1;36m'
NC='\e[0m'
BOOTSTRAP=qemu-debootstrap # Datei muss vorhanden sein
QEMU=qemu-user-static          # Paketname für qemu-debootstrap
BINFMT=binfmt-support          # Datei muss vorhanden sein
DEBOOTSTRAP=debootstrap    # Paketname für binfmt-support
GIT=git           # git Client für Kernel-Sourcecode download
SHARUTILS=sharutils
LIBUSBDEV=libusb-1.0-0-dev   # libusb
CROSSCOMPILER_DEB=gcc-arm-linux-gnueabihf # Paket des Crosscompilers
#CROSSCOMPILER=arm-linux-gnueabihf-gcc-4.7 # Crosscompiler für ARM-
Architektur; wird nur bei älteren Version <4.8 benötigt
BESSENTIAL=build-essential
LIBNCURSES=libncurses5-dev
ARCH=armhf
VARIANT=minbase

HOMEDIR=`pwd` # akutelles Home-Verzeichnis; Start-Pfad, in den der
Unterordner "minix" erstellt wird

WORKDIR=minix
BASEDIR=${HOMEDIR}/$WORKDIR
ROOTFSDIR=${BASEDIR}/$WORKDIR-rootfs
KERNELDIR=${BASEDIR}/$WORKDIR-kernel
KERNELNAME=rk3066-kernel

KERNELCONFIG=.config_minix_neo_x5_20131018 # Konfiguration für Kernel-
Compile

# Konfiguration des Minix System
MINIXROOTUSER=root     # setzen des sudo-User des minix;
MINIXHOSTNAME=minix    # Systemname des minix
MINIXSSID=wlanssid # WLAN SSID des minix
MINIXPSK=wlanpass      # WLAN PSK des minix
# Liste mit zusätzlichen Tools, die auf dem Minix Neo installiert
werden sollen.
MINIXEXTRATOOLS="nano,openssh-server,ifupdown,netbase,net-tools,isc-
dhcp-client,keyboard-configuration,vim,sudo"

DIST_MAIN=debian   # debian oder ubuntu als Zielsystem festlegen;
Version wird unten über UBUNTU_VERSION bzw. DEBIAN_VERSION festgelegt;
wenn LEER, dann wird der Wert erfragt
```

```
DIST_VERSION=wheezy    # percise für Ubuntu 12.04 ODER wheezy für
Debian 7.0; wenn leer, dann wird der Wert erfragt

MIRROR=empty
SOURCES=empty
# Parameter für Minix Neo System konfigurieren
# Ubuntu 12.04 Precise Pangolin

if [ -z $DIST_MAIN ] || [ -z $DIST_VERSION ]
then
  echo -e "Betriebssystem wählen ${GREEN}ubuntu${NC} oder
${GREEN}debian${NC}; [default: ${RED}debian${NC}]"
  read -p "Wahl :" choice
    case "$choice" in
      ubuntu|UBUNTU )
        DIST_MAIN=ubuntu
        echo -e "Ubuntu-Distribution angeben; [Default:
${RED}precise${NC}]"
        read -p "Wahl :" choice
          if [ $choice ]
          then DIST_VERSION=${choice}
          else DIST_VERSION=precise
          fi
      ;;
      * )
        DIST_MAIN=debian
        echo -e "Debian-Distribution angeben; [Default:
${RED}wheezy${NC}]"
        read -p "Wahl :" choice
          if [ $choice ]
          then DIST_VERSION=${choice}
          else DIST_VERSION=wheezy
          fi
      ;;
  esac
echo "Distribution: $DIST_MAIN und $DIST_VERSION ausgewählt."
fi

MIRROR_UBUNTU=http://ports.ubuntu.com
SOURCES_UBUNTU="deb $MIRROR_UBUNTU/ubuntu-ports/ $DIST_VERSION main
restricted universe multiverse
deb-src $MIRROR_UBUNTU/ubuntu-ports/ $DIST_VERSION main restricted
universe multiverse
deb $MIRROR_UBUNTU/ubuntu-ports/ $DIST_VERSION-updates main restricted
universe multiverse
deb-src $MIRROR_UBUNTU/ubuntu-ports/ $DIST_VERSION-updates main
restricted universe multiverse
deb $MIRROR_UBUNTU/ubuntu-ports/ $DIST_VERSION-security main restricted
universe multiverse
deb-src $MIRROR_UBUNTU/ubuntu-ports/ $DIST_VERSION-security main
```

```
restricted universe multiverse"

# Debian 7.0 Wheezy
MIRROR_DEBIAN=http://ftp.de.debian.org/debian
SOURCES_DEBIAN="deb $MIRROR_DEBIAN $DIST_VERSION main contrib non-free
deb-src $MIRROR_DEBIAN $DIST_VERSION main contrib non-free"

# Variablen für Distributionsauswahl vorbereiten.
case "$DIST_MAIN" in
  debian)
  DIST=$DIST_VERSION
  MIRROR=$MIRROR_DEBIAN
  SOURCES=$SOURCES_DEBIAN
  ;;
  ubuntu)
  DIST=$DIST_VERSION
  MIRROR=$MIRROR_UBUNTU
  SOURCES=$SOURCES_UBUNTU
  ;;
esac

[ $UID -ne 0 ] && {
    echo -e "${RED}Fehler: Das Script benötigt root-Rechte.${NC}"
    echo -e "Aufruf mit \"${GREEN}sudo $0${NC}\""
    exit 1
}

# Arbeitsarchive sichern
alles_packen() {
echo -e "Packe rootfs und kernel in ${GREEN}minix.tar.bz2${NC}"
cd $HOMEDIR
cp $HOMEDIR/${0} $WORKDIR
tar -cvjf minix.tar.bz2 $WORKDIR
}

# Arbeitsarchive wiederherstellen
alles_auspacken() {
echo -e "Packe ${GREEN}minix.tar.bz2${NC} aus nach
${GREEN}$WORKDIR${NC}"
if [ -d $WORKDIR ]
then
  echo -e "${RED}Fehler${NC}: Ordner ${RED}$WORKDIR${NC} existiert
schon!"
  read -p "Überschreiben [j|N]: " choice
   case "$choice" in
   j|J )
     rm -rf $WORKDIR;;
   * )
     echo -e "Nichts passiert. ${GREEN}OK${NC}."
     exit 1
     ;;
```

```
   esac
fi
echo -e "Erstelle ${RED}$WORKDIR${NC}."
mkdir $WORKDIR
tar -xvjf minix.tar.bz2
echo -e "Alles ausgepackt. ${GREEN}OK${NC}."
}

# System vorbereiten
prepare() {
echo -e "-------- BEGIN Vorbereitungen (${CYAN}prepare${NC}) ---------"
if [ -z $(which ${BOOTSTRAP}) ] || [ -z $(which /usr/sbin/update-
binfmts) ] || [ -z $(which ${DEBOOTSTRAP}) ]
then
  echo -e "Installiere ${RED}${QEMU} ${BINFMT} ${DEBOOTSTRAP}${NC}."
  apt-get update
  apt-get -y install $QEMU $BINFMT $DEBOOTSTRAP
else
  echo -e "${BOOTSTRAP} ${BINFMT} und ${DEBOOTSTRAP} sind bereits
installiert. ${GREEN}OK${NC}."
fi
 # extra build tools

if [ -z $(which ${GIT}) ] || [ -z $(which arm-linux-gnueabihf-gcc) ] ||
[ ! -e /usr/share/build-essential/essential-packages-list ] || [ -z
$(which uudecode) ] || [ ! -d /usr/include/libusb-1.0 ]
then
  echo -e "Installiere ${RED}${GIT}, ${CROSSCOMPILER_DEB},
${SHARUTILS}, ${LIBUSBDEV} und ${BESSENTIAL}${NC}."
  apt-get update
  apt-get -y install $GIT $CROSSCOMPILER_DEB $SHARUTILS $LIBUSBDEV
$BESSENTIAL
# gcc wird als arm-linux-gnueabihf-gcc-4.7 installiert, make erwartet
aber arm-linux-gnueabihf-gcc
# LÖSUNG: sym-link anlegen
#  ln -s $(dirname `which $CROSSCOMPILER`)/$CROSSCOMPILER /usr/bin/arm-
linux-gnueabihf-gcc
else
  echo -e "${GIT}, ${CROSSCOMPILER}, ${SHARUTILS}, ${LIBUSBDEV} und
${BESSENTIAL} sind bereits installiert. ${GREEN}OK${NC}."
fi

#ncurses für make menuconfig
if [ ! -e /usr/include/curses.h ]
then
  echo -e "Installiere ${RED}${LIBNCURSES}${NC}."
  apt-get -y install $LIBNCURSES
else
  echo -e "${LIBNCURSES} ist bereits installiert. ${GREEN}OK${NC}."
fi
```

```
if [ ! -d $BASEDIR ]
then
  echo -e "Arbeitsverzeichnis werden erstellt. ${RED}$BASEDIR${NC}."
  mkdir $BASEDIR && mkdir $KERNELDIR && mkdir $ROOTFSDIR && mkdir
${KERNELDIR}/kernel_mod
  chown -R $SUDO_USER:$SUDO_USER $BASEDIR
else
  echo -e "Arbeitsverzeichnis existiert bereits. ${GREEN}OK${NC}."
fi
echo -e "-------- END Vorbereitungen (${CYAN}prepare${NC}) ---------"
}


# rootFS anlegen
bootstrap() {
echo -e "-------- BEGIN RootFS erzeugen (${CYAN}bootstrap${NC}) -------
--"
echo -e "${RED}Boootstrap anlegen.${NC}"
cd $ROOTFSDIR
pwd
$BOOTSTRAP --verbose --no-check-gpg --variant=$VARIANT --
include=$MINIXEXTRATOOLS --arch=$ARCH $DIST $ROOTFSDIR $MIRROR
echo -e "-------- END RootFS erzeugen (${CYAN}bootstrap${NC}) ---------
"
}


# Neuen recovery.img Kernel bauen
kernel() {
echo -e "-------- BEGIN recovery.img Kernel erzeugen
(${CYAN}kernel${NC})---------"
echo -e "Kernel ${RED}herunterladen/bauen${NC}"
cd $KERNELDIR

#Kernel sourcen schon vorhanden? Sonst herunterladen
if [ ! -d rk3066-kernel ]
then
  echo -e "Hole ${RED}rk3066-kernel${NC}!"
  git clone --depth 1 https://github.com/Myria-de/rk3066-kernel-minix-
neo-x5 rk3066-kernel
else
  echo -e "rk3066-kernel ist bereits vorhanden. ${GREEN}OK${NC}."
fi

#initramfs
if [ ! -d initramfs ]
then
  echo -e "Hole ${RED}initramfs${NC}!"
  git clone --depth 1
https://github.com/Galland/rk30_linux_initramfs.git initramfs
```

```
  cd initramfs
  gzip -dc debian-3.0.8+fkubi.cpio.gz > initramfs.cpio
else
  echo -e "initramfs ist bereits vorhanden. ${GREEN}OK${NC}."
fi

# kernel schon vorhanden?
BUILDKERNEL=yes
if [ -e ${KERNELDIR}/${KERNELNAME}/arch/arm/boot/zImage ]
then
  read -p "Kernel-Image existiert bereits. Neu erstellen (j/N)?" choice
   case "$choice" in
   j|J ) BUILDKERNEL=yes;;
   * ) BUILDKERNEL=no;;
   esac
fi

#kernel erstellen
if [ ${BUILDKERNEL} == yes ]
then
  echo -e "Baue ${RED}kernel${NC}!"
  cd ${KERNELDIR}/${KERNELNAME}
# Compiler Parameter setzen
  export ARCH=arm
  export CROSS_COMPILE=arm-linux-gnueabihf-
  export INSTALL_MOD_PATH=${KERNELDIR}/kernel_mod
  export KDIR=./
  export LOCALVERSION=""
  MAKE="make -j$(getconf _NPROCESSORS_ONLN)"
  $MAKE mrproper
  cp $KERNELCONFIG .config # Default-Konfiguration für Kernel-Compile
setzen
  #cp config.pcw .config
  $MAKE
  $MAKE modules_install
else
  echo -e "Existierender kernel wird verwendet. ${GREEN}OK${NC}."
fi

# mkbootimge für das Erstellen von recovery.img
if [ ! -d ${KERNELDIR}/tools ]
then
  echo -e "Hole ${RED}mkbootimge!${NC}!"
  cd ${KERNELDIR}
  git clone --depth 1 https://github.com/olegk0/tools.git
else
  echo -e "mkbootimge ist bereits vorhanden. ${GREEN}OK${NC}."
fi

#rkflashtool zum Flashen von recovery.img
```

```
cd ${KERNELDIR}
if [ ! -d ${KERNELDIR}/rkflashtool ]
then
echo -e "Hole ${RED}rkflashtool_rk3066${NC}!"
git clone --depth 1 https://github.com/Galland/rkflashtool_rk3066.git
rkflashtool
cd ${KERNELDIR}/rkflashtool
if [ -e flash_kernel.sh ]
then
  rm flash_kernel.sh
  echo "flash_kernel.sh vorsichtshalber gelöscht!"  # mit falschen
Parameter kann es den minix zerstören
  echo -e "statt flash_kernel.sh bitte ${GREEN}flash2minix.sh${NC}
nutzen."
fi

make
if [ -e ${KERNELDIR}/rkflashtool/rkflashtool ]
then
  echo -e "rkflashtool erfolgreich erstellt. ${GREEN}OK${NC}."
  else
  echo -e "${RED}Fehler konnte rkflashtool nicht erstellen!${NC}"
  fi
else
  echo -e "rkflashtool ist bereits vorhanden. ${GREEN}OK${NC}."
fi

echo -e "Erstelle ${RED}recovery.img${NC}!"
cd ${KERNELDIR}/tools
./mkbootimg --kernel ${KERNELDIR}/${KERNELNAME}/arch/arm/boot/zImage \
--ramdisk ${KERNELDIR}/initramfs/fakeramdisk.gz --base 60400000 \
--pagesize 16384 --ramdiskaddr 62000000 \
-o ${KERNELDIR}/recovery.img
cd ${KERNELDIR}
if [ -e ${KERNELDIR}/recovery.img ]
then
  echo -e "recovery.img erfolgreich erstellt. ${GREEN}OK${NC}."
  mv ${KERNELDIR}/recovery.img ${KERNELDIR}/rkflashtool/recovery.img
else
  echo -e "${RED}Fehler: recovery.img wurde nicht erstellt!${NC}"
fi
echo -e "-------- END recovery.img Kernel erzeugen (${CYAN}kernel${NC})
---------"
}

#copy files to SD card $SDCARDDIR /media/linuxroot
copy_files() {
echo -e "-------- BEGIN Dateien auf SD-Karte (${CYAN}copy2sd${NC}) ----
-----"
echo "Dateien auf SD-Karte kopieren"
if [ -d ${SDCARDDIR} ]
```

```
then
  echo -e "Kopiere ${RED}rootfs${NC}!"
  cp -av ${ROOTFSDIR}/* ${SDCARDDIR}
  echo -e "Kopiere ${RED}Kernel-Module${NC}"
  cp -av ${KERNELDIR}/kernel_mod/* ${SDCARDDIR}
  echo -e "Kopieren beendet. ${GREEN}OK${NC}."
else
 echo -e "${RED}Fehler: Verzeichnis ${SDCARDDIR} existiert nicht. Bitte
SD-Karte einhängen.${NC}"
fi
echo -e "-------- END Dateien auf SD-Karte (${CYAN}copy2sd${NC}) ------
---"
}

# recovery.img auf minix flashen
flash_recovery() {
if [ -d ${KERNELDIR}/rkflashtool ]
then
cd ${KERNELDIR}/rkflashtool
#flash2minix.sh erstellen
echo -e "${RED}flash2minix.sh${NC} generieren. Wird zum flashen des
neuen Kernels verwendet."
cat<<EOF>flash2minix.sh
#!/bin/bash
# Machine-Model:  NEO-X5-116A
# Machine-ID:         007
# Manufactorer:   RK30SDK
#
# Partitionmap
# Partition   @Addr        length
# misc        0x2000        0x2000
# kernel  0x4000        0x6000
# boot        0xA000        0x8000
# recovery    0x12000        0x8000
# backup  0x1A000        0xC0000
# cache       0xDA000        0x40000
# userdata    0x11A000    0x800000
# kpanic  0x91A000    0x2000
# system  0x91C000    0x100000
# syntax: rkflashtool w ADDR LEN < IMG_NAME.img
# example: flash w 0x12000 0x8000 < recovery.img
RED='\e[1;31m'
GREEN='\e[1;32m'
NC='\e[0m'

if [ -f recovery.img ];
then
  echo -e "\${RED}ACHTUNG: Die Startwerte MÜSSEN korrekt sein!\${NC}"
  echo "Wenn die Adresse oder der Offset falsch ist, dann kann das
Gerät beschädigt werden!"
```

```
  echo "Lese Speicher von Minix aus!!"
  sudo ./rkflashtool r 0x0 0x1 > read.img
  echo -e "Dump-Format:
\${GREEN}OFFSET@ADRESSE(NAME)\${NC}"
  echo -e "Dump des Minix-Speichers:\${RED} \c"
  cat read.img | strings | grep --color -Po
'(?<=\(boot\)\,).*(?=\(backup)' | grep -Po '^.*(?=\,)'
  echo -e "\${NC}Ermittelte Werte für den Flash-Vorgang:"
  offset=\$(cat read.img | strings | grep -Po
'(?<=\(boot\)\,).*(?=\(recovery)' | grep -o '^0x[0-9]\{8\}')
  addr=\$(cat read.img | strings | grep -Po
'(?<=\(boot\)\,).*(?=\(recovery)' | grep -o '0x[0-9]\{8\}\$')

  rm read.img
  echo "Ermittelte Werte für recovery.img:"
  echo -e "Größe Image   : \${GREEN}\$offset\${NC}"
  echo -e "Start-Adresse : \${GREEN}\$addr\${NC}"
  echo -e "\${GREEN}Verwende folgenden Befehl zum Flashen:\${RED}"
  echo -e "\${RED}./rkflashtool w \$addr \$offset < recovery.img\${NC}"
  read -p "Parameter korrekt? [j|N]" choice
  case "\$choice" in
    j|J )
      echo -e "\\n\${RED}!!Gerät nicht abschalten schreibe
image!!\${NC}"
      echo -e "./rkflashtool w \$addr \$offset < recovery.img"
      sudo ./rkflashtool w \$addr \$offset < recovery.img
  ;;
  * ) echo -e "Nichts passiert. \${GREEN}OK\${NC}."
  ;;
  esac
else
  echo -e "Es muss zunächst ein kernel übersetzt werden und eine
gültige \${GREEN}recovery.img\${NC} Datei existieren,"
  echo "um dieses Skript zu nutzen!"
fi
EOF
chmod +x ${KERNELDIR}/rkflashtool/flash2minix.sh
./flash2minix.sh
else
  echo -e "${RED}Fehler:${NC} rkflashtool nicht installiert!"
fi
}


# in neues rootFS wechseln und letzte Änderungen vornehmen
rootfs() {
echo -e "-------- BEGIN In rootFS wechseln (${CYAN}rootfs${NC}) -------
--"
echo "Rootfs bearbeiten"

chmod 755 ${ROOTFSDIR}/install_tools.sh
```

```
chmod 755 ${ROOTFSDIR}/config_keyboard.sh

mount -t proc proc ${ROOTFSDIR}/proc
mount -t sysfs sysfs ${ROOTFSDIR}/sys
mount -o bind /dev ${ROOTFSDIR}/dev
mount -t devpts devpts ${ROOTFSDIR}/dev/pts
echo -e "${GREEN}Wechsele in ${ROOTFSDIR}${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}."
echo -e "-------- END rootFS vorbereiten (${CYAN}rootfs${NC}) ---------
"
}

# rootFS Grundkonfiguration vornehmen
prepare_rootfs() {
echo -e "-------- BEGIN Vorbereitung rootFS
(${CYAN}prepare_rootfs${NC}) ---------"
echo "Bereite rootfs vor"

# Paketquellen konfigurieren
echo -e "Lege Paketquellen fest für minix (${RED}$DIST_MAIN${NC} /
${RED}$DIST${NC})"
cat<<EOF>${ROOTFSDIR}/etc/apt/sources.list
$SOURCES
EOF

#Hostname setzen
echo -e "Hostname für minix in ${RED}/etc/hostname${NC} auf
${RED}$MINIXHOSTNAME${NC} setzen."
echo ${MINIXHOSTNAME} >  ${ROOTFSDIR}/etc/hostname
echo "127.0.1.1 ${MINIXHOSTNAME}" >>  ${ROOTFSDIR}/etc/host

#fstab
echo -e "Mountpoints für minix in ${RED}/etc/fstab${NC} setzen."
cat<<EOF>${ROOTFSDIR}/etc/fstab
/dev/root / ext4 defaults,noatime 0 0
tmpfs    /var/log    tmpfs    defaults    0 0
tmpfs    /tmp        tmpfs    defaults    0 0
tmpfs    /var/tmp    tmpfs    defaults    0 0
EOF
```

```
#Netzwerk setup
echo -e "Netzwerk für minix ${RED}/etc/network/interfaces${NC} setzen."
cat<<EOF>${ROOTFSDIR}/etc/network/interfaces
auto lo
iface lo inet loopback
# Ethernet interface eth0
auto eth0
iface eth0 inet dhcp

# WLAN interface eth1
auto eth1
iface eth1 inet dhcp
wpa-ssid $MINIXSSID
wpa-psk $MINIXPSK
EOF

echo -e "Nameserver aus Hauptsystem ${RED}/etc/resolve.conf${NC} für
minix setzen."
cp -L /etc/resolv.conf ${ROOTFSDIR}/etc/resolv.conf

# Installationsskript install_tools.sh und in rootFS ablegen. Muss nach
chroot aufgerufen werden!
echo -e "Installtionsskript ${RED}install_tools.sh${NC} anlegen. MUSS
NACH ${RED}chroot${NC} aufgerufen werden!"
cat<<EOF>${ROOTFSDIR}/install_tools.sh
PURP='\e[1;35m'
CYAN='\e[1;36m'
NC='\e[0m'
echo -e "Installiere Tools im \${PURP}rootfs\${NC}."
export LANG=C
apt-get update
apt-get -y install apt-utils dialog locales
cat <<END > /etc/apt/apt.conf.d/71neo
APT::Install-Recommends "0";
APT::Install-Suggests "0";
END
# Sprache auf deutsch wechseln
cat <<END > /etc/locale.gen
de_DE.UTF-8 UTF-8
END
export LANG=de_DE.UTF-8
locale-gen de_DE.UTF-8
dpkg-reconfigure locales
localedef -i de_DE -c -f UTF-8 de_DE.UTF-8

# Fallunterscheidung, weil firmware-Paket bei Ubuntu anders heisst als
bei Debian
if grep -iq "ubuntu" /etc/issue
then
  apt-get -y install sudo udev iproute iputils-ping wget ntpdate ntp
vim less most tzdata console-tools console-data console-common module-
```

```
init-tools linux-firmware
else
  apt-get -y install sudo udev iproute iputils-ping wget ntpdate ntp
vim less most tzdata console-tools console-data console-common module-
init-tools firmware-linux-free firmware-linux-nonfree
fi

echo -e "Bitte geben Sie das \${PURP}Passwort\${NC} und die
\${PURP}Daten\${NC} für den \${PURP}neuen root-Benutzer\${NC} ein."
adduser $MINIXROOTUSER
adduser $MINIXROOTUSER sudo
EOF

# Installationsskript config_keyboard.sh und in rootFS ablegen. Muss
nach chroot aufgerufen werden!
echo -e "Installtionsskript ${RED}config_keyboard.sh${NC} anlegen. MUSS
NACH ${RED}chroot${NC} aufgerufen werden!"
cat<<EOF>${ROOTFSDIR}/config_keyboard.sh
dpkg-reconfigure tzdata
dpkg-reconfigure console-data
dpkg-reconfigure console-common
dpkg-reconfigure keyboard-configuration
EOF

echo -e "-------- END Vorbereitung rootFS (${CYAN}prepare_rootfs${NC})
---------"
}

hilfe() {
cat <<EOF
Aufruf: sudo $0 OPTION

Optionen:
prepare .... System vorbereiten, nötige Pakete installieren
bootstrap .. System vorbereiten (prepare) und Dateien für rootfs
herunterladen (bootstrap)
kernel ..... Kernel herunterladen und compilieren
chrootfs ... mit chroot in das rootfs wechseln und das minix-system
vorkonfigurieren
mksystem ... führt prepare, bootstrap, chrootfs und kernel nacheinander
aus

copy2sd .... System auf SD-Karte kopieren
flash2minix. recovery.img in minix Speicher flashen
packen ..... $WORKDIR für Backup in Datei minix.tar.bz2 packen
auspacken .. Backup minix.tar.bz2 in den Ordner $WORKDIR auspacken
adb ........ Android-SDK installieren

help ....... diese Hilfe anzeigen
```

```
Beispiel für mksystem:
EOF
echo -e "${RED}sudo $0 mksystem${NC}"
}

# Android Tools installieren, zu Fernsteuerung des Minix Neo per Linux-Terminal
install_adb() {
apt-get --no-install-recommends install openjdk-7-jre
cd $BASEDIR
wget -c http://dl.google.com/android/android-sdk_r22.3-linux.tgz
tar zxvf android-sdk_r22.3-linux.tgz
mv android-sdk-linux $BASEDIR/android
chown -R $SUDO_USER:$SUDO_USER $BASEDIR/android

if [ ! -d ${HOMEDIR}/.android ]
then
  mkdir ${HOMEDIR}/.android
  chown -R $SUDO_USER:$SUDO_USER ${HOMEDIR}/.android
fi

if [ ! -e ${HOMEDIR}/.android/adb_usb.ini ]
then
cat<<EOF>${HOMEDIR}/.android/adb_usb.ini
# ANDROID 3RD PARTY USB VENDOR ID LIST -- DO NOT EDIT.
# USE 'android update adb' TO GENERATE.
# 1 USB VENDOR ID PER LINE.
0x2207
EOF
  chown $SUDO_USER:$SUDO_USER ${HOMEDIR}/.android/adb_usb.ini
fi

if [ ! -e /etc/udev/rules.d/51-android.rules ]
then
cat<<EOF>/etc/udev/rules.d/51-android.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="2207", MODE="0666"
EOF
  chown root:root /etc/udev/rules.d/51-android.rules
  chmod 644 /etc/udev/rules.d/51-android.rules
  sudo service udev restart
fi

cd ${BASEDIR}/android/tools
su $SUDO_USER ./android update sdk --no-ui
}


case "$1" in
  prepare)
  prepare
  ;;
```

```
  bootstrap)
  prepare
  bootstrap
  ;;
  kernel)
  prepare
  kernel
  ;;
  copy2sd)
  copy_files
  ;;
  flash2minix)
  flash_recovery
  ;;
  packen)
  alles_packen
  ;;
  auspacken)
  alles_auspacken
  ;;
  adb)
  install_adb
  ;;
  chrootfs)
  prepare_rootfs
  rootfs
  ;;
  mksystem)
  prepare
  bootstrap
  prepare_rootfs
  rootfs
  kernel
  ;;
  hilfe)
  hilfe
  ;;
  *)
  hilfe
  ;;
esac
exit 0
```

Das Flash-Script flash2minix.sh

Falls jemand nur das `flash2minix.sh` Script genötigt, hier eine direkt lauffähige Version.
ACHTUNG: Das Script muss im Ordner `HOMEDIR/minix/minix-kernel/rkflashtool` liegen und

`rkflashtool` muss existieren, damit alle Dateien gefunden werden und das Script läuft.

flash2minix.sh

```
#!/bin/bash
# flash2minix.sh is a script for flashing a recovery.img to a Minix NEO
X5 Mediaplayer
# No warranty for damages, use at your own risk
# Diese Script kann recovery.img Dateien auf einen Minix NEO X5
Mediaplayer flashen.
# Ich übernehme keine Verantwortung für Schäden am Gerät. Benutzung auf
eigene Gefahr!
# Machine-Model:  NEO-X5-116A
# Machine-ID:         007
# Manufactorer:   RK30SDK
#
# Partitionmap
# Partition   @Addr        length
# misc        0x2000        0x2000
# kernel  0x4000        0x6000
# boot        0xA000        0x8000
# recovery    0x12000        0x8000
# backup  0x1A000        0xC0000
# cache       0xDA000        0x40000
# userdata    0x11A000    0x800000
# kpanic  0x91A000    0x2000
# system  0x91C000    0x100000
# syntax: rkflashtool w ADDR LEN < IMG_NAME.img
# example: flash w 0x12000 0x8000 < recovery.img
RED='\e[1;31m'
GREEN='\e[1;32m'
NC='\e[0m'

if [ -f recovery.img ];
then
  echo -e "${RED}ACHTUNG: Die Startwerte MÜSSEN korrekt sein!${NC}"
  echo "Wenn die Adresse oder der Offset falsch ist, dann kann das
Gerät beschädigt werden!"
  echo "Lese Speicher von Minix aus!!"
  sudo ./rkflashtool r 0x0 0x1 > read.img
  echo -e "Dump-Format:              ${GREEN}OFFSET@ADRESSE(NAME)${NC}"
  echo -e "Dump des Minix-Speichers:${RED} \c"
  cat read.img | strings | grep --color -Po
'(?<=\(boot\)\,).*(?=\(backup)' | grep -Po '^.*(?=\,)'
  echo -e "${NC}Ermittelte Werte für den Flash-Vorgang:"
  offset=$(cat read.img | strings | grep -Po
'(?<=\(boot\)\,).*(?=\(recovery)' | grep -o '^0x[0-9]\{8\}')
  addr=$(cat read.img | strings | grep -Po
'(?<=\(boot\)\,).*(?=\(recovery)' | grep -o '0x[0-9]\{8\}$')

  rm read.img
```

```
  echo "Ermittelte Werte für recovery.img:"
  echo -e "Größe Image   : ${GREEN}$offset${NC}"
  echo -e "Start-Adresse : ${GREEN}$addr${NC}"
  echo -e "${GREEN}Verwende folgenden Befehl zum Flashen:${RED}"
  echo -e "${RED}./rkflashtool w $addr $offset < recovery.img${NC}"
  read -p "Parameter korrekt? [j|N]" choice
  case "$choice" in
    j|J )
      echo -e "\n${RED}!!Gerät nicht abschalten schreibe image!!${NC}"
      echo -e "./rkflashtool w $addr $offset < recovery.img"
      sudo ./rkflashtool w $addr $offset < recovery.img
  ;;
  * ) echo -e "Nichts passiert. ${GREEN}OK${NC}."
  ;;
  esac
else
  echo -e "Es muss zunächst ein kernel übersetzt werden und eine
gültige ${GREEN}recovery.img${NC} Datei existieren,"
  echo "um dieses Skript zu nutzen!"
fi
```

Change-Notes build_minix_x5_sys_v2_1.sh

- Android SDK wird jetzt ohne GUI automatisch aktualisiert. `su $SUDO_USER ./android update sdk –no-ui`

Change-Notes build_minix_x5_sys_v2.sh

- `auspacken` ist neu. Es packt das zuvor mit `packen` eingepackte Arbeitsverzeichnis wieder aus.
- Debian 7 / wheezy als Distribution auswählbar. Default: `DIST_MAIN=debian` und `DIST_VERSION=wheezy`, sind die beiden Variablen nicht gesetzt, dann wird die Distribution abgefragt.
- diverse farblich hinterlegte Statusmeldungen.
- Einführung zusätzlicher Konstanten, zu leichteren Anpassbarkeit
- Das build-Skript wird mit eingepackt. (vgl. `COMMAND=packen`)
- `flash_kernel.sh` wird gelöscht, da es das Gerät zerstören könnte. Stattdessen wird das Skript `flash2minix.sh` erzeugt.

1)
Android Debug Bridge

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

Permanent link:
http://www.kopfload.de/doku.php?id=allgemein:minix:minix_script&rev=1408718851

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