Putting Linux into Context
Towards a reproducible example system with Linux, Zephyr & Xen

Philipp Ahmann, Robert Bosch GmbH

With work from ELISA contributors:
Alessandro Carminati (Red Hat), Maurizio Papini (Red Hat),
Shefali Sharma, Shuah Khan (LF), Stefano Stabelllini (Xilinx/AMD),
Sudip Mukherjee (Codethink), Thomas Mittelstädt (Robert Bosch GmbH),
Product Manager for Embedded Open Source

Chair of the Technical Steering Committee
Lead of the Systems Working Group

Member of the Inaugural Advisory Board

OSS enthusiast and promoter
ENABLING LINUX IN SAFETY APPLICATIONS
ENABLING LINUX IN SAFETY APPLICATIONS

SAFETY ... don't mix it up with SECURITY
“The mission of the project is to define and maintain a common set of elements, processes and tools that can be incorporated into Linux-based, safety-critical systems amenable to safety certification.”

from the technical charter
The mission of the project is to define and maintain a common set of elements, processes and tools that can be incorporated into Linux-based, safety-critical systems amenable to safety certification.
Working Groups (WGs) - Verticals

Aerospace
- Boeing

Automotive
- Bosch

Medical Devices
- Linux Foundation

OpenAPS elements
1. Continuous glucose monitor
2. Computer
3. Battery
4. Radio stick
5. Insulin pump

Dana Lewis’ OpenAPS project: https://youtu.be/kgu-AYSnyZ8
“Linux differs from a ‘traditional’ safety critical OS,… but both face challenges in modern complex system setups.”
Clash of worlds
(or what is often considered unsafe by safety experts):

- Memory management
- Dynamic memory allocation
- Caches
- Interrupt handling
- Real time scheduling
- …
Tools + Documentation help to understand complex systems better

- STPA
- strace and cscope for workload tracing
- ks-nav (graphical representation kernel sources)
- real-time analysis
STPA HANDBOOK

NANCY G. LEVESON
JOHN P. THOMAS

MARCH 2018

This handbook is intended for those interested in using STPA on real systems. It is not meant to introduce the theoretical foundation, which is described elsewhere. Here our goal is to provide direction for those starting out with STPA on a real project or to supplement other materials in a class teaching STPA.

STPA – Basics

- Relatively new hazard analysis technique
- Very complex systems can be analyzed
- Iterative towards detailed design decisions
- Includes software and human operators
- Provides documentation of system functionality
- Can be easily integrated into (model-based) system engineering process
STPA – Basics

4 key elements

- **Controller** sends
- **Control Action(s)** to a
- **Controlled Process** which provides
- **Feedback** to a controller

A controlled process can be a controller.

Q: What can be unsafe control actions?
STPA – In action (example for OpenAPS)
STPA – In action (example for OpenAPS)
Deeper level of analysis required workload tracing

Main tools used:
- strace
- cscope

Extract information:
- System Call
- Frequency of call
- Involved Subsystem
- System Call Entry Point

<table>
<thead>
<tr>
<th>System Call</th>
<th>Frequency</th>
<th>Linux Subsystem</th>
<th>System Call Entry Point (API)</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>3</td>
<td>Filesystem</td>
<td>sys_read()</td>
</tr>
<tr>
<td>write</td>
<td>11</td>
<td>Filesystem</td>
<td>sys_write()</td>
</tr>
<tr>
<td>close</td>
<td>41</td>
<td>Filesystem</td>
<td>sys_close()</td>
</tr>
<tr>
<td>stat</td>
<td>24</td>
<td>Filesystem</td>
<td>sys_stat()</td>
</tr>
<tr>
<td>fstat</td>
<td>2</td>
<td>Filesystem</td>
<td>sys_fstat()</td>
</tr>
<tr>
<td>pread64</td>
<td>6</td>
<td>Filesystem</td>
<td>sys_pread64()</td>
</tr>
<tr>
<td>access</td>
<td>1</td>
<td>Filesystem</td>
<td>sys_access()</td>
</tr>
<tr>
<td>pipe</td>
<td>1</td>
<td>Filesystem</td>
<td>sys_pipe()</td>
</tr>
<tr>
<td>dup2</td>
<td>24</td>
<td>Filesystem</td>
<td>sys_dup2()</td>
</tr>
<tr>
<td>execve</td>
<td>1</td>
<td>Filesystem</td>
<td>sys_execve()</td>
</tr>
<tr>
<td>fork</td>
<td>26</td>
<td>Filesystem</td>
<td>sys_fork()</td>
</tr>
<tr>
<td>openat</td>
<td>14</td>
<td>Filesystem</td>
<td>sys_openat()</td>
</tr>
<tr>
<td>rt_sigaction</td>
<td>7</td>
<td>Signal</td>
<td>sys_rt_sigaction()</td>
</tr>
<tr>
<td>rt_sigreturn</td>
<td>38</td>
<td>Signal</td>
<td>sys_rt_sigreturn()</td>
</tr>
<tr>
<td>clone</td>
<td>38</td>
<td>Process Mgmt.</td>
<td>sys_clone()</td>
</tr>
<tr>
<td>wait4</td>
<td>44</td>
<td>Time</td>
<td>sys_wait4()</td>
</tr>
<tr>
<td>mmap</td>
<td>7</td>
<td>Memory Mgmt.</td>
<td>sys_mmap()</td>
</tr>
<tr>
<td>mprotect</td>
<td>3</td>
<td>Memory Mgmt.</td>
<td>sys_mprotect()</td>
</tr>
<tr>
<td>munmap</td>
<td>1</td>
<td>Memory Mgmt.</td>
<td>sys_munmap()</td>
</tr>
<tr>
<td>brk</td>
<td>3</td>
<td>Memory Mgmt.</td>
<td>sys_brk()</td>
</tr>
<tr>
<td>getpid</td>
<td>1</td>
<td>Process Mgmt.</td>
<td>sys_getpid()</td>
</tr>
<tr>
<td>getuid</td>
<td>1</td>
<td>Process Mgmt.</td>
<td>sys_getuid()</td>
</tr>
<tr>
<td>getgid</td>
<td>1</td>
<td>Process Mgmt.</td>
<td>sys_getgid()</td>
</tr>
<tr>
<td>geteuid</td>
<td>2</td>
<td>Process Mgmt.</td>
<td>sys_geteuid()</td>
</tr>
<tr>
<td>getegid</td>
<td>1</td>
<td>Process Mgmt.</td>
<td>sys_getegid()</td>
</tr>
<tr>
<td>getppid</td>
<td>1</td>
<td>Process Mgmt.</td>
<td>sys_getppid()</td>
</tr>
<tr>
<td>arch_profile</td>
<td>2</td>
<td>Process Mgmt.</td>
<td>sys_arch_profile()</td>
</tr>
</tbody>
</table>
Workload tracing documentation is mainlined.

- Understanding system resource necessary to build and run a workload is important
- Linux tracing and strace can be used to discover the system resource in use by a workload
- Additional tools (like perf, stress-ng, paxtest) can help to analyze performance and security of the OS
- Credits to Shuah Khan & Shefali Sharma for bringing it mainline
  - /Documentation/admin-guide/workload-tracing.rst
Dynamic tracing is supported by Static Analysis Navigator (ks-nav tool)

- Supports the analysis on code/kernel level
- Graphical representation of source code
- Provides insights about the Kernel construction
- Is there a good place upstream?
- Credits to Alessandro Carminati & Maurizio Papini (both Red Hat)

https://github.com/elisa-tech/ks-nav
Use case centric vs. common/generic use of Linux (the core)

- Use cases bring a different point of view and set context, but deal with similar problems

- Requires deep dives

- Deep dive from the past were e.g.:
  - PREEMPT_RT and how to not break it.
  - Real-time Linux analysis tool set.

- All results should end up in upstream documentation

- Helps system integrators to build safe software and improve Kernel quality

Important topics for potential deep-dives:

- Synchronization / timing
- Interrupt and exception management
- Resource access management
- Dynamic memory allocation
- Inter process communication & inter processor communication
- System initialization
- Kernel configuration & trimming
Possible next documentation: Admin guide for PREEMPT_RT

- PREEMPT_RT mostly upstream, but documentation on use can still be improved.
  - Nothing available so far in the admin-guide kernel documentation.

- Shuah Khan and Elana Copperman presented first results.
  - [RT Linux in Safety Critical Systems: the potential and the challenges](#)

- The Linux Features for Safety-Critical Systems (LFSCS) within ELISA is looking for support by PREEMPT_RT users/experts to bring this forward!
Interaction with other communities (outside of ELISA)

- Open source projects focusing on safety-critical analysis
  ![Xen Project](image1)
  ![Zephyr](image2)

- Open source projects with safety-critical relevance and comparable system architecture considerations
  ![Automotive Grade Linux](image3)
  ![SOAFeE](image4)
  ![SDV](image5)

- Further community interactions
  ![yocto](image6)
  ![SPDX](image7)
  ![Linaro](image8)

"If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas."

— George Bernard Shaw
“When it comes to prototyping systems, the existing guidelines are limited; reproducing demos is hard and time consuming.”
Static Partitioning with Xen, LinuxRT, and Zephyr: a concrete end-to-end example

Stefano Stabellini
Embedded Linux Conference 2022
Content of the Xen end-to-end example

- Build a reference system with default tooling
  - Xen, Linux kernel & rootfs and Zephyr
  - Use ImageBuilder for bootable configuration
  - Xen Device Tree examples

- Give guidance on features (“steps”)
  - Static partitioning
  - Device Assignment
  - Cache Coloring
  - Shared Memory and Event Channels
  - PV drivers
“A product will run on real hardware.”
Challenges

- New hardware
- Community support
- OS distro
- Tools & CI
- Proprietary drivers
- Images
- SBOM
Big thanks to…

Thomas Mittelstädt

- Robert Bosch GmbH
- Brings 30 years of experience at multiple operating systems and at build & integration systems. He provides trainings, documentation and technical support to various kind of Bosch users.

Sudip Mukherjee

- Codethink
- He has been a mainline kernel contributor since 2014. Sudip is also a Debian Developer and has worked in multiple automotive projects for Codethink’s clients.
Major challenges during setup of XEN systems

- Select target board with
  - Hardware support for XEN, especially SMMU controller
  - XEN community support
  - Documentation for build and setup
  - Licenses compliant to OSS project
- Setup of Yocto build environment
  - Amount of computer resources
  - Network and Host dependencies
- Finding valid descriptions
- Build image parts based on descriptions
- Finding community support at occurring build problems
- Understanding XEN setup and structure
### Evaluated targets

<table>
<thead>
<tr>
<th>Renesas RCAR 3.0 family</th>
<th>Xilinx Zynqmp and Ultrascale family</th>
</tr>
</thead>
<tbody>
<tr>
<td>(link to Wiki of eLinux)</td>
<td>(link to product page)</td>
</tr>
</tbody>
</table>

- **Renesas RCAR 3.0 family**
  - XEN hardware support
  - Functional XEN systems (also graphic)
  - Proprietary licenses for essential parts like graphic
  - Not available at standard market

- **Xilinx Zynqmp and Ultrascale family**
  - XEN hardware support
  - Functional XEN systems
  - Good documentation and open source support of Xilinx
  - Graphic at Zcu102 atm not able to be handled by XEN
  - Zcu102 well supported, but additional complexity due to FPGA programming
Evaluated targets – cont.

- **Qemu systems for Xilinx**
  (link with some hints for setup with XEN at Xilinx boards)
  - XEN support
  - Functional XEN systems
  - no hardware needed
  - Only for development, not for hardware related demo cases

- **Raspberry Pi systems**
  - Hardware support not sufficient for security requirements of XEN

- **NXP i.mx8 systems**
  - Good hardware support for hypervisor like XEN
  - Less community support
Used hardware

- Board ZCU102 ([link to description](#))
  - Reference manual ([link](#))
  - SD card 16GB for boot loader
  - USB Stick 16GB for demonstrator setup
  - USB-Ethernet-Adapter (DLink)

- Environment for setup
  - Local DHCP server (VM with system networkd)
  - Putty for serial console
  - USB Keyboard (for TTY console)
  - HDMI screen
Overview of the XEN example system

- **Hardware**
  - ZCU 102
  - Demo setups
  - SD Card
  - USB Stick
  - USB ethernet adapter
  - Local Ethernet
  - Putty (serial console)

- **Build sources**
  - Xilinx Yocto 2022.2 for XEN
  - BSP v2022.2 Zcu 102
  - Apertis Build
  - RCAR demonstrator (meta-xt-prod-devel-rcar)
  - Building Xen Hypervisor with PetaLinux 2022.2

- **Local tools**

- **SW parts**

- **Hardware**
  - Local network with DHCP
  - Bootloader
  - XEN system
  - Simple Petalinux
  - Apertis (Debian)
  - Zephyr
  - Demo setups
External parts of system images

- Xen Hypervisor ([link](#) for build description)
  - Image, ramdisk, device tree
  - Boot.bin
- Petalinux ([link](#) for binaries from "BSP")
  - Image, ramdisk, (device tree: not used for XEN)
- Zephyr (atm got from demo for Renesas RCAR, [link](#) for build description)
  - Image
  - Configuration file for XEN
- XEN configuration files (created on description at [link](#))
- Apertis (Debian based, specific image, but general build instructions at [link](#))
  - Image, ramdisk, (device tree: not used for XEN)
- XEN image builder ([link](#) for download and usage)
CI enablement: https://gitlab.com/elisa-tech/systems-wg-ci

Build & packages

Runs daily

Artifacts download of recent images
meta-elisa: Various starting points provided

- Plain and native from source
  https://github.com/elisa-tech/meta-elisa

- Using docker container
  https://github.com/elisa-tech/wg-automotive/tree/master/Docker_container

- With cached build using SSTATE
  modify "conf/local.conf" after the "source" command before the "bitbake" command

- Download binaries directly from build server
  https://gitlab.com/elisa-tech/meta-elisa-ci
Pipeline dependencies

Full description in the blog
https://elisa.tech/blog/2023/04/05/elisa-ci-enablement-automation-tools-for-easier-collaboration/
Limitations of the current implementation.

Pipeline flow

Sources → Docker file → Docker image → Build Image → Generated Image

Features
Availability
Costs

Pure Linux system “meta-elisa”:

Work in Progress - License: CC-BY-4.0
From hardware to qemu (again?!)  

- QEMU increases availability, lowers costs, but misses some features (like HW interfaces)

- Uncovered topics: System diversity, hardware prototyping, virtual GPU performance, “real µC” involvement
Open questions…

● What is a good hardware to extend the PoC scope?
● Are there further existing examples where open source, security, safety and compliance come best together?
● Which alternative real-time operating system and virtualization should be incorporated?
Benefits provided by the ELISA project

- **Provide system engineering and safety competencies**
  - Workload tracing upstream in kernel mainline
  - Tools for kernel analysis in ELISA github repo
- **Provide a start into safety critical system creation**
  - Hosting seminars to educate system creators/integrators
  - Create a knowledge base around Linux in safety critical environments
  - Provide a working example system for easy start into system creation
- **Want to get much more of this system experience upstream**
Where do we want to do, where do we need help, … (path to grow)

<table>
<thead>
<tr>
<th>Available inputs (to work on task)</th>
<th>Completed example “workload tracing”</th>
<th>Ongoing activity “PREEMPT_RT guide”</th>
<th>Planned work “reproducible example system”</th>
<th>Future activities “critical kernel core components docu”</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>RT and RT tooling deep dives</td>
<td>Interaction with Xen and Zephyr community</td>
<td>none yet!</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target activity</th>
<th>Tools to get a better understanding</th>
<th>Provide a guide on how to work with real time inside the kernel</th>
<th>Help people understand to create complex Linux based systems</th>
<th>Identification of mission critical kernel parts supporting safety of products</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creating a SBOM and porting to yocto</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Where we could need help</th>
<th>Help in porting the Raspbian demo to Yocto incl. kernel build</th>
<th>Contribution from PREEMPT_RT users to create the document</th>
<th>Proposals for well supported community hardware with HV support</th>
<th>Deep dives into special topics:  - memory, interrupts, …</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Help in porting the Raspbian demo to Yocto</td>
<td></td>
<td>Proposals for well supported community hardware with HV support</td>
<td>Deep dives into special topics:  - memory, interrupts, …</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Place for the results</th>
<th>Documentation in Kernel admin-guide yocto with SBOM for openAPS community</th>
<th>Documentation in Kernel admin-guide</th>
<th>ELISA project github</th>
<th>- Kernel patches  - Documentation in admin-guide - manpages</th>
</tr>
</thead>
</table>


Thank you!

Linux Plumbers Conference

Richmond, Virginia | November 13-15, 2023