Ein ganzes GNU/Linux

Hans Buchmann FHNW/IME

1. Juni 2015

Um was geht es ?

- ein GNU/Linux von Grund auf bauen
  - nicht meht so schwer wie auch schon
- nicht völlig automatisiert
- Alternative zu **yocto** & Co.

Ziel

GNU/Linux auf dem **RaspberryPi**

- command based
- Ethernet
- `ssh`
- `sshfs`
- moderne Toolchain inkl. *c++11* **C++**

Komponenten

RaspberryPi und *Host*

RaspberryPi

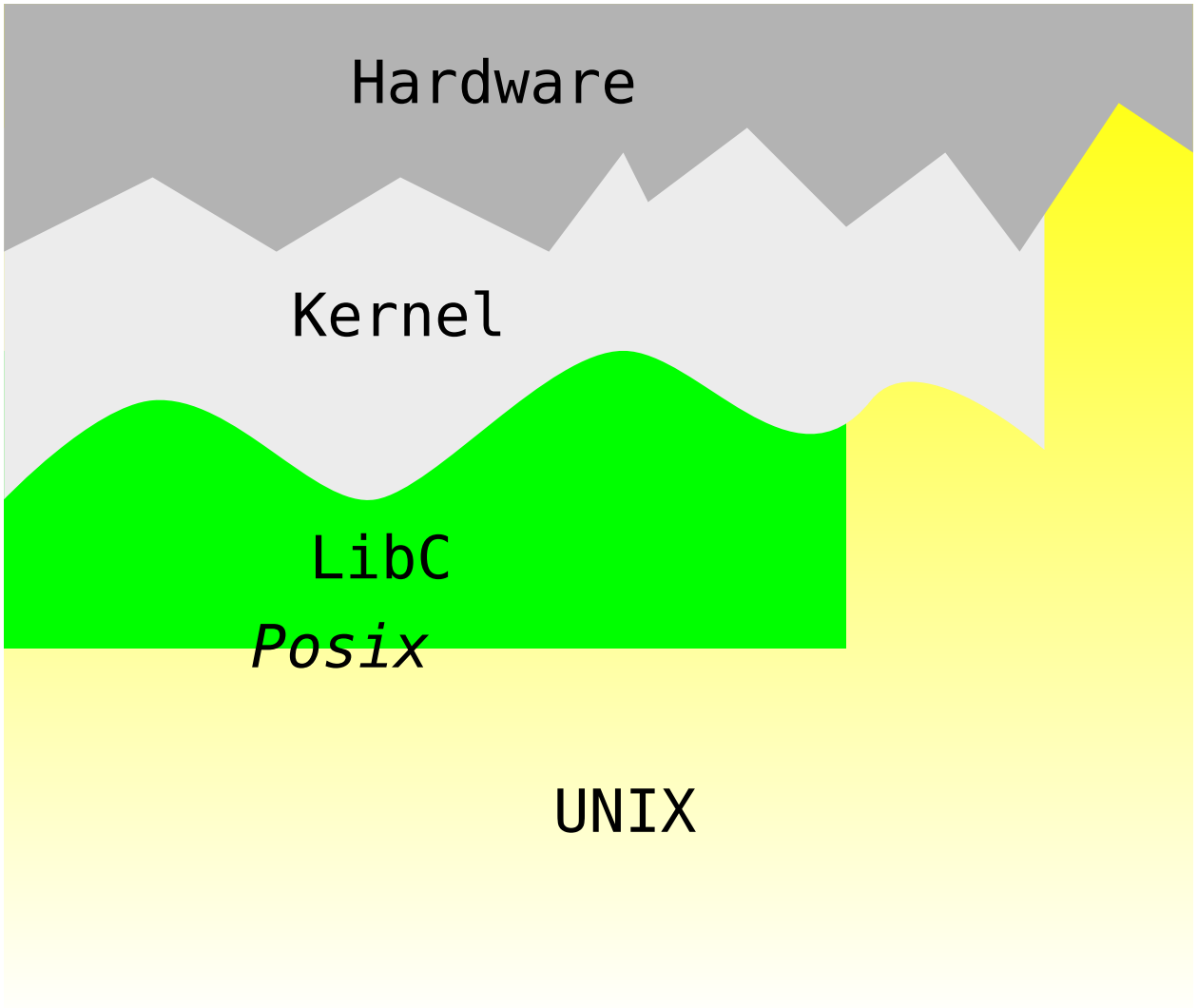
Kernel ein File

root ein Filesystem

Host

Toolchain binutils, gcc

Übersicht



Die Komponenten
für **RaspberryPi**

Hardware **RaspberryPi**

Kernel zugeschnitten auf **RaspberryPi**

- `https://github.com/raspberrypi/linux.git`

root das Filesystem

LibC eglibc

- `http://www.eglibc.org/home`

UNIX busybox

- `http://www.busybox.net/`

... Weitere UNIX basierte Komponenten

- das `configure`, `make`, `make install` Triple

Toolchain

binutils linker & Co.

gcc compiler

- `libgcc` die Bibliothek für den Compiler

Remark(s):

- die Toolchain muss zweimal gebaut werden
  - für den **kernel** und `libc`
  - für UNIX/**POSIX**
- das target
  - `cpu-vendor-os`

Die Verzeichnisstruktur

```
├── scripts
│   └── common.h ..... used in (all) scripts
├── config
│   └── Makefile
├── build ..... home of the build files
├── target-root
├── target-mount
├── tc ..... the new toolchain
├── src ..... own programs
└── work
    └── → ../config/Makefile
```

Build

die Schritte 1

- `binutils`
- `gcc-bare`
- `kernel` mit ein paar *rules*
  - `bcmrpi_defconfig`
  - `zImage`
  - `headers_install`
- `eglibc`
- `gcc`
- Test
  - im Verzeichnis `work`

Build
die Schritte 2

- `busybox`
  - Installation auf SD-Card
  - `fakeroot`
- `openssh` die volle Implementation
  - `zlib`
  - `openssl`
  - `openssh`
- `sshfs`

Skripts und Argumente

initiales System

`binutils.sh`		
`gcc-bare.sh`		kernel libc
`kernel.sh`	`bcmrpi_defconfig`	
	`zImage`	
	`headers_install`	
`eglibc.sh`		
`gcc.sh`		C/C++ POSIX
`busybox.sh`	busybox	
	install	
`target-root.sh`		

Target
erster Versuch

- transfer auf SD Karte
- Internet

Skripts und Argumente

ssh

`zlib.sh`
`openssl.sh` die kryptographischen Algorithmen
`openssh.sh`

Remark: `openssh.sh` hängt von `zlib.sh` und `openssl.sh` ab

Target

ssh

- `ssh`
- `sshd`
- Keys

sshfs

funktioniert noch nicht

- Die Bibliothek `glib`
- Ersatz
  - `sftp`

Modules

- siehe `6-modules`

Modules

Verzeichnisstruktur

Host

```
BUILD_HOME
├── modules
│   ├── Makefile ..... for own modules
│   └── *.c/h ..... the own sources
└── scripts
    └── module.sh
```

RaspberryPi

```
/
└── work
```

Herstellung
Workflow

Host

- `sh scripts/module.sh` *`module`*
- `scp`, `sftp`, `ftp`

RaspberryPi

- `insmod`
- `rmmod`

Teil I

Installation TFT Display

Um was geht es ?

Adafruit TFT Touchscreen

- Installation auf einem minimalen System
- `git` update
- Konfiguration
- Kernelcompilation
- Driver
- Module
- *framebuffer*

Hardware

- `https://learn.adafruit.com/adafruit-2-8-pitft-capacitive-touch/downloads`
- Zwei Chips
  - `STMPE610` für den *touchscreen*
  - `ILI9341` für die Graphik
- Verbunden per `SPI`

Funktionierendes System

- `http://adafruit-download.s3.amazonaws.com/2015-02-16-raspbian-pitft28r_150312.zip`
- Verbindung per `ssh`

Remark(s):

  - user `pi`
  - pw: `raspberry`

- Host `dhcp`
- Host `nmap`
- Framebuffer `/dev/fb*`
  - alles ist ein File
- Backlight Control

Kernel

Neue Version

- `git pull`
- `drivers/staging/fbtft/fb_ili9341.c`

Module herstellen

- als Modul herstellen: *menuconfig*
  - `FB_TFT_FBTFT_DEVICE`
    - `drivers/staging/fbtft/fbtft_device.c`
  - `FB_TFT_ILI9341`
    - `drivers/staging/fbtft/fb_ili9341.c`
- nur einzelne Module herstellen
  - *dir/file.ko*
    - *dir* relativ zu kernel source

Module ausführen

- zum *target* kopieren
- `insmod` *name parameters*
- `mkdev -s` erzeugt `/dev/fb1`
- siehe auch
  - `/sys/class/graphics/`