How to boot Linux in one second
...why userland is a waste of time ;)

Jan Altenberg

Linutronix GmbH
from zero...
to hero...
Overview

1 Basics
   Motivation
   Some technical basics

2 Optimizations
   Bootloader
   Kernel
   Filesystem
   Application

3 Example
   Optimizing an ARMv5 based device
   Optimizing the test system
Motivation

1 Basics
Motivation
Some technical basics

2 Optimizations
Bootloader
Kernel
Filesystem
Application

3 Example
Optimizing an ARMv5 based device
Optimizing the test system
Motivation

- "marketing"
- automotive applications
- energy saving
Motivation

- "marketing"
- automotive applications
- energy saving
  - solution: power-off instead of suspending
Motivation

- "marketing"
- automotive applications
- energy saving
  - solution: power-off instead of suspending
  - BUT: Users are not used to wait
Some technical basics

1 Basics
   Motivation
   Some technical basics

2 Optimizations
   Bootloader
   Kernel
   Filesystem
   Application

3 Example
   Optimizing an ARMv5 based device
   Optimizing the test system
Some technical basics

First step: Define your requirements!!!!

- What's the limit for the boot time?
- Which functionality should be available?
- Speed vs. flexibility

NOTE: FastBOOT is not a product, it's a concept!!
Some technical basics

Boot process

Application

Operating System

Bootloader

Hardware
Components of the boot process

- Hardware reset
- Bootloader
- Operating System (drivers, filesystem, ...)
- INIT process, application (userland)
Some technical basics

Critical hardware components

- Power supply
- Reset logic
- Boot logic / boot order
- Boot media
- Peripherals which need to be accessed while booting

**IMPORTANT:** the hardware is a central part of a fastboot concept!!!
Some technical basics

Bootloader

- Basic setup of the CPU
- Preparing and handing over ATAGS / devicetree
- Flushing the caches
- Switch off the MMU
The Linux Kernel

- A lot of functions for boot time optimization
- Very flexible
- Configurable compression type
- Can defer or parallelize initializations
- 150ms - 250ms from starting the kernel to mounting the RFS
Some technical basics

The application

- Usually the biggest target for optimizations
- Start scripts / INIT process
- Linking
1 Basics
   Motivation
   Some technical basics

2 Optimizations
   Bootloader
   Kernel
   Filesystem
   Application

3 Example
   Optimizing an ARMv5 based device
   Optimizing the test system
Optimizing the bootloader (U-Boot)

Remove unused features:

/* include/configs/boardname.h */
 [...]  
#include <config_cmd_default.h>
#undef CONFIG_CMD_NET
 [...]
Optimizing the bootloader (U-Boot) 2

Verifying the kernel image:

```
setenv verify n
```

Switch off the bootloader console:

```
setenv silent 1
```

Switch off the boot delay:

```
setenv bootdelay 0
```
Optimizing the bootloader: IPL / SPL

- Replacing the general purpose bootloader by an optimized IPL
- ...also useful for update concepts
- U-Boot offers a generic way: The U-Boot SPL (CONFIG_SPL_OS_BOOT)
1 Basics
Motivation
Some technical basics

2 Optimizations
Bootloader
Kernel
Filesystem
Application

3 Example
Optimizing an ARMv5 based device
Optimizing the test system
Optimizing the kernel

- Configuration and build
- Compression method
- Boot parameters (kernel commandline)
- Driver init calls
- Rootfilesystem (RFS)
Optimizing the kernel: Configuration

General setup
Kernel compression mode

- LZO usually a good choice for embedded system
- Copy vs. de-compress
- "Execute in Place (XIP)"
Delay Loop Calibration: "lpj="; can save > 100ms on ARMv5 based systems

Parameters for boot time analysis: "initcall_debug", "printk_time=1"
Kernel

Optimizing the kernel: printk.time

[0.000000] VIC @f1140000: id 0x00041190, vendor 0x41
[0.000000] FPGA IRQ chip 0 "SIC" @ f1003000, 21 irqs
[0.000000] Console: colour dummy device 80x30
[0.018847] Calibrating delay loop...
  626.68 BogoMIPS (lpj=3133440)
[0.316717] pid_max: default: 32768 minimum: 301
[0.317552] Mount-cache hash table entries: 512
...
Optimizing the kernel: Delay Loop

[0.018847] Calibrating delay loop...
  626.68 BogoMIPS (lpj=3133440)
[0.316717] pid_max: default: 32768 minimum: 301
Optimizing the kernel: initcall_debug

[0.452115] calling exceptions_init+0x0/0x90 @ 1
[0.452172] initcall exceptions_init+0x0/0x90
    returned 0 after 0 usecs
[0.452203] calling versatile_i2c_init+0x0/0x24 @ 1
[0.452321] initcall versatile_i2c_init+0x0/0x24
    returned 0 after 0 usecs
[0.452352] calling pl011_init+0x0/0x54 @ 1
[0.452382] Serial: AMBA PL011 UART driver
[0.453647] dev:f1: ttyAMA0 at MMIO 0x101f1000
  (irq = 12) is a PL011 rev1
[0.481540] console [ttyAMA0] enabled
...
[0.484427] initcall pl011_init+0x0/0x54
    returned 0 after 29296 usecs
Kernel

bootgraph.pl

1. **Boot your system with** "initcall_debug loglevel=8"
2. **On the target:**
   
   ```
   $ dmesg > bootlog.txt
   ```
3. **On the host:**
   
   ```
   $ cd linux-XXX
   $ cat /path_to_rfs/bootlog.txt | perl scripts/bootgraph.pl > bootlog.svg
   ```
Kernel

scripts/bootchart.pl
1 Basics
   Motivation
   Some technical basics

2 Optimizations
   Bootloader
   Kernel
   Filesystem
   Application

3 Example
   Optimizing an ARMv5 based device
   Optimizing the test system
UbiFS

- The best choice for flash devices
- Power-Fail safe
- The underlying UBI layer can be optimized with (FastMAP)
- ...

Linutronix GmbH
Filesystem

InitRAMFS

```
dir /dev 755 0 0
nod /dev/console 644 0 0 c 5 1
nod /dev/loop0 644 0 0 b 7 0
dir /bin 755 1000 1000
slink /bin/sh busybox 777 0 0
file /bin/busybox initfs/busybox 755 0 0
[...

dir /proc 755 0 0
dir /sys 755 0 0
dir /mnt 755 0 0
```
The INIT process for the InitRAMFS can be configured with rdinit=. For example: rdinit=/etc/init.d/start.sh
InitRAMFS: Switch root

/etc/init.d/start.sh:

#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs devtmpfs /dev

# Mount RFS / do some critical stuff
mount /dev/mmcblk0p1 /media
fbsplash -s /media/splash.ppm -d /dev/fb0

mount -o move /proc /media/proc
mount -o move /sys /media/sys
mount -o move /dev /media/dev

# Switch to production system
exec switch_root /media /linuxrc
Application

1. Basics
   Motivation
   Some technical basics

2. Optimizations
   Bootloader
   Kernel
   Filesystem
   Application

3. Example
   Optimizing an ARMv5 based device
   Optimizing the test system
The INIT process

- SystemV
- SystemD

One letter makes a BIG difference ;-}
Optimizing the application

- Analyse the INIT process with bootchartd or systemd-analyze
- Replace the INIT process with your own application (init=)
- Linking
- Pre-Linking and function reordering
Moving start script tasks into your application

```c
ret = mount("sysfs", "/sys",
    "sysfs", 0, NULL);
if(ret < 0)
    perror("Can't mount sysfs\n");
```
Application

Dynamic linking

1. ELF DT_RPATH section
2. LD_LIBRARY_PATH
3. ELF DT_RUNPATH section
4. Binary file /etc/ld.so.cache
5. Default paths /lib und /usr/lib

Jan Altenberg
Linutronix GmbH
Dynamic linking: Debug and visualize

```bash
$ LD_DEBUG=libs ls
 3082: find library=librt.so.1 [0];
      searching
 3082: search cache=/etc/ld.so.cache
 3082: trying file=/lib/librt.so.1
```
1 Basics
   Motivation
   Some technical basics

2 Optimizations
   Bootloader
   Kernel
   Filesystem
   Application

3 Example
   Optimizing an ARMv5 based device
   Optimizing the test system
Test system

- ARM9 CPU (Atmel AT91 series)
- Starting point: Busybox based image (Angstrom Distribution)
- Boot media: NAND-Flash
- Test application: Toggling a GPIO via SysFS
Optimizing an ARMv5 based device

Boot strategy of the AT91 controller family

Application
Linux Kernel
U-Boot
AT91 Bootstrap
Hardware
Optimizing an ARMv5 based device

Bootmodes of the AT91 controller family

- RomBOOT: internal boot logic
- External bus interface (CS0, e.g. NOR flash)
Optimizing an ARMv5 based device

**AT91 RomBOOT**

- **Application**
- **Linux Kernel**
- **U-Boot**
- **AT91 Bootstrap**
- **ROM Boot**
- **Reset Logic**
- **Hardware**
Optimizing an ARMv5 based device

Power supply

![Graph showing power supply waveform](image-url)
Optimizing an ARMv5 based device

Reset logic

---

Wittig Technologies AG - W2000 Series - 09.05.2010 14:27:45 - DP_Scope01
Timebase: 200 ms / Div  Trigger: Channel 1  Trigger Level: 2.7 V

ca. 1200 ms
Optimizing an ARMv5 based device

RomBOOT

\[\text{Wireg Technologies AG - W2000 Series - 09.05.2010 14:37:24 - QP_Scope01}\
\text{Timebase: 50 ns / Div - Trigger: Channel 1 - Trigger Level: 1.3 V}\

\text{ca. 120 ms}
Optimizing an ARMv5 based device

Possible hardware optimizations

- Using the internal oscillator for deriving the slowclock saves > 1s!!
- booting from CS0 will save 100 - 150ms
Optimizing the test system

1 Basics
   Motivation
   Some technical basics

2 Optimizations
   Bootloader
   Kernel
   Filesystem
   Application

3 Example
   Optimizing an ARMv5 based device
   Optimizing the test system
Optimizing the test system

Boot time measurements with a GPIO

AT91 Bootstrap → U-Boot → Kernel → Application
Optimizing the test system

Measuring points

- **Bootstrap - U-Boot**
- **U-Boot - Early-Boot-Code of the kernel (incl. relocation and decompression)**
- **Kernel - application (incl. mounting the RFS)**
Optimizing the test system

Initial boot time
Optimizing the test system

Initial boot time

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - u-boot</td>
<td>---</td>
</tr>
<tr>
<td>u-boot - kernel</td>
<td>6,5s</td>
</tr>
<tr>
<td>kernel - application</td>
<td>4,5s</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>11s</strong></td>
</tr>
</tbody>
</table>
Simple optimizations
Optimizing the test system

U-Boot w/o networking support

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - u-boot</td>
<td>---</td>
</tr>
<tr>
<td>u-boot - kernel</td>
<td>4,25s</td>
</tr>
<tr>
<td>kernel - application</td>
<td>4,5s</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>8,75s</strong></td>
</tr>
</tbody>
</table>
Optimizing the test system

U-Boot verify=n

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - u-boot</td>
<td>---</td>
</tr>
<tr>
<td>u-boot - kernel</td>
<td>3.89s</td>
</tr>
<tr>
<td>kernel - application</td>
<td>4.5s</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>8.39s</strong></td>
</tr>
</tbody>
</table>
Optimizing the test system

Optimizing the kernel config

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - u-boot</td>
<td>---</td>
</tr>
<tr>
<td>u-boot - kernel</td>
<td>3,77s</td>
</tr>
<tr>
<td>kernel - application</td>
<td>4,33s</td>
</tr>
<tr>
<td>total</td>
<td>8,1s</td>
</tr>
</tbody>
</table>
Optimizing the test system

Analyzing the INIT process: Bootchartd
Optimizing the test system

Optimizing the start scripts

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - u-boot</td>
<td>---</td>
</tr>
<tr>
<td>u-boot - kernel</td>
<td>3,77s</td>
</tr>
<tr>
<td>kernel - application</td>
<td>3,61</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>7,38s</strong></td>
</tr>
</tbody>
</table>
Optimizing the test system

Booting an InitRAMFS
Optimizing the test system

LZO compressed InitRAMFS

The test application is used as an INIT process (rdinit=)

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - u-boot</td>
<td>---</td>
</tr>
<tr>
<td>u-boot - kernel</td>
<td>3,79s</td>
</tr>
<tr>
<td>kernel - application</td>
<td>0,372s</td>
</tr>
<tr>
<td>total</td>
<td>4,162s</td>
</tr>
</tbody>
</table>
Optimizing the test system

Modified AT91 Bootstrap
Optimizing the test system

Modified AT91 Bootstrap

AT91 Bootstrap starts Linux (without U-Boot)

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - kernel</td>
<td>676ms</td>
</tr>
<tr>
<td>kernel - application</td>
<td>584ms</td>
</tr>
<tr>
<td>total</td>
<td>1,260s</td>
</tr>
</tbody>
</table>
Optimizing the test system

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - kernel</td>
<td>676ms</td>
</tr>
<tr>
<td>kernel - application</td>
<td>384ms</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>1,060s</strong></td>
</tr>
</tbody>
</table>
Optimizing the test system

< 1s !!
Optimizing the test system

No (serial) console output (quiet)

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - kernel</td>
<td>524ms</td>
</tr>
<tr>
<td>kernel - application</td>
<td>212ms</td>
</tr>
<tr>
<td>total</td>
<td>736ms</td>
</tr>
</tbody>
</table>
Optimizing the test system

**LZO compressed kernel image**

<table>
<thead>
<tr>
<th>measuring point</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap - kernel</td>
<td>444ms</td>
</tr>
<tr>
<td>kernel - application</td>
<td>212ms</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>656ms</strong></td>
</tr>
</tbody>
</table>
Optimizing the test system

Final boot behaviour
Conclusion

- Linux can combine the advantages of a modern OS with hard boot time requirements
- Saving boot time with simple optimizations
- The hardware is an IMPORTANT part of a FastBOOT concept
- The boot concept is architecture independent!
Optimizing the test system

Questions?

I'll also be around at the technical showcase! :)