The ELISA Project
Enabling Linux in Safety Applications
Philipp Ahmann, Robert Bosch GmbH
Embedded IoT
Linux@Bosch
whoami

- Technical business development manager for embedded open source - Robert Bosch GmbH
- Technical Steering Committee Chair & WG Lead - Linux Foundation’s ELISA project
- 15 years+ Linux user (and open source enthusiast)
- 10 years+ Linux in Automotive (Infotainment)

https://oss.bosch-cm.com/
Linux in Safety Critical Systems

Assessing whether a system is safe, requires understanding the system sufficiently.

Understand Linux within that system context and how Linux is used in that system.

Selecting Linux components and features that can be evaluated for safety.

Identifying gaps that exist where more work is needed to evaluate safety sufficiently.
Challenge: Linux in Safety-Critical Systems

The Linux kernel has:
- Large Development Ecosystem
- Security Capabilities
- Multi-Core Support
- Unmatched Hardware Support
- Many Linux Experts at all levels available

Traditional safety-critical OS has:
- Hard Real-time Capabilities
- Proven Safety-compliant Development Process
- …

Can these differences be tackled?
Understanding the Limits

The collaboration…

- *cannot engineer* your system to be safe.
- *cannot ensure* that you know how to apply the described process and methods.
- *cannot create* an out-of-tree Linux kernel for safety-critical applications. (Remember the continuous process improvement argument!)
- *cannot relieve* you from your responsibilities, legal obligations and liabilities.

But…

**ELISA provides a path forward and peers to collaborate with!**
“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.”

“The scope of the project includes software and documentation development under an OSI-approved license supporting the mission, including documentation, testing, integration and the creation of other artifacts that aid the development, deployment, operation or adoption of the project.”

from the technical charter
ELISA Working Groups - Deliverable grouping (based on mission)

- Elements / Software
- Processes
- Tools
- Documentation
Working groups (WGs)
Horizontal Working Groups

Safety Architecture
- Red Hat

Linux Features
- mobileye®/intel.

Tool investigation & Code Improvement
- Elektrobit
- Codethink

Open Source Engineering Process
- Codethink

Systems
- BOSCH
Vertical Use Cases

Aerospace

Automotive

Medical Devices

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

@DanaMLewis
ELISA Working Groups

- Automotive (use-case)
- Medical (use-case)
- Aerospace (use-case)

Open Source Engineering Process
- Systems
- Safety Architecture
- Linux Features for Safety-Critical Systems
- Tool Investigation and Code Improvement

ELISA Deliverables
ELISA Working Groups - Fit in an exemplary system

- **Linux Features, Architecture and Code Improvements** should be integrated into the reference system directly.

- **Tools** and **Engineering process** should serve the reproducible product creation.

- **Medical, Automotive, Aerospace** and future WG use cases should be able to strip down the reference system to their use case demands.
Interaction with other communities (outside of ELISA)

- Safety-critical open source projects
  - Xen Project
  - Zephyr

- OSS project with safety-critical relevance and comparable system architecture considerations
  - Automotive Grade Linux
  - SOAFEE
  - Eclipse Software Defined Vehicle

- Further interactions
  - Yocto Project
  - SPDX

“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
Artifacts & Activities
ELISA Working Groups - Deliverables

- Elements / Software
- Processes
- Tools
- Documentation

- STPA
- Codechecker
- Workload tracing
- Call-Tree

GitHub / Gdrive / Blog / Whitepaper

meta-elisa
S T P A
System - Theoretic Process Analysis
STPA – Basic Idea

- 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

Reading more about the methodology in the handbook:
STPA – Basic Idea

4 key elements

● Controller sends
● Control Action(s) to a
● Controlled Process which provides
● Feedback

Q: What can be unsafe control actions?
STPA – Example for OpenAPS
STPA – Example for OpenAPS

Level 2

Algorithm developer

- E-mail notifications
- Forum
- Questions

Infrastructure developer

- E-mail notifications
- Forum
- Questions

Human Operator

- Install or update dependencies
- Specify and set up algorithm and loop
- Start OpenAPS
- Report operating data
- Stop OpenAPS

OpenAPS System

Raspberry Pi OS

- Command to set up algorithm
- Request to communicate with other devices
- Receive data from other devices
- Stop OpenAPS

OpenAPS Toolkit

- Set up algorithm and loop
- Provide data from other devices
- Stop algorithm

Algorithm

- Algorithm output

NightScout

Insulin Pump
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>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>3</td>
</tr>
<tr>
<td>write</td>
<td>11</td>
</tr>
<tr>
<td>close</td>
<td>41</td>
</tr>
<tr>
<td>stat</td>
<td>24</td>
</tr>
<tr>
<td>fstat</td>
<td>2</td>
</tr>
<tr>
<td>pread64</td>
<td>6</td>
</tr>
<tr>
<td>access</td>
<td>1</td>
</tr>
<tr>
<td>pipe</td>
<td>1</td>
</tr>
<tr>
<td>dup2</td>
<td>24</td>
</tr>
<tr>
<td>execve</td>
<td>1</td>
</tr>
<tr>
<td>fdctl</td>
<td>26</td>
</tr>
<tr>
<td>openat</td>
<td>14</td>
</tr>
<tr>
<td>rt_sigaction</td>
<td>7</td>
</tr>
<tr>
<td>rt_sigreturn</td>
<td>38</td>
</tr>
<tr>
<td>clone</td>
<td>38</td>
</tr>
<tr>
<td>waiov</td>
<td>44</td>
</tr>
<tr>
<td>mmapping</td>
<td>7</td>
</tr>
<tr>
<td>mprotect</td>
<td>3</td>
</tr>
<tr>
<td>munmap</td>
<td>1</td>
</tr>
<tr>
<td>brk</td>
<td>3</td>
</tr>
<tr>
<td>getpid</td>
<td>1</td>
</tr>
<tr>
<td>getuid</td>
<td>1</td>
</tr>
<tr>
<td>getgids</td>
<td>1</td>
</tr>
<tr>
<td>geteuid</td>
<td>2</td>
</tr>
<tr>
<td>getpgid</td>
<td>1</td>
</tr>
<tr>
<td>arch_prctl</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linux Subsystem</th>
<th>System Call Entry Point (API)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filesystem</td>
<td>sys_read()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_write()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_close()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_stat()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_fstat()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_pread64()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_access()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_pipe()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_dup2()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_execve()</td>
</tr>
<tr>
<td>Filesystem</td>
<td>sys_fdctl()</td>
</tr>
<tr>
<td>Signal</td>
<td>sys_rt_sigaction()</td>
</tr>
<tr>
<td>Signal</td>
<td>sys_rt_sigreturn()</td>
</tr>
<tr>
<td>Process Mgmt.</td>
<td>sys_clone()</td>
</tr>
<tr>
<td>Time</td>
<td>sys_wait4()</td>
</tr>
<tr>
<td>Memory Mgmt.</td>
<td>sys_mmap()</td>
</tr>
<tr>
<td>Memory Mgmt.</td>
<td>sys_mprotect()</td>
</tr>
<tr>
<td>Memory Mgmt.</td>
<td>sys_munmap()</td>
</tr>
<tr>
<td>Memory Mgmt.</td>
<td>sys_brk()</td>
</tr>
<tr>
<td>Process Mgmt.</td>
<td>sys_getpid()</td>
</tr>
<tr>
<td>Process Mgmt.</td>
<td>sys_getuid()</td>
</tr>
<tr>
<td>Process Mgmt.</td>
<td>sys_geteuid()</td>
</tr>
<tr>
<td>Process Mgmt.</td>
<td>sys_getpgid()</td>
</tr>
<tr>
<td>Process Mgmt.</td>
<td>sys_arch_prctl()</td>
</tr>
</tbody>
</table>

https://github.com/elisa-tech/ELISA-White-Papers/blob/master/Processes/Discovering_Linux_kernel_subsystems_used_by_a_workload.md
Call Tree Tool

- Supports the analysis on code/kernel level
- Graphical representation of source code
- Provides insights about the Kernel construction

https://github.com/elisa-tech/Safety_Architecture_WG/tree/main/call_tree_tool
- Reuse Automotive Grade Linux (AGL) instrument cluster demo
- QT based running on qemu
- **DANGER** added to illustrate tell tale safety monitoring

[https://github.com/elisa-tech/meta-elisa](https://github.com/elisa-tech/meta-elisa)
External Watchdog

- The challenge-response watchdog serves as the “safety net” for the safety-critical workload
- The concept is widely used in Automotive and other industrial applications
- It can be used as an iterative approach to assign more safety-critical functionality to Linux

With a proper system design the watchdog will never need to trigger the “safe state”.

Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units

Work in Progress - License: CC-BY-4.0
Use case recording on the event side
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
Next steps

- **STPA**
  - Align implementation and design using workload tracing and Linux analysis

- **Call Tree**
  - Extend call tree with additional tools. [PR](#) for a tool called “ks-nav” is already open

- **Workload tracing**
  - Use the workload tracing from medical devices for the automotive use case
  - Add other tracing tools like SystemTap

- **meta-elisa**
  - Bring the qemu showcase on real hardware using the work of the systems WG
  - Improve the monitoring and checking apps

And more: System SBOM generation, Kernel configuration & image size trim down, RT docu review, cluster demo on complex system architecture, more docu…
Conclusion
Conclusion

- Challenges
- Goals and technical strategy
- Different work groups & their interaction

- Contributions & Deliverables
- Used methodologies and tools
- Current status & what comes next
Questions? info@elisa.tech

Advancing Open Source Safety-Critical Systems

The mission of the Enabling Linux in Safety Applications (ELISA) project is to make it easier for companies to build and certify Linux-based safety-critical applications — systems whose failure could result in loss of human life, significant property damage or environmental damage. ELISA members are working together to define and maintain a common set of tools and processes that can help companies demonstrate that a specific Linux-based system meets the necessary safety requirements for certification.
Getting involved...

● Join main technical and weekly calls of interest:
  ○ Main Technical List: devel@lists.elisa.tech
  ○ Safety Architecture Workgroup: safety-architecture@lists.elisa.tech
  ○ Open-Source Engineering Process WG: osep@lists.elisa.tech
  ○ Linux Features for Safety-Critical Systems WG: linux-features@lists.elisa.tech
  ○ Medical Devices Workgroup: medical-devices@lists.elisa.tech
  ○ Automotive Workgroup: automotive@lists.elisa.tech
  ○ Systems Workgroup: systems@lists.elisa.tech
  ○ (Full list at: https://lists.elisa.tech/g/linux-features/subgroups)

● Contribute content, review materials and add your comments to:
  ○ ELISA Technical Community Google Drive: https://drive.google.com/open?id=1Y6Uwqt5VEDEZjIpRe0CBlbdtXpDwlg
  ○ ELISA github repository: https://github.com/elisa-tech/workgroups
  ○ ELISA github issue tracker: https://github.com/elisa-tech/workgroups/issues
Advancing Open Source Safety-Critical Systems

Elements • Processes • Tools • Documentation