Eine eigene Linux-Distribution für den Raspberry Pi mit Yocto

Ubucon 2014

18. Oktober 2014

Stefan Seyfried
Linux Consultant & Developer
B1 Sytems GmbH
seife@b1-systems.de

Vorstellung B1 Systems

- gegründet 2004
- primär Linux/Open Source-Themen
- national & international tätig
- über 60 Mitarbeiter
- unabhängig von Soft- und Hardware-Herstellern
- Leistungsangebot:
  - Beratung & Consulting
  - Support
  - Entwicklung
  - Training
  - Betrieb
  - Lösungen
- dezentrale Strukturen

Schwerpunkte

- Virtualisierung (XEN, KVM & RHEV)
- Systemmanagement (Spacewalk, Red Hat Satellite, SUSE Manager)
- Konfigurationsmanagement (Puppet & Chef)
- Monitoring (Nagios & Icinga)
- IaaS Cloud (OpenStack & SUSE Cloud & RDO)
- Hochverfügbarkeit (Pacemaker)
- Shared Storage (GPFS, OCFS2, DRBD & CEPH)
- Dateiaustausch (ownCloud)
- Paketierung (Open Build Service)
- Administratoren oder Entwickler zur Unterstützung des Teams vor Ort

Partner


SUSE
PartnerNet
Platinum
CERTIFIED
SUSE PARTNER


redhat.
PREMIER
BUSINESS
PARTNER


ubuntu
Advantage reseller


puppet
labs


pfSense


Nagios
Authorized
Reseller


Hortonworks

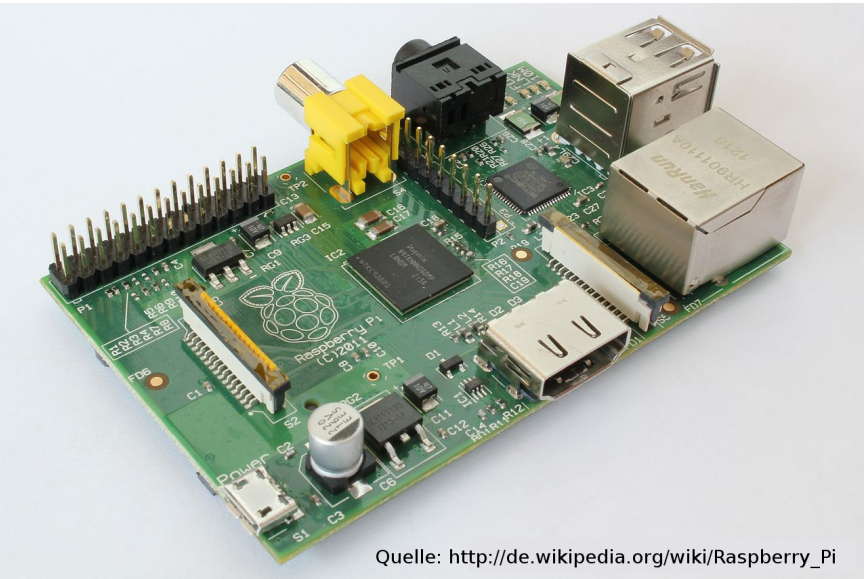

ARISTA


rackspace
the open cloud company


ownCloud
YOUR CLOUD, YOUR DATA, YOUR WAY!
Gold Partner

Der Raspberry Pi im Überblick

Raspberry Pi Hardware


Quelle: http://de.wikipedia.org/wiki/Raspberry_Pi

- Broadcom BCM2835
- ARMv6 mit FPU
- VideoCore Coprozessor

Vorteile des Raspberry Pi

- günstiger Preis
- hohe Stückzahl
- vollständig Open Source
- lange Verfügbarkeit in gleicher Konfiguration (Embedded-Boards sind oft nur wenige Monate lieferbar, dann kommen neue Boards in anderer Konfiguration)
- Multimediafähigkeit gut unterstützt

→ starke Verbreitung und gute Softwareunterstützung

Nachteile des Raspberry Pi

- kein SATA Anschluss
- fast alle Peripherie (auch das integrierte Fast-Ethernet) wird über USB angebunden
- „gemütliche“ CPU

Fertige Linux-Distributionen für den Raspi

Liste von `http://www.raspberrypi.org/downloads/`

- Raspbian (Debian Wheezy)
- Pidora (Fedora Remix)
- Openelec (XBMC Media Center)
- Raspbmc (XBMC Media Center)
- Arch Linux
- ...

Vorteile dieser Linux-Distributionen

- einfach zu installieren
- großes Softwareangebot
- guter Support

Warum selbst bauen?

Nachteile einer selbstgebauten Distribution

- Komplexität
- Arbeitsaufwand
- Dauer des Buildvorgangs
- Support: Hilf dir selbst...

„Warum selbst machen, die existierenden Distributionen funktionieren gut.“

Vorteile einer selbstgebauten Distribution

- Es macht Spaß :-)
- individuelle Anpassungen
  - dadurch sind extrem kleine, abgespeckte Images möglich, die nur die benötigten Komponenten enthalten
- Support: Hilf dir selbst...
- Crosscompilation

Verschiedene Methoden zum Selbstbau

- Nativ auf dem Raspi neu bauen, z.B. Linux from scratch
- Toolchain mit Crosstool, dann die Target-Software bauen (automatisch per Skript oder Makefile)
  - „Been there, done that“ – für andere Embedded-Boards/Boxen
  - kein Spaß, oftmals subtile Fehler im Ergebnis
- fertige Buildumgebungen für (Embedded-)Distributionen
  - OpenWRT
  - Buildroot
  - Openembedded/Yocto
  - ...

Nativ auf dem Zielsystem (Raspi) bauen

- Vorteile:
  - keine Crosskompilierprobleme
  - funktioniert
- Nachteil:
  - zu langsam

Crosstool/Buildscript

- Vorteile:
  - relativ leicht Überschaubar (anfangs...)
  - niedrige Einstiegshürde
  - „Quick Hacks“ einfach machbar
- Nachteile:
  - fehlerträchtig
  - aufwändig zu warten
  - Abhängigkeiten

Fertige Buildumgebungen

- Nachteile:
  - Komplexität
  - Aufwand zur Einarbeitung
  - eigene Anpassungen müssen in das Framework passen
- Vorteile:
  - Regeln für Standardsoftware fertig verfügbar
  - werden meist professionell maintained
  - Paketverwaltung meist inklusive
  - nur die eigenen Anpassungen müssen selbst gepflegt werden

Warum Yocto?

- „Openembedded in besser benutzbar“
- umfangreiche Dokumentation auf `http://yoctoproject.org`
- gut maintained
- SDK

Yocto bauen – jetzt geht's los!

Vorbereitung – Was wird benötigt?

- Plattenplatz (50GB+)
- schneller Rechner (Core2 duo+)
- Breitbandanschluss (5GB+ Download)
- SSD schadet nicht...

Jetzt aber!

```
git clone git://git.yoctoproject.org/poky yocto-poky
cd yocto-poky
git checkout -b daisy origin/daisy # der aktuelle branch
git clone git://git.yoctoproject.org/meta-raspberrypi
cd meta-raspberrypi
git checkout -b daisy origin/daisy
cd ..
. oe-init-build-env
# landet im neu erstellten Verzeichnis "build"
```

Konfiguration – Bevor es losgeht

Zwei Dateien unterhalb des `build`-Verzeichnisses:

- `conf/bblayers.conf`
  - `meta-raspberrypi` mit eintragen
- `conf/local.conf`
  - `MACHINE ?= "raspberrypi"`
  - `PACKAGE_CLASSES = "package_ipk"`
  - `EXTRA_IMAGE_FEATURES += "package-management"`
  - `PRSERV_HOST = "localhost:0"`
  - eventuell: `INHERIT += "rm_work"`

Bau des Images

- `bitbake rpi-basic-image`
- dauert lange (5h auf Core2 Duo L9400):
  1. Herunterladen
  2. Bauen der Abhängigkeiten für den Buildhost
  3. Bauen der Crosscompiler/Toolchain
  4. Bauen der Software für den Zielhost (Raspi)
- braucht Platz:
  - mit `"rm_work"`: 8.7G
  - ohne `"rm_work"`: 19G

Ergebnis

Das Image landet in `tmp/deploy/images/raspberrypi`:

- `rpi-basic-image-raspberrypi.rpi-sdimg`
- `rpi-basic-image-raspberrypi.ext3`
- `rpi-basic-image-raspberrypi.tar.bz2`
- Symlinks auf per Zeitstempel „versionierte“ Dateien.
- Das `.rpi-sdimg` kommt auf die SD-Karte (mit `dd`)
- Größe des rootfs (ungefähr):

```
bzcat rpi-basic-image-raspberrypi.tar.bz2 | wc -c
65402880
```

Das SDK zum Image bauen 1/2

- SDK enthält Archiv mit Crosscompiler, Headerdateien etc. für alle im Image vorhandenen Softwarepakete
- kann auf anderem Rechner verwendet werden, um unabhängig von Yocto lauffähige Binaries für den Raspberrypi zu bauen
- mit dem SDK gebaute Binaries passen genau zu den per Yocto gebauten Libraries

Das SDK zum Image bauen 2/2

- Bauen: `bitbake rpi-basic-image -c populate_sdk`
- dauert wieder ... ;-)
- Ergebnis in `tmp/deploy/sdk`:
  `poky-eglibc-x86_64-my-image-armv6-vfp-toolchain-1.6.1.sh`
- selbstauspackendes Shellskript, ca. 150 MB groß

Das SDK benutzen

1. Auf dem Zielrechner das Skript aufrufen, fragt nach Installationsverzeichnis
2. In das Installationsverzeichnis wechseln
3. `source environment-setup-armv6-vfp-poky-linux-gnueabi`
   - Setzt Variablen, z.B. CC, AS, LD, CPP
   - SITE_CONFIG für `autoconf` basierte Projekte
4. `$CC hello.c -o hello`
5. `autoconf` basierende Projekte: einfach wie immer bauen

Anpassen des Images

Nun haben wir ein Image, aber es ist Standard, ohne Anpassungen.
Also packen wir jetzt eigene Pakete mit hinein:

- In `conf/local.conf` mittels
  `IMAGE_INSTALL_append = " pulseaudio"`
  - geringer Aufwand
  - Diese Pakete (plus Abhängigkeiten) sind dann in *jedem* Image
- Eigene Image-Definition in eigenem meta-Layer
  - mehr Aufand, komplexer
  - viel flexibler
  - eigene Recipes für benötigte Pakete können da mit hinein
  - Beispiel (eventuell nicht das Beste... ;-)
    `https://github.com/seife/meta-neutrino-mp`

Eigener meta-Layer zur Image-Anpassung 1/2

Im Top-level Yocto-Verzeichnis (über `build`):

```
yocto-layer create mylayer 10 # name prio
```

- Konfiguration in `meta-mylayer/conf/layer.conf` (nicht ändern)

```
mkdir -p meta-mylayer/recipes-mine/images
cp meta-raspberrypi/recipes-core/images/rpi-hwup-image.bb \
meta-mylayer/recipes-mine/images/my-image.bb
vi meta-mylayer/recipes-mine/images/my-image.bb
```

Eigener meta-Layer zur Image-Anpassung 2/2

```
# Base this image on core-image-minimal
include recipes-core/images/core-image-minimal.bb
# from rpi-basic-image
IMAGE_FEATURES += "ssh-server-dropbear splash"
# Include modules in rootfs
IMAGE_INSTALL += " \
        kernel-modules \
        pulseaudio-misc \
        pulseaudio-lib-bluez4-util \
        pulseaudio-module-bluez4-discover \
        pulseaudio-module-bluez4-device \
        pulseaudio-module-bluetooth-policy \
        pulseaudio-module-bluetooth-discover \
        "
```

Layer in `conf/bblayers.conf` eintragen!

Vielen Dank für Ihre Aufmerksamkeit!

Bei weiteren Fragen wenden Sie sich bitte an info@b1-systems.de
oder +49 (0)8457 - 931096