barebox – An introduction

Sascha Hauer <s.hauer@pengutronix.de>
Wolfram Sang <w.sang@pengutronix.de>
(ヴォルフラム　ザング)

CELF Japan Technical Jamboree 32
Tokyo, 2010-03-05

barebox: Agenda

Bootloaders: What they do and why we (still) need them

The barebox Project: Motivation for a fork

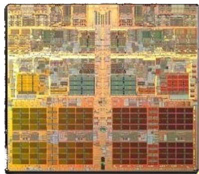
Design Decisions: A Bootloader for Kernel Hackers

Flow of Execution: From Power-On to the Kernel

Sugar and Candies: Some Highlights

Summary & Discussion

Bootloaders: What they do ...

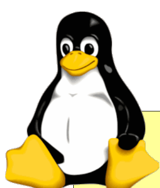


Low Level Hardware Init
RAM, Flash, PLLs + Clocks, ...

BIOS (PC)
Bootloader (SoC)

Fetch Kernel(s) from Boot Medium
NOR-Flash, NAND-Flash, SD, USB, SATA, Network...

BIOS (PC)
Bootloader (SoC, PC)



Start Kernel
with kernel command line parameters

Bootloader (SoC, PC)

Pengutronix.

Requirements for Production Systems

No interaction: power-on and boot
No delays by the bootloader!

The bootloader shall stay out of the way!

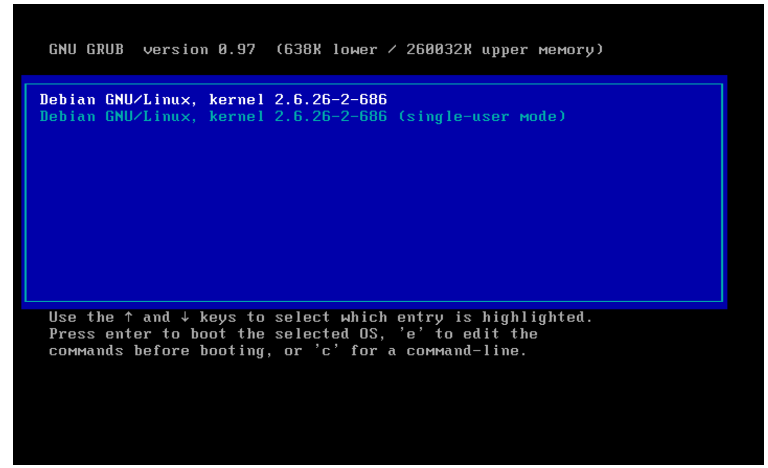
no selection screen
no nothing

Pengutronix.

Requirements: Development & Maintenance

Stop boot process in the bootloader:

key press on keybord (PC)
serial console key or
hardware button (embedded)



Choose between pre-existing kernels

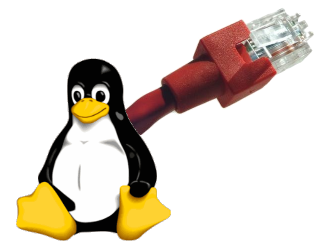
Be able to edit kernel location - where to boot from
(flash partition, disk partition, tftp location, ...)

Change kernel command line

Make changes persistent
(change - store - boot with new config)

Common requirements for Embedded

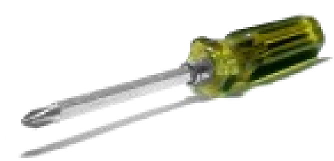
Pre-loader for NAND



TFTP booting the kernel
(for quick development cycle)

Redundancy Boot
(start watchdog, boot,
boot other kernel on startup-failure)

Hardware testing environment:



have register access from a commandline
while kernel was not ported to a
new platform yet

have a non-complex environment for
hardware people to test prototypes

Pengutronix.

barebox: Agenda

Bootloaders: What they do and why we (still) need them

The barebox Project: Motivation for a fork

Design Decisions: A Bootloader for Kernel Hackers

Flow of Execution: From Power-On to the Kernel

Sugar and Candies: Some Highlights

Summary & Discussion

The barebox Project: Motivation for a fork

Pro U-Boot:

„Das U-Boot“ is a succesful bootloader
for SoC type Linux systems!

Multi platform design, runs on
ARM, MIPS, PowerPC, Blackfin etc.

Well established

High level of configurability (saveable environment)

“True” Open Source project
(no hidden development, public git + list)

The barebox Project: Motivation for a fork

Contra U-Boot:

Operating U-Boot requires deep board knowledge

abuse of the environment for scripting

no clean driver model and multi instance concepts

Hard to configure: many macros have to be edited by hand to configure the features of U-Boot

“Must not break existing boards” policy
(makes it hard to change designs)

The barebox Project: Motivation for a fork

barebox was started as a technology study (named U-Boot-V2)

Idea: How can the U-Boot principle be improved when...

... we take proven concepts from Linux

... we "think POSIX"

...(we are allowed to risk breaking unmaintained platforms if really needed)

Can a bootloader feel more like Linux?

barebox: Agenda

Bootloaders: What they do and why we (still) need them

The barebox Project: Motivation for a fork

Design Decisions: A Bootloader for Kernel Hackers

Flow of Execution: From Power-On to the Kernel

Sugar and Candies: Some Highlights

Summary & Discussion

Design Decisions:
A Bootloader for Kernel Hackers

Some design decisions we made for barebox:

more abstraction
(devices instead of direct memory access + special knowledge)

multi instance driver model
(no global variables, "`ethaddr`", "`eth1addr`" etc. any more)

frameworks instead of multiple-drivers-with-(almost-)same-api

"usual" commands: rm, cp, ls, ...

Design Decisions:
A Bootloader for Kernel Hackers

Some design decisions we made for barebox:

scripts are scripts, no “runnable environment variables”

use a shell-like environment

KBuild + Kconfig (easy configuration & build process)

models taken from the Linux kernel (clocks, ...)

kernel coding style

“best of U-Boot and Linux”

barebox: Agenda

Bootloaders: What they do and why we (still) need them

The barebox Project: Motivation for a fork

Design Decisions: A Bootloader for Kernel Hackers

Flow of Execution: From Power-On to the Kernel

Sugar and Candies: Some Highlights

Summary & Discussion

Pengutronix

“Hello World” in barebox

Here is a typical startup from barebox:

```
barebox 2010.02.0-00065-g7aa3161 (Feb  4 2010 - 19:15:42)

Board: Phytec phyCard-i.MX27
NAND device: Manufacturer ID: 0x20, Chip ID: 0x36 (ST Micro NAND 64MiB
   1,8V 8-bit)
Malloc space: 0xa7a00000 -> 0xa7f00000 (size  5 MB)
Stack space : 0xa79f8000 -> 0xa7a00000 (size 32 kB)
running /env/bin/init...

Hit any key to stop autoboot:  3

type update_kernel nand|nor [<imagename>] to update kernel into flash
type update_root nand|nor [<imagename>] to update rootfs into flash

barebox:/
```

File System

During startup, a RAM filesystem is mounted to **/**

A device filesystem is mounted to **`/dev`**

The environment is copied to **`/env`**

At the prompt, the well known commands like '**ls**','**rm**', '**cp**' work the way we are used to:

```
barebox:/ ls
.           ..          dev         env
```

Devices

Drivers can create device nodes under **`/dev`**
which can be accessed like normal files:

```
barebox:/ ls /dev/
zero       defaultenv     mem          nand0          ram0
phy0       self_raw       self0        env_raw        env0
```

Accessing Devices

While **`/dev/mem`** is the default "file" for the memory commands, it can be changed:

```
barebox:/ md -w -s /dev/phy0
00000000: 1000 786d 0022 1613 01e1 45e1 0007 2001 ..mx"......E...
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

This displays the contents of **`/dev/phy0`** (-s) in 16 bit wordsize (-w)

Device Variables

Design criterium: "avoid magic variables"

Introduction of "device variables": **`eth0.ipaddr`**

Device variables can be displayed using the devinfo command:

```
barebox:/ devinfo eth0
base   : 0x00000000
size   : 0x00000000
driver: none

Parameters:
          ipaddr = 192.168.24.26
         ethaddr = 00:50:c2:a5:bb:87
         gateway = 192.168.1.1
         netmask = 255.255.0.0
        serverip = 192.168.23.2
```

The device variables can be used like any other variable:

```
barebox:/ eth0.serverip=192.168.23.123
barebox:/ echo $eth0.serverip
192.168.23.123
```

Partitioning

Device files can be partitioned to get a convenient access to flashes and to get a consistent partition layout between barebox and Linux:

```
barebox:/ addpart nor0 256k(barebox),128k(barebox-env),2M(kernel),-
   (rootfs)
barebox:/ ls -l /dev/nor0*
crw-------   31064064 /dev/nor0.rootfs
crw-------    2097152 /dev/nor0.kernel
crw-------     131072 /dev/nor0.barebox-env
crw-------     262144 /dev/nor0.barebox
crw-------   33554432 /dev/nor0
```

The partition description for the “addpart” command is the same as the Linux mtd layer uses for command line partitioning, so this string can be directly given to the Kernel

Pengutronix.

Getting the Kernel via TFTP

Most important networking commands:

dhcp (configure the network adapter)
tftp (transfer files via tftp, can write directly to flash)

```
barebox:/ erase /dev/nor0.kernel
barebox:/ dhcp
phy0: Link is up - 100/Full
BOOTP broadcast 1
DHCP client bound to address 192.168.24.26

barebox:/ tftp zImage-pca100 /dev/nor0.kernel
phy0: Link is up - 100/Full
TFTP from server 192.168.23.2; our IP address is 192.168.24.26
Filename 'zImage-pca100'.
Loading: #################################################################
         #################################################################
         #################################################################
done
Bytes transferred = 1815016 (1bb1e8 hex)
```

Build System

Building should be familiar to kernel hackers:

```
# export ARCH=arm
# export CROSS_COMPILE=arm-linux-
# make pcm038_defconfig
# make menuconfig
# make
```


Pengutronix.

barebox: Agenda

Bootloaders: What they do and why we (still) need them

The barebox Project: Motivation for a fork

Design Decisions: A Bootloader for Kernel Hackers

Flow of Execution: From Power-On to the Kernel

Sugar and Candies: Some Highlights

Summary & Discussion

Sugar & Candies: Highlights

Minimal porting effort

Can be a 2nd stage bootloader (e.g if first level is proprietary)

“One-image-starts-everywhere” (NAND, NOR, RAM...)

Highly scalable (modular)

Support for MMU, USB (host/device), DFU, splashscreens,...

GPL v2 (as the kernel)

integrated shell & editor

Binary size usually from 9KB to 150KB

Sandbox

barebox can be built as a normal Linux binary
(inspired by user mode linux)

Features:

Working on barebox without real hardware
Run under gdb
Networking using a tap device

To compile the sandbox: pass `ARCH=sandbox` while compiling

Showtime!

See barebox in action

barebox: Agenda

Bootloaders: What they do and why we (still) need them

The barebox Project: Motivation for a fork

Design Decisions: A Bootloader for Kernel Hackers

Flow of Execution: From Power-On to the Kernel

Sugar and Candies: Some Highlights

Summary & Discussion

Summary & Discussion

barebox wants to give you a Linux feeling inside a bootloader

- when working on the command-line
- when developing for it

So developers feel at home and can work faster making less errors (hopefully :)).

It aims for great flexibility to meet a number of use cases.

Alternative: boot with Linux (kexec)

initial porting effort?
penalties in boot time and storage consumption?
NAND pre-loader?

Pengutronix

Links

Web Site for barebox:
http://www.barebox.org/

Source Code:
git clone git://git.pengutronix.de/git/barebox.git

Mailing List:
http://lists.infradead.org/mailman/listinfo/barebox

Questions? Ask me now or anytime later:

w.sang@pengutronix.de