The Internet of Things and Life beyond Linux

Embedded Linux Conference Europe 2016

Prof. Dr. Wolfgang Mauerer
Siemens AG, Corporate Research and Technologies
Smart Embedded Systems

Corporate Competence Centre Embedded Linux
Overview

1 Introduction

2 IoT vs. Linux: Conceptual Differences

3 IoT OSes: Properties

4 Development with RTEMS
   - Application Style
   - Building RTEMS systems
   - Example: Networked Appliance with Dynamic Language
Outline

1 Introduction

2 IoT vs. Linux: Conceptual Differences

3 IoT OSes: Properties

4 Development with RTEMS
   - Application Style
   - Building RTEMS systems
   - Example: Networked Appliance with Dynamic Language
Introduction

Target Audience

- Linux is your standard development target
- Your embedded boards come with Linux/Android
- Open Source is default choice
- “Standard embedded engineer”

TA Check

- Linux application development?
- Embedded Linux system development?
- Deeply embedded systems development?
Introduction

Target Audience

- Linux is your standard development target
- Your embedded boards come with Linux/Android
- Open Source is default choice
- “Standard embedded engineer”

TA Check

- Linux application development?
- Embedded Linux system development?
- Deeply embedded systems development?
Introduction

Target Audience

- Linux is your standard development target
- Your embedded boards come with Linux/Android
- Open Source is default choice
- “Standard embedded engineer”

TA Check

- Linux *application* development?
- Embedded Linux *system* development?
- *Deeply embedded systems* development?
Introduction

Target Audience

- Linux is your standard development target
- Your embedded boards come with Linux/Android
- Open Source is default choice
- “Standard embedded engineer”

TA Check

- Linux *application* development?
- Embedded Linux *system* development?
- *Deeply embedded systems* development?
Internet of Things: What is it about?

- Wireless sensor networks, home control
- Ubiquitous connectivity
  - Novel communication approaches (non-IP mesh networks)
  - Not covered in this presentation
- 2020: 25-30 billion devices
- Hardware costs extremely important
Hardware for IoT I

### Infinite Ressources

- Supermarket class smartphone: 2GiB RAM, 2-4 cores
- Raspberry Pi: 1GiB RAM, 4 cores

### Deeply Embedded: Cortex-M class

- NXP: ≈ 200 devices, TI: ≈ 400 devices
- On-Board memory, 100s of KiB
- Too large for bare metal programming, too small for Linux
- Available during the last 20 years
- Likely not going away any time soon
## Hardware for IoT II

<table>
<thead>
<tr>
<th>ARM offerings</th>
<th>Networked Node</th>
<th>Embedded Cntrl.</th>
<th>Embedded Comp.</th>
<th>Embedded Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0/M0+/M3/M4</td>
<td>M4/7,A9,R4/5/7</td>
<td>ARM A9/A35,R7</td>
<td>ARM A53/A72</td>
<td></td>
</tr>
<tr>
<td>ARM M4/7,A9,R4/5/7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel offerings</td>
<td>Quark MCU</td>
<td>Quark SoC</td>
<td>Atom</td>
<td>Core, Xeon</td>
</tr>
<tr>
<td>Quark MCU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture, clock</td>
<td>32-bit, &lt;500 MHz</td>
<td>32-bit, &lt;1 GHz</td>
<td>32/64-bit, &lt;2 GHz</td>
<td>64-bit, &gt;2 GHz</td>
</tr>
<tr>
<td>32/64-bit, &lt;2 GHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MiBs</td>
<td>GiBs</td>
<td>GiBs</td>
<td>TiBs</td>
<td></td>
</tr>
<tr>
<td>MiBs</td>
<td>&lt; 8 MiB</td>
<td>&lt; 1 GiB</td>
<td>&lt; 4 GiB</td>
<td>&gt; 4 GiB</td>
</tr>
<tr>
<td>non-volatile storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; 8 MiB</td>
<td>&lt; 1 GiB</td>
<td>&lt; 4 GiB</td>
<td>&gt; 4 GiB</td>
<td></td>
</tr>
<tr>
<td>Arduino class board</td>
<td>Sensor, field device</td>
<td>control systems</td>
<td>special purpose &amp; server based controllers</td>
<td></td>
</tr>
<tr>
<td>RPI class board</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SoC-FPGA (Zync,…)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW ref. platform</td>
<td></td>
<td></td>
<td></td>
<td>Industrial PC</td>
</tr>
<tr>
<td>Sensor, field device</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>application examples</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLC, IoT node</td>
<td>gateways</td>
<td></td>
<td>multi-purpose controllers</td>
<td></td>
</tr>
<tr>
<td>multi-purpose controllers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Network maintainer’s point of view

What parts would you remove to get the footprint down for a 2MB single purpose machine?
I wouldn’t use Linux, end of story.
Maybe two decades ago, but not now, those days are over.

(Response to net diet patch series)
Alternatives

Linux Weltanschauung

“Linux has a lot more longevity and generality than most embedded OSes. Most such OSes are proprietary. All of them lack the range of capabilities, drivers, and general level of code quality and review found in Linux. Most have far smaller communities (or no communities at all).”

tiny.wiki.kernel.org/faq
Weltanschauung and Veracity

- Many parts of Linux: Very high quality
- Tremendous complexity. Necessary?
- “Corner Cases” like real-time: Community?
Consequences I

Perception

“A growing kernel makes it hard for the people who are trying to build tiny systems, forcing them to go to a proprietary real-time operating system instead.”

Alternative RTOSes

- RTEMS, eCos, Contiki, RIOT, mbed, FreeRTOS, uclinux, threadX...
- ⟨Favourite proprietary OS⟩
Consequences II

No communities at all?

Operating System

- Preempt RT
- RTEMS
- Riot

Postings (weekly median)

Date


Biased, naturally: Specific feature vs. complete OS
Outline

1. Introduction
2. IoT vs. Linux: Conceptual Differences
3. IoT OSes: Properties
4. Development with RTEMS
   - Application Style
   - Building RTEMS systems
   - Example: Networked Appliance with Dynamic Language
Δ(IoT, Linux): Address Spaces & Execution
△(IoT, Linux): Scheduling

RT scheduling//determinism//scheduling choices//pre-computed schedules//time vs. event based
Δ(IoT, Linux): Building Appliances

App packaging//highly configurable kernels
IoT vs. Linux: Legal Caveats

- Linux: Transition Kernel ↔ Userland: *license barrier*
- IoT: Kernel + “Userland” in single address space
- Code (statically) linked together
- Some licenses: Implications on payload code, up to inheriting OS license!
IoT vs. Linux: Commonalities

IoT and Linux: Commonalities

- Toolchain (Cross Building), Build System, Version Control
- Debugging Mechanisms
- Standard C/C++ programming techniques
- Non-system libraries: Custom API
  - POSIX is overrated!
  - IoT system libraries: Yet another library…
Outline

1. Introduction

2. IoT vs. Linux: Conceptual Differences

3. IoT OSes: Properties

4. Development with RTEMS
   - Application Style
   - Building RTEMS systems
   - Example: Networked Appliance with Dynamic Language
# High-Level Comparison

<table>
<thead>
<tr>
<th>System</th>
<th>POSIX</th>
<th>Maturity</th>
<th>VM</th>
<th>Archs</th>
<th>Drivers</th>
<th>Ressources</th>
<th>Docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>X</td>
<td>high</td>
<td>✓</td>
<td>high</td>
<td>high</td>
<td>low</td>
<td>good</td>
</tr>
<tr>
<td>RTEMS</td>
<td>✓</td>
<td>very high</td>
<td>X</td>
<td>high</td>
<td>very high</td>
<td>avg</td>
<td>very good</td>
</tr>
<tr>
<td>µClinux</td>
<td>✓</td>
<td>avg</td>
<td>X</td>
<td>avg</td>
<td>high</td>
<td>avg</td>
<td>poor</td>
</tr>
<tr>
<td>mbed</td>
<td>X</td>
<td>high</td>
<td>X</td>
<td>low</td>
<td>low</td>
<td>avg</td>
<td>very good</td>
</tr>
<tr>
<td>Zephyr</td>
<td>X</td>
<td>high</td>
<td>X</td>
<td>low</td>
<td>avg</td>
<td>low</td>
<td>avg</td>
</tr>
</tbody>
</table>
RTEMS – Real-Time Executive for Multiprocessor Systems

- Extensive support for CPU architectures
- Comprehensive board support
- Commercial Vendors involved in development
- POSIX/Berkeley sockets support
- FreeBSD and LWIP networking stacks
- MPU, but no MMU support

www.rtems.org

≈ 250KiB
FreeRTOS

- Virtual address space support
- Dynamic task creation
- Very small community
- Multiple TCP/IP stacks; Berkeley sockets API

- www.freertos.org
- A few KiB onwards
mbed

- Restricted to ARM targets (Cortex-M)
- Ease of use for (first-time) developers: Web-based dev environment
- Very easy deployment of applications (w/ suitable HW)
  - Upload binary to mass storage device
  - HW target support required (mbed HDK)
- High level C++ SDK

ARM\textsuperscript{mbed}

- www.mbed.org
- \( \approx 512 \text{ KiB} \)
Zephyr

- Linux Foundation Community Project
- Build system and programming style very similar to Linux kernel
- Nano- and Microkernel with different APIs/capabilities
  - Nanokernel: Single task + ISR + fibres (non-preemptive; cooperative multitasking)
  - Microkernel: Multiple tasks (+ ISR + fibres); preemptive multitasking

www.zephyrproject.org

A few KiB onwards
Outline

1. Introduction

2. IoT vs. Linux: Conceptual Differences

3. IoT OSes: Properties

4. Development with RTEMS
   - Application Style
   - Building RTEMS systems
   - Example: Networked Appliance with Dynamic Language
RTEMS example application I

#include <stdio.h>
#include <stdlib.h>

void *POSIX_Init (void *argument)
{
    printf("Hello, world");
    exit(0);
}

/* configuration information */
#include <bsp.h>

#define CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_POSIX_INIT_THREAD_TABLE
#define CONFIGURE_MAXIMUM_POSIX_THREADS 1
#define CONFIGURE_INIT
#include <rtems/confdefs.h>
Building RTEMS systems

Building a complete system

1. Download sources (links: see published slides)
2. Define target environment
3. Define build environment details

Technical Details

wget http://url/of/component.tar.bz2
Building RTEMS systems

Building a complete *system*

1. Download sources (links: see published slides)
2. Define target environment
3. Define build environment details

Technical Details

```bash
export ARCH=arm
export BSP=raspberrypi
```
Building RTEMS systems

Building a complete system

1. Download sources (links: see published slides)
2. Define target environment
3. Define build environment details

Technical Details

export ARCH=arm
export BSP=stm32f4
Building RTEMS systems

Building a complete *system*

1. Download sources (links: see published slides)
2. Define target environment
3. Define build environment details

Technical Details

```bash
export ARCH=i386
export BSP=pc386
```
Building RTEMS systems

Building a complete system

1. Download sources (links: see published slides)
2. Define target environment
3. Define build environment details

Technical Details

export TARGET=${ARCH}-rtems4.11
export TOOLDIR=${HOME}/rtems-bin
export JOBS=8
export PATH=${TOOLDIR}/bin:${PATH}
export RTEMS_MAKEFILE_PATH=
    ${TOOLDIR}/rtems/bsps/4.11/${ARCH}-rtems4.11/${BSP}
export RTEMS_ROOT=${TOOLDIR}/rtems/bsps/4.11/share/rtems4.11/
## Building RTEMS systems

### Building a complete system

1. Build binutils
2. Build the initial C compiler/standard C library/final C compiler
3. Build the RTEMS kernel

### Technical Details

```
mkdir binutils-build; cd binutils-build
../binutils-version/configure --target=${TARGET} \ 
   --prefix=${TOOLDIR}
make -j${JOBS} && make install
```
### Building RTEMS systems

#### Building a complete system

1. Build binutils
2. Build the initial C compiler/standard C library/final C compiler
3. Build the RTEMS kernel

#### Technical Details

```bash
mkdir gcc-build; cd gcc-build
../gcc-4.9.1/configure --target=$TARGET --without-headers
--with-gnu-as --with-newlib
--enable-threads --prefix=$TOOLDIR
--enable-languages="c,c++"
make all-gcc -j${JOBS} && make install-gcc
```
Building RTEMS systems

Building a complete system

1. Build binutils
2. Build the initial C compiler/standard C library/final C compiler
3. Build the RTEMS kernel

Technical Details

mkdir newlib-build; cd newlib-build
../newlib/configure --target=${TARGET} --prefix=${TOOLDIR}
make -j${JOBS} && make install
Building RTEMS systems

Building a complete *system*

1. Build binutils
2. Build the initial C compiler/standard C library/final C compiler
3. Build the RTEMS kernel

Technical Details

```
cd gccbuild
./gcc-4.9.1/configure --target=${TARGET} --with-gnu-as \ 
   --with-newlib --enable-threads \ 
   --enable-languages="c,c++" \ 
   --prefix=${TOOLDIR}
make -j${JOBS} && make install
```
Building RTEMS systems

Building a complete system

1. Build binutils
2. Build the initial C compiler/standard C library/final C compiler
3. Build the RTEMS kernel

Technical Details

cd rtems; ./bootstrap
cd ..; mkdir rtems-build; cd rtems-build
./rtems/configure --target=${ARCH}-rtems4.11
  --enable-rtemsbsp=${BSP} \
  --enable-networking --enable-posix \n  --prefix=${TOOLDIR}/rtems/bsps/4.11
make && make install
Building the payload application

- Collect application sources (libraries: separate build, of course)
- Makefile templates: `rtems/make/Template`
- Build process delivers binaries in `o-optimized/`
  - `file.exe`: ELF executable. Symbol information, sections etc.
  - `file.ralf`: *RTEMS Application Loadable File*. “Core dump” of the binary. \(\approx\) `objdump -O binary` – execute on *raw* hardware (debugger, flash tool, bootloader)
Running pc386 inside qemu

System in emulated machine

```bash
qemu-system-i386 -no-reboot -serial stdio -monitor null \  -nographic -m 2 \  -append "--console=com1" \  -s -kernel file.exe
```
Running pc386 inside qemu

System in emulated machine + debugging

```sh
quemumentiy386 --no-reboot --serial stdio --monitor null \
    --nographic --m 2 \
    --append "--console=com1" \
    --s --kernel file.exe
```

- `gdb file.elf`
- `(gdb) target remote localhost:1234`
- ...as easy as debugging a simple Linux userland application
Running pc386 inside qemu

<table>
<thead>
<tr>
<th>System in emulated machine + debugging + networking</th>
</tr>
</thead>
</table>
| `qemu-system-i386 -no-reboot -serial stdio -monitor null \
  -nographic -m 4 \ 
  -append "--console=com1 --ne2k-irq=9" \ 
  -device ne2k.isa,netdev=usernet \ 
  -netdev user,id=usernet \ 
  -redir tcp:24742::24742 \ 
  -s -kernel file.exe` |
Networked appliance with dynamic language

Requirements

- Payload: No deeply embedded experts required
  - Port lua to device (essentially: adapt Makefile)
  - Run standard C applications (RT etc.) in parallel
- Standard networking (configuration option!)
- Linux-like interactive development
- Works well with \( \approx 0.5 \text{ MiB of RAM} \)
- See link in published slides
Thanks for your interest!