ELISA

Enabling **Linux** in **Safety** Applications

WORKSHOP

How far do we go at the hardware level?

Olivier Charrier (Wind River), Alessandro Carminati (Red Hat)

May 7-9, 2025

Agenda

- Introduction: the Software Integrator view
- Looking into the Linux Kernel
- Hardware Abstraction

ELISA Enabling Linux in Safety Applications

WORKSHOP

Introduction:
The Software Integrator View

ELISA Enabling Linux in Safety Applications

WORKSHOP

Integration at End System Level

Business World

End User (OEM)

The End User is cascading his Safety Requirement and Safety Concept (Design) to his supplier.

Request for Proposal (RFP)

The End User is assessing the suitability of the existing equipment and adapt his Safety Concept (Design) accordingly.

Safety Manual of an existing equipment

Equipment Supplier

ELISA Enabling Linux in Safety Applications

WORKSHOP

Integration at Equipment Level

Business World

Equipment Supplier

The Equipment Supplier is designing his system, including a Safety Concept then allocating Safety Requirements to HW and SW

Linux Integration: HW/SW + SW/SW

Linux (OSS Community)

OSS World

ELISA
Enabling Linux in Safety Applications

WORKSHOP

Linux Integration – the Who

Who is performing the Linux Integration?

- The Equipment Supplier building the complete Software Stack of the Equipment
- A Hardware Supplier performing a pre-integration HW/SW of Linux + BSP
- A Linux distribution maker can propose re-usable Linux based solution for Safety
- A Solution Maker providing a pre-canned HW+SW Computing Platform with a Safety Manual

A close collaboration between all stakeholders is mandatory, early in the project, to globally define the integration work and to achieve safety.

The "Who" is adding his liability to the usage of OSS.

Linux Integration – the What

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

What does the Linux Integrator have to do about Linux?

The Equipment Supplier will always have to:

- Assess the suitability of Linux against the System Functional and Safety requirements cascaded by the End User
- Identify the Safety Requirements to be allocated to Linux
- Fill the potential gaps to meet Safety Certification Objectives given by Safety Standards and/or Safety Regulatory Agencies

This work can be split with a Hardware Supplier, Linux Distribution Supplier, or Solution Maker, but always within a well-defined integration framework allowing the Equipment Supplier to get his system approved for Safety.

Linux Integration – the When

When will the equipment using Linux by ready to be sold and deployed?

- When the "What" is complete in the context of following existing standard and regulation.
- When the "What" is complete in the context of a NEW approach for safety + the time required to convince the Safety Community and the Safety Regulator (if applicable) that such NEW approach brings confidence that safety is achieved.

ELISA Enabling Linux in Safety Applications

WORKSHOP

Linux Integration – the How

How can the Linux Integrator guarantee that the Safety Requirements are fulfilled with Linux? … with the help of ELISA Enabling Linux in Safety Applications

1. To demonstrate trustfulness of the required Linux capabilities (the bigger Linux is, the more work to be done)
2. Implement a preventing monitoring around Linux to prevent violation of the safe behavior before they occur
3. Implement an error monitoring around Linux to detect violation of the safe behavior

For all the above, a certain level of information will be required on potential failure mode of the considered Linux Components / sub-systems.

The level of information available can have an impact on the system **Availability**.

Looking into the Linux Kernel

Kernel Role

Provides API to userspace to interact with resources

- Software resources
- Hardware resources

Kernel configuration

- Kernel is monolithic and depends on its configuration
- Stub function can become full fledged complex depending on configuration
- Minimal configuration helps investigate the essentials.

Minimal expectation from the kernel

The logical step ahead is to ask what is the minimal expectation from kernel

- Elisa community has made a formal investigation over the topic
- Investigation shows that minimal programs are still considered the hardware abstracted

Investigation - Objectives & Scope

- Objective: Define the minimum required Linux kernel features for safe execution.
- Scope:
  - Identify essential kernel functions & subsystems.
  - Analyze dependencies and unexpected function calls.
  - Recommend a minimal kernel configuration.
  - Produce a set of feature (subsystems) used by a minimal application and hence any application.

Minimum kernel requirements - Methodology

- **Approach:**
  - Use `ftrace` to trace kernel interactions. (indirect measurement leave some holes)
  - Hole filled comparing ks-nav static views and ftrace produced graphs.
  - Minimize system userspace via Buildroot.
  - To minimize the kernel dependencies, the userspace filesystem is passed as initramfs
- **Data Collection:**
  - `min.c`: Smallest viable C program for kernel interaction. Static linked without crt.o
  - `ftrace_it.c`: Tracing setup to capture function calls.
- **Challenges:** Noise from external processes, missing traceable functions.

Original Linux Feature WG [investigation]

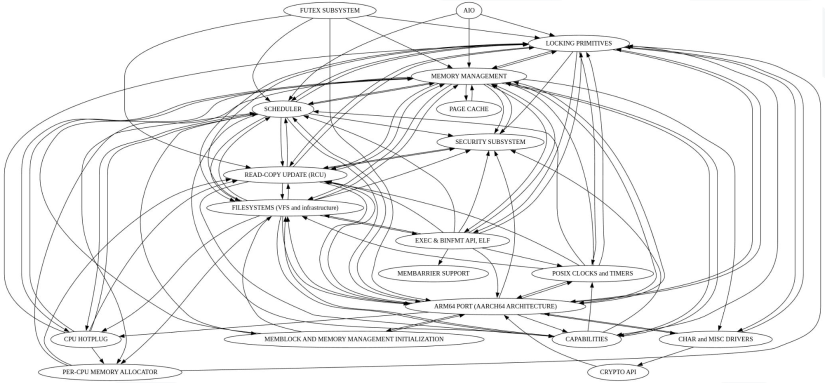
Minimum kernel requirements - Key Findings

- Minimal required kernel subsystems:
  - `execve` syscall: Core to process execution.
  - Memory management: Address allocation & paging.
  - Tracing confirms essential interactions.
- Unexpected discovery:
  - Crypto API (`chacha_block_generic`) unexpectedly required.
  - Because of the padding, the stack can be larger than a single page as it was reasonable to expect.

Minimum kernel requirements - Conclusion

Conclusion:

- List of the subsystem needed by `min.c`



AIO	CRYPTO API	LOCKING PRIMITIVES	PER-CPU MEMORY ALLOCATOR	CHAR DRIVERS
ARM64 PORT	EXEC & BINFMT API	MEMBARRIER SUPPORT	POSIX CLOCKS and TIMERS	SLAB ALLOCATOR
CAPABILITIES	FILESYSTEMS	MEMORY MANAGEMENT	RANDOM NUMBER DRIVER	
SECURITY	FUTEX	MMU AND TLB	READ-COPY UPDATE	
CPU HOTPLUG	GENERIC INCLUDE	PAGE CACHE	SCHEDULER	

Hardware enablement

- Applications rely on kernel abstractions to access hardware.
- The community cannot be responsible for hardware-specific code.
- Hardware abstraction layers may fall within community scope.
- Safety responsibility for drivers and silicon should rest with vendors.

Hardware Abstraction

ELISA
Enabling Linux in Safety Applications

WORKSHOP

Linux Integration – the Hardware Support

What do we do within ELISA for the Hardware Support?

There are 2 main kinds of Hardware Support layer:

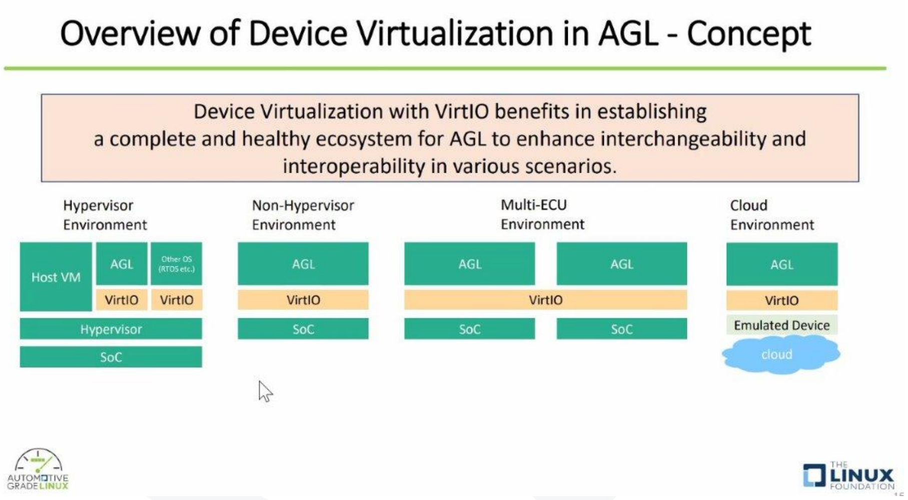
- The Support for Linux Kernel Capabilities –Tightly coupled with the Kernel Needs (ARM64 PORT)
  This layer is required to evaluate the trustfulness of associated capabilities (MMU, etc.)
  The scope of this part needs also to be discussed on what is included and what shall not (i.e. define a usage domain)

- The Hardware Resources only used by Applications, i.e. defined by the cascaded System Requirements.

Hardware Abstraction

Potential Abstraction Layers:

- C-Std Lib File Stream (fopen, fclose, fread, etc.) - buffered
- Block Device (bio) for File System
- POSIX I/O (read, write, ioctl, etc.) - unbuffered
- mmap()
- sysfs
- Network Device
- Virtio



Linux Integration – the Hardware Support

From an ELISA perspective:

- Shall we limit the User level APIs to access hardware resources (i.e. POSIX I/O and leave the C-Std Lib File Stream to the Application Developer)?
- Shall we drive the discussion with Silicon/Hardware vendors and Application Developers to determine which Abstraction Layer to analyze and provide materials for? A couple? An ordered wish list?
- Shall we leave the device drivers to Silicon and Hardware Suppliers?
    - Using guidance from ELISA to develop (Best Practices Standard), analyze and document
- Shall we gather all this ELISA work into a "Linux Integration Guide for Safety Critical Systems"

Thoughts ?

ELISA
Enabling Linux in Safety Applications

WORKSHOP

Licensing of Workshop Results

All work created during the workshop is licensed under Creative Commons Attribution 4.0 International (CC-BY-4.0) [https://creativecommons.org/licenses/by/4.0/] by default, or under another suitable open-source license, e.g., GPL-2.0 for kernel code contributions.

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.