Quantitative analysis of system initialization in embedded Linux systems

André Puschmann
andre.puschmann@imms.de

IMMS Institut für Mikroelektronik- und Mechatronik-Systeme gGmbH
Ehrenbergstrasse 27
98693 Ilmenau, Germany

January 26, 2009

Table of Contents

1. Introduction
   - Motivation and Goals
   - System setup
   - The boot process
2. Instrumentation
   - Measurement
   - Reference system
3. Boot time reduction
   - Overview
   - Universal techniques
   - Bootloader layer
   - Kernel layer
   - Application layer
4. Conclusion
   - Results
   - Conclusion and outlook

Motivation

Motivation

- Linux widely used:
  e.g. automation, consumer electronic devices, ...
- Time from power up to “usable” system is mission critical

Example

Consider a car audio system which needs 5 minutes for start up.
Would you buy it?

Motivation

Motivation

- Linux widely used:
  e.g. automation, consumer electronic devices, ...
- Time from power up to “usable” system is mission critical

Example

Consider a car audio system which needs 5 minutes for start up.
Would you buy it?

Goals

Goals

- Identify most critical and time consuming sections
  ⇒Ensure compliance with *"Pareto principle"* (80-20 rule)
- Survey techniques for boot time reduction
- Evaluation on *Gumstix verdex XM* (XScale PXA270)

System setup

Hardware platform

- *Gumstix verdex XM4* mainboard with *netpro vx* extension
- *Marvel XScale PXA270* at 400 *Mhz* (ARM V5TE architecture)
- 64 *MB* RAM
- 16 *MB* *Intel* NOR flash

Software platform

- Based on *gumstix-buildroot* environment
  - *u-boot* bootloader
  - Linux kernel 2.6.21

System setup

Hardware platform

- *Gumstix verdex XM4* mainboard with *netpro vx* extension
- *Marvel XScale PXA270* at 400 *Mhz* (ARM V5TE architecture)
- 64 *MB* RAM
- 16 *MB* *Intel* NOR flash

Software platform

- Based on *gumstix-buildroot* environment
  - *u-boot* bootloader
  - Linux kernel 2.6.21

The boot process

"Time from power on to a "usable" system".

Phases

- Bootloader phase:
  - Initial hardware setup
  - Load "more complicated system"
- Kernel phase:
  - Most intrinsic part
  - Initializes all hard- and software components
- Application phase:
  - Longest phase

Instrumentation (1)

What we haven't done:

- Kernelspace measurement
- Userspace measurement

What we have done:

- Software based test points that toggle GPIO pins
- Each boot phase divided into one or more sections
- Oscilloscope to measure time-span between logical values

Instrumentation (2)

Reference system

- Necessary for meaningful comparison
  ⇒"Don't compare apples and oranges!"
- But: very application specific:
  ⇒Use system with most common components

- *Buildroot* based
- u-boot, Linux kernel, root-fs
- *Busybox*: "The Swiss Army Knife of Embedded Linux"
- System libraries: C/C++, pthread, ...

Overview

"100% IO and CPU workload are the ideal world."

Layer oriented:

- Universal techniques
- Bootloader layer techniques
- Kernel layer techniques
- Application layer techniques

Universal techniques (1)

File and code size

- Board support packages are often full blown with features
- Only enable features that you really need

Compression

- Less space is needed. But is not necessarily faster
- Weigh up speed of storage media and decompression speed

Execution in place

- Execute code directly from non-volatile memory

Pros: Less power and RAM consumption
Lower time-to-start

Cons: Needs more (expensive NOR) flash
Possible lower overall throughput

Universal techniques (1)

File and code size

- Board support packages are often full blown with features
- Only enable features that you really need

Compression

- Less space is needed. But is not necessarily faster
- Weigh up speed of storage media and decompression speed

Execution in place

- Execute code directly from non-volatile memory

Pros: Less power and RAM consumption
Lower time-to-start

Cons: Needs more (expensive NOR) flash
Possible lower overall throughput

Universal techniques (1)

File and code size

- Board support packages are often full blown with features
- Only enable features that you really need

Compression

- Less space is needed. But is not necessarily faster
- Weigh up speed of storage media and decompression speed

Execution in place

- Execute code directly from non-volatile memory

Pros: Less power and RAM consumption
Lower time-to-start

Cons: Needs more (expensive NOR) flash
Possible lower overall throughput

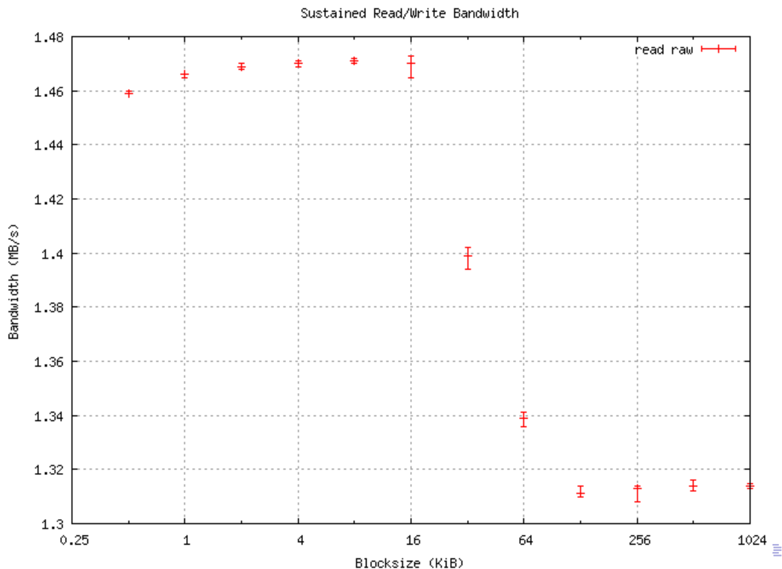
Universal techniques (2)

Memory configuration

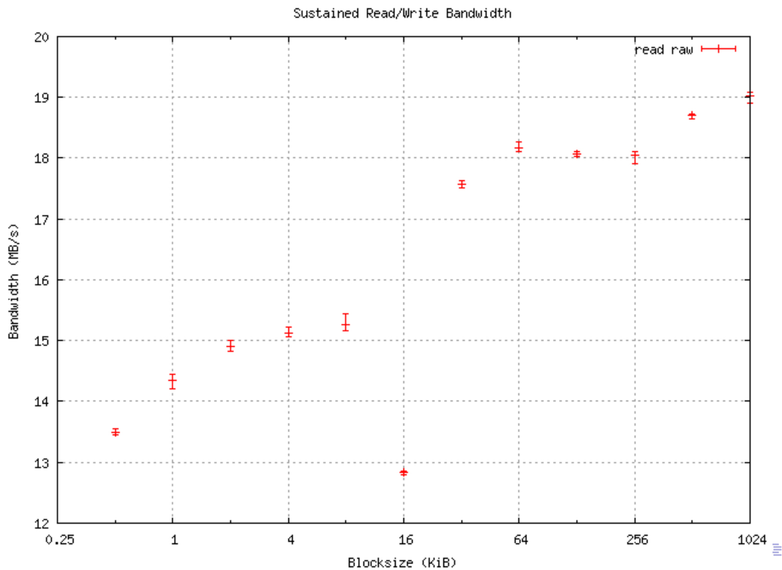
- Fully use the capabilities of the memory chips
- RAM is much faster than flash $\Rightarrow$ flash is bottleneck
- Review of memory timing configuration
- Review of cache configuration
- Flash performance analysis with *flanatoo*[a] (and `cp`)

[a] *written by Daniel Parthey* at *Chemnitz University of Technology*

Flash read performance (1) - Original



Flash read performance (2) - Optimized



Bootloader layer techniques

Effort justified?

- Bootloader phase only of relatively short duration
- Optimization quite complex
- Promises only marginal improvement
- But: General rules also apply here
- Don't spent too much time, unless there is a good reason

Little orientation:

- u-boot start up:
  - unoptimized: 150 *ms*
  - optimized: 135 *ms*

Bootloader layer techniques

Effort justified?

- Bootloader phase only of relatively short duration
- Optimization quite complex
- Promises only marginal improvement
- But: General rules also apply here
- Don't spent too much time, unless there is a good reason

Little orientation:

- u-boot start up:
  - unoptimized: 150 *ms*
  - optimized: 135 *ms*

Kernel layer techniques (1)

Kernel image type comparison

	normal (non-XIP)	XIP kernel	ZBOOT_ROM kernel
size	885.76 kB	1,826.63 kB	885.76 kB
decompress/copy	730 ms	65 ms	616 ms
kernel init	275 ms	535 ms	275 ms
mount/start root fs	679 ms	1,156 ms	679 ms
total	1,684 ms	1,756 ms	1,570 ms

Kernel layer techniques (2)

Eliminate console output:

- Append `quiet` parameter to kernel command line
- Reduction of fairly 300 *ms*

Eliminate lpj calculation:

- Append calculated `lpj` parameter to kernel command line
- Reduction of 200 *ms*

Driver initialization:

- Avoid firmware loading/hardware probing overhead
- Deferred and concurrent driver initialization with modules
- Improvement driver specific

Kernel layer techniques (2)

Eliminate console output:

- Append **`quiet`** parameter to kernel command line
- Reduction of fairly 300 *ms*

Eliminate lpj calculation:

- Append calculated **`lpj`** parameter to kernel command line
- Reduction of 200 *ms*

Driver initialization:

- Avoid firmware loading/hardware probing overhead
- Deferred and concurrent driver initialization with modules
- Improvement driver specific

Kernel layer techniques (2)

Eliminate console output:

- Append `quiet` parameter to kernel command line
- Reduction of fairly 300 *ms*

Eliminate lpj calculation:

- Append calculated `lpj` parameter to kernel command line
- Reduction of 200 *ms*

Driver initialization:

- Avoid firmware loading/hardware probing overhead
- Deferred and concurrent driver initialization with modules
- Improvement driver specific

Application layer techniques (1)

Filesystem

1. *CramFS*, access through mtd layer [~2.2 MiB]
2. *Linear CramFS*, XIP disabled [~2.2 MiB]
3. *Linear CramFS*, busybox marked as XIP [~2.2 MiB]
4. *AXFS*, no XIP, profiling off, **physaddr**-flag [~2.2 MiB]

	1	2	3	4
mount filesystem	0.016 s	0.024 s	0.024 s	0.012 s
start user scripts	1.119 s	0.892 s	0.935 s	1.019 s
start application	0.140 s	0.100 s	0.150 s	0.110 s
copy file	0.150 s	0.110 s	0.060 s	0.050 s
total	1.425 s	1.126 s	1.169 s	1.191 s

Application layer techniques (2)

Device node population

- *udev* / *mdev* increase system comfort, quality and start up time
- Way out: Copy static nodes, start hotplug-daemon afterwards
- Reduction of 200 *ms*

Parallel init script execution

- Replace *init* with *myinit*[a]
- Allows parallel execution and dependencies
- Potential savings application specific
- Here: Reduction of 200 *ms*

[a] written by Stamatis Mitrofanis, available at `sourceforge.net`

Application layer techniques (2)

Device node population

- *udev*/*mdev* increase system comfort, quality and start up time
- Way out: Copy static nodes, start hotplug-daemon afterwards
- Reduction of 200 *ms*

Parallel init script execution

- Replace *init* with *myinit*[a]
- Allows parallel execution and dependencies
- Potential savings application specific
- Here: Reduction of 200 *ms*

[a] written by Stamatis Mitrofanis, available at `sourceforge.net`

Conclusion (1)

Results:

- Presentation of several techniques for boot time reduction
- Evaluation on widely used PXA270 hardware

	partly optimized	optimized
bootloader phase	152 ms	135 ms
kernel copy/decompression	814 ms	600 ms
kernel initialization	1,343 ms	247 ms
application phase	1,047 ms	488 ms
total	3,356 ms	1,470 ms

Conclusion (2)

Conclusion:

- Boottime always discussed topic in embedded Linux
- Bootstrapping of complex systems is a challenging task
  - Divide process into phases
  - Ensure compliance with *Pareto principle* (80-20 rule)

Outlook

- Further research necessary
- Things on our To-do list:
  - Impact of techniques like prelinking and library optimization
  - Flash in synchronous operation mode
  - *AXFS* with XIP enabled

Conclusion (2)

Conclusion:

- Boottime always discussed topic in embedded Linux
- Bootstrapping of complex systems is a challenging task
  - Divide process into phases
  - Ensure compliance with *Pareto principle* (80-20 rule)

Outlook

- Further research necessary
- Things on our To-do list:
  - Impact of techniques like prelinking and library optimization
  - Flash in synchronous operation mode
  - *AXFS* with XIP enabled

The End[1]

Questions, comments?!?

[1] bibliography is listed in the paper