Raspberry Pi

Karsten Müller
email@kmkcl.de

1. Mai 2013

Inhaltsverzeichnis

1 Vorwort

Ich besitze seit einiger Zeit ein Raspberry Pi und habe bisher schon das ein- oder andere ausprobiert. Dazu habe ich vorallem Anleitungen und Informationen aus dem Internet benutzt. Bevor ich nun völlig den Überblick über meine Quellen, aber vor allem über die speziellen Anpassungen auf meine Bedürfnisse verliere möchte die Gelegenheit nutzen und diese kleine Dokumentation erstellen. Da es ein privates Projekt ist, sind Fehler nicht ausgeschlossen. Für mögliche Schäden kann ich daher keine Haftung übernehmen.

2 SSH-Server einrichten

Beim Einrichten des SSH-Servers ist vor allem auf Sicherheit zu achten. Daher habe ich mich dazu entschieden, den root-Account und den Standart pi-Account zu sperren und eine Authorisierung nur über das Public-key Verfahren (2048 bit Schlüssel) zu erlauben, da dies gegen Angriffe sicherer ist, als Passwörter. Dazu habe ich auf Laptop, Smartphone, USB-Stick jeweils einen passwortgeschüzten Schlüssel.

In der `/etc/ssh/sshd_config` ist auf folgende Einstellungen zu achten:

```
PermitRootLogin no
RSAAuthentication yes
PubkeyAuthentication yes
PasswordAuthentication no
UsePAM no
DenyUsers pi
```

2.1 dynamische DNS

Um den Raspberry Pi von überall aus erreichbar zu machen, werden zwei Dinge benötigt:

1. ein offener Port auf dem Router mit Weiterleitung
2. eine dynamische DNS-Adresse, da sich die Router-IP in regelmäßigen Abständen ändert.

Nachdem ein Kommilitone die Erfahrung gemacht hat, dass der SSH-Port 22 von außerhalb häufig angesprochen wird, entschloss ich mich einen hohen Port (¿3000) zu nutzen. Da die Einstellung im Router geschieht und bei jedem Router anders funktioniert, taucht sie hier nicht auf.
Als Dienst für die dynamische Adresse benutze ich das kostenlose Angebot von no-ip.org. Das angebotene Linux-Programm ließ sich ohne Probleme auf dem Raspian kompilieren.

3 GPIO ansteuern

Die programmierbaren Anschlüsse, auch GPIOs genannt, lassen sich ganz einfach von der Konsole aus bedienen. Der Befehl

```
echo "7" > /sys/class/gpio/export
```

aktiviert den GPIO7. Dabei wird ein Verzeichnis `/sys/class/gpio/gpio7` erstellt, in denen es unter anderem die Dateien `direction` und `value` gibt. Diese Dateien sind selbsterklärend: `direction` gibt an, ob es sich um einen Ein- oder Ausgang handelt, `value` den Wert. Standartmäßig ist direction=`in` aktiv. Für einen Ausgang muss folgender Befehl aufgerufen werden:

```
echo "out" > /sys/class/gpio/gpio7/direction
```

An dem Pin liegt allerdings erst ein Potential an, nachdem `value` mindestens einmal geschrieben wurde.

Weiterhin ist auf die Rechtevergabe der Dateien zu achten. Dazu gibt es bei mir eine Gruppe *gpio*, deren all diese Dateien angehören und alle Benutzer, die darauf Zugriff haben sollen. Eine Initialisierung, um die GPIOs 7,8,9 und 25 als Ausgänge zu schalten, könnte so aussehen:

```
for i in {7,8,9,25}; do
  echo $i    > /sys/class/gpio/export
  echo "out" > /sys/class/gpio/gpio${i}/direction
  echo "0"   > /sys/class/gpio/gpio${i}/value
  chgrp gpio   /sys/class/gpio/gpio${i}/value
done
```

3.1 Geburtstagskuchen

Mit diesen ersten Schritten ließ sich schon ein kleines Projekt realiseren, eine kleine Geburtstagsüberraschung für eine Freundin. Auf die Idee kam ich, da Raspberry Pi phonetisch identisch mit Raspberry Pie, also Himbeerkuchen ist. Eine Mit einer LED-Kerze auf dem Gehäuse wurde der Raspberry nun zu einem Geburtstagskuchen mit Kerze.
Die Freundin bekam per E-Mail einen ssh-Schlüssel und konnte sich über die dynamische DNS-Adresse anmelden. Auf dem Raspberry-Pi-Account wartete ein kleiner Geburtstagstext und die Möglichkeit die Kerze sowohl „anzuzünden“, als auch „auszupsuten“. Das Skript dazu sieht so aus:

```
#!/bin/bash
case ${1} in
        "anzuenden") echo "1" > /sys/class/gpio/gpio7/value ;;
        "auspusten") echo "0" > /sys/class/gpio/gpio7/value ;;
        *) grep -q "0" /sys/class/gpio/gpio7/value && echo "Die Kerze ist aus!
            Tippe \"kerze anzuenden\" um die Kerze anzuzuenden."||echo "Die Kerze
            brennt! Tippe \"kerze auspusten\" um die Kerze auszupusten." ;;
esac
```

3.2 Ausgänge elektrisch

Das direkte anschließen an die Pins ist jedoch nicht der optimale Weg, da sehr genau auf die zulässigen Ströme und Potentiale geachtet werden muss. Deshalb steuern die Ausgänge nun einen `ULN2803` an. Dieser IC beinhaltet 8 Darlington-Stufen. Die resultierenden Ausgänge sind Open-Collector und schalten bei einem positiven Eingang das GND-Potential durch.

4 UART

Der Raspberry Pi besitzt neben primitiven Ein-/ Ausgängen auch einige fertige Schnittstellen, under anderem den UART. Standartmäßig ist die UART-Schnittstelle aktiviert, fungiert als Konsole und gibt Debugging-Informationen raus. Soll der UART zur normalen Kommunikation mit einem Programm genutzt werden, müssen diese Funktionen abgeschaltet werden. Dazu muss in der Datei `/boot/cmdline.txt` folgendes entfernt werden:

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

Der Rest der Zeile muss unbedingt stehen bleiben!
Weiterhin muss in der Datei `/etc/inittab` folgendes auskommentiert oder gelöscht werden:

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Der UART findet sich unter `/dev/ttyAMA0` mit der Gruppenzugehörigkeit `dialout`. Am sinvollsten ist es, die Benutzer, die darauf Zugriff haben sollen ebenfalls in diese Gruppe einzutragen.

Es hat sich gezeigt, dass mit Python eine einfache Kommunikation mit der seriellen Schnittstelle möglich ist. Dazu muss lediglich das Paket `python-serial` nachgeladen werden. Ein kleines Skript soll zeigen, wie einfach die Verwendung ist:

```
#!/usr/bin/python
from time import *
lt=localtime()
import serial
ser=serial.Serial('/dev/ttyAMA0',9600,timeout=2)
ser.write("+B+")
s=ser.readline()
ser.close()
print strftime("%d.%m.%Y %H:%M",lt), s[:-1]
```

Dieses Skript sendet die Zeichenkette „+B+“zu einem Mikrocontroller und gibt das Ergebnis mit dem aktuellen Datum aus. Da sowohl `ser.readline`, als auch `print` einen Zeilenumbruch erzeugen, wird `s` nur bis zum vorletzten Zeichen ausgegeben.

4.1 Mikrocontroller

Der Mikrocontroller dient als Erweiterung der Ein- und Ausgabemöglichkeiten. Kommuniziert wird über die UART Schnittstelle mit einem trivialen Protokoll. Der Computer sendet einen Befehl in einer Zeile und erhält eine einzeilige Antwort. An dem Microcontroller ist ein Luftfeuchtigkeitsfühler mit Frequenzausgang und eine IR-LED angeschlossen. Mit einer Programmbibliothek[1] kann der Mikrocontroller verschiedene Fernbedienungen immitieren. So lässt sich beispielsweise meine RGB-LED-Beleuchtung über den Raspberry steuern.

5 One-wire

Die one-wire-Schnittstelle ist eine weitere Schnittstelle, die der Raspberry unterstützt. Sie muss allerdings erst in den Kernel eingebunden werden. Dazu muss die Datei `/etc/modules` um folgene Einträge ergänzt werden:

```
wire
w1_gpio
```

Danach muss ein Neustart erfolgen. Alternativ können die Module auch temporär mit dem Befehl `modprobe` nacheinander geladen werden.

5.1 Temperaturfühler

Der Temperaturfühler DS18x20 wird von Haus aus mit dem Kernelmodul `w1_therm` unterstützt. Auch dieses muss entweder per `modprobe` oder durch einen Eintrag in `/etc/modules` geladen werden. Abhängig von den Seriennummern entstehen dann folgende Dateien, aus denen die Temperatur ausgelesen werden kann:

```
/sys/devices/w1_bus_master1/SERIENNUMMER/w1_slave
```

Alle erkannten Seriennummern finden sich in der Datei `/sys/devices/w1_bus_master1/w1_master_slaves`

6 I^2C

Um die I^2C Schnittselle zu benutzen müssen folgende Kernelmodule geladen werden:

```
i2c-bcm2708
i2c-dev
```

Zugegriffen wird über die Datei `/dev/i2c-1`, da dies bei Rev.B. der I^2C Bus ist, der auf die 26pol Stiftleiste geführt ist. Dabei ist wieder auf die Zugriffsrechte zu achten! (Am Besten der Gruppe gpio hinzufügen) Hilfreich ist zudem das Programmpaket `i2c-tools`, welches einfach über die Paketverwaltung installiert werden kann. Zum einen lassen sich damit angeschlossene Devices erkennen, zum anderen bietet es eine einfache Kommunikation per Konsole. Programme, die `i2c-tools` zur Verfügung stellt:

```
i2cdetect
i2cset
i2cget
```

Ersteres scannt die I^2C Schnittstelle nach Devices, die anderen beiden greifen schreibend, bzw. lesend auf ein Device zu. Näheres ist in den manpages zu finden. Auch hier lässt sich jedoch auch recht einfach mit Python zugreifen. Das notwendige Paket dafür ist `python-smbus`. Eine Klasse die den Zugriff vereinfacht stammt von Mathew Skolaut[2]:

```
class i2c_device:
        def __init__(self, addr, port):
                self.addr = addr
                self.bus = smbus.SMBus(port)

        def write(self, byte):
                self.bus.write_byte(self.addr, byte)

        def read(self):
```

```
        return self.bus.read_byte(self.addr)

    def read_nbytes_data(self, data, n): # For sequential reads > 1 byte
        return self.bus.read_i2c_block_data(self.addr, data, n)
```

6.1 Display

Mathew Skolaut hat neben der oben beschriebenen Klasse auch eine Klasse für die Ansteuerung eines 20x4 Zeichen Displays mit Standart-Controller geschrieben. Dafür wird ein `PCF8574` als Adapter von I^2C zu paralell eingesetzt. Dies erlaubt den einfachen und Pin sparenden Anschluss eines 5V Displays an den Raspberry Pi. Ich habe die Klasse leicht abgewandelt, da sich das Display nicht bei jedem Aufruf neu initialisieren soll. Dies erlaubt unabhängige Zugriffe auf das Display, wobei das Display nicht jedesmal gelöscht wird.

7 Wetterbericht aus dem Internet

Eine Anwendung für mein Display ist die anzeige des aktuellen Wetterberichts. Das ist bei mir ganz praktisch, da das Display gut sichtbar über meinen Schreibtisch hängt. Leider gibt es nur wenige Dienste, die ihren Wetterbericht kostenlos in einem verwertbaren Format anbieten. Das xml-Format wird von yahoo völlig frei und von wetter.com nach kostenloser Registrierung angeboten. Vorausgesetzt für private Projekte, nach anderen Nutzungsbedingungen habe ich nicht geschaut. Ein Python-Skript zur Auswertung dieser Wetter xml-Daten hat Daniel Kampert[3] geschrieben.

8 Webcam

Linux sollte die Webcam automatisch erkennen und die Datei `/dev/video0` anlegen. Dann lässt sich die Webcam mehr oder weniger problemlos verwenden. Bei mir hat es leider eine ganze Weile gedauert, bis ich herausgefunden habe, dass meine billig-Webcam nur mit einer Auflösung von 320x240 am Raspberry funktioniert. Grund scheint die Übertragungsrate zu sein, die der USB-Port des Raspberrys sonst nicht bewältigen kann.

Mein Ziel mit der Webcam war, per ssh ein Foto von meinem Zimmer zu erstellen und betrachten zu können. Auch für diese Aufgabe erwies sich Python als optimal. Allerdings muss das Paket `python-opencv` nachinstalliert werden. Das Python-Skript sieht dann so aus:

```
import cv

capture =cv.CaptureFromCAM(0)
cv.SetCaptureProperty(capture,cv.CV_CAP_PROP_FRAME_WIDTH,320)
cv.SetCaptureProperty(capture,cv.CV_CAP_PROP_FRAME_HEIGHT,240)

frame=cv.QueryFrame(capture)
cv.SaveImage("capture.jpg",frame)
```

Qualität und erweiterte Einstellungen sind mir erstmal egal. So kann ich jedenfalls schon sehen, ob meine RGB-Beleuchtung an oder aus ist ;)

Ich habe die Informationen übrigens auf der Internetseite acadopus[4] gefunden. Neben der verwendeten minimalen Zeilen kann das Python-Skript wie folgt noch ausgebaut werden:

```
config = {
    cv.CV_CAP_PROP_BRIGHTNESS: 50,
    cv.CV_CAP_PROP_CONTRAST: 50,
    cv.CV_CAP_PROP_SATURATION: 50,
}

for param, value in config.iteritems():
    cv.SetCaptureProperty(capture, param, value)
```

um zum Beispiel Helligkeit und Kontrast zu steuern, oder wie folgt um ein Video abzuspeichern:

```
import cv

capture = cv.CaptureFromCAM(0)
fourcc = cv.CV_FOURCC('M','J','P','G')
fps = 16
w, h = 640, 480
stream = cv.CreateVideoWriter("test.avi", fourcc, fps, (w, h))
while True:
    frame = cv.QueryFrame(capture)
    cv.WriteFrame(stream, frame)
```

9 Software

9.1 truecrypt

Leider gibt es kein fertiges Paket für die ARM-Architektur. Es bleibt also nur der Weg des selber kompilierens. Die Anleitung stammt von Mathias[5]. Ich habe sie auf das Notwendigste gekürzt:

1. aktuelles Truecrypt runterladen (tar.gz)
2. wget http://prdownloads.sourceforge.net/wxwindows/wxWidgets-2.8.12.tar.gz (anscheinend gab es Probleme mit neueren Versionen)
3. sudo aptitude install libfuse-dev
4. wget ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/*.h (Dateien in Ordner pkcs-header-dir speichern)
5. mit tar -xf sowohl truecrypt, als auch wxWidgets entpacken
6. export PKCS11_INC=/home/pi/pkcs-header-dir/ (absoluter Pfad zu pkcs-header-dir)
7. cd truecrypt (natürlich der entsprechende truecrypt Ordner)
8. make NOGUI=1 WX_ROOT=/home/pi/wxWidgets-2.8.12 wxbuild
9. make NOGUI=1 WXSTATIC=1
10. media/truecrypt -m=nokernelcrypto (Der Parameter ist wichtig! Sonst Truecrypt wie gewohnt)

10 Anhang

10.1 meine Pin-Belegung

Pin	Funktion	Verwendung
2	5V	5V-Versorgung
3	SDA	I^2C
5	SCL	I^2C
7	GPIO 4	one-wire
8	TXD	UART
10	RXD	UART
17	3V	3V-Versorgung
20	GND	GND
21	GPIO 9	Ausgang 3
22	GPIO 25	Reset Mikrocontroller
24	GPIO 8	Ausgang 2
26	GPIO 7	Ausgang 1

Literatur

[1] http://www.mikrocontroller.net/articles/IRMP Zugriff 01.05.2013

[2] http://tech2077.blogspot.de/2012/06/running-hd44780-lcd-over-i2c-on.html Zugriff 01.05.2013

[3] http://kampis-elektroecke.de/?page_id=3894

[4] http://acadopus.de/python/bilder-aus-webcam-auslesen_2604.html Zugriff 01.05.2013

[5] http://disfunctions.de/allgemein/truecrypt-auf-dem-raspberry-pi-installieren Zugriff 01.05.2013