embeddedworld
Exhibition&Conference

An approach to a maintainable OS-solution based on embedded Linux, safe and secure

and the fundamental role of proper lifecycle management.

Michael Armbruster, Thomas Brinker, Michel von Czettritz, Heike Jordan
safe and secure embedded Linux Solutions
emlix GmbH
37073 Göttingen, Germany
solutions@emlix.com

***Abstract*—The innovative embedded OS-solution described within this paper has been motivated by customers asking for a "safe" embedded Linux. "safe" is understood as "functionally safe" acc. to an established standard, specifically ISO26262 or IEC61508. The innovation does not only implement its property on Functional Safety, but also on Cybersecurity which is needed by further guidelines, regulations and standards (e.g. "Cyber Resilience Act" with guidelines such as TR-03183 or IEC62443). The field of tension in between Functional Safety, Cybersecurity and long-term maintenance is explained. As a second field of tension, the impact from Linux' architectural nature and design practices on the attempt to make it safe is broken down. However, the innovative approach derived from requirements and challenges follows a different strategy. Rather than making Linux safe, the OS-solution ensures a dependable data-space for safety-related applications up to SIL 2/ ASIL B. This makes this solution suitable for almost any regulated industry, including automotive. It is based on a decoupling of the lifecycle of the Linux kernel from that of other software elements. By virtue of this, its maintenance and patching can be done with minimal effort while maintaining its property on Cybersecurity. The technological concept is based on a "supervisor" software layer that detects and prevents events that are able to adversely effect the dependability of the data-space for safety-related applications. The supervision of the Linux kernel is made possible thanks a hypervisor that leverages functionalities provided by the hardware platform. A minimum viable product has been built, is functional and has been positively assessed by an independent assessor. The paper explains, how this OS-solution based on embedded Linux enables the utilization of Linux in context of functionally safe and secure systems, effectively and efficiently.**

Keywords—embedded Linux, Functional Safety, Cybersecurity, maintenance, hypervisor, supervisor, SIL 2, ASIL B, safety applications, Kernel, safe Linux

I. INTRODUCTION

Software defined functionality is on the rise in various industries. Very often those hold requirements associated with Functional Safety and Security. Linux would be an ideal fit as an embedded operating system. And it comes with a large multiplicity of customizable features, established reliability, and stability across various use cases. But as Linux has not been developed with the prescriptions on safety-standards in mind: what about safety and how to build safe and secure solutions based on embedded Linux that meet the requirements of accepted standards and regulations, such as e.g., IEC61508, ISO26262, or IEC62443 and "Cyber Resilience Act" with guidelines such as TR-03183?

This paper describes a solution that enables the industry to make use of the benefits of open source software within an embedded OS-solution for safety-related applications up to SIL 2 according to EN 61508 and up to ASIL B according to ISO 26262. It is based on a "supervisor" software layer (named Supervisor) that detects and prevents undesirable behaviors of the Linux kernel. This supervision over the Linux kernel is enabled by a Hypervisor that leverages functionalities provided by the hardware platform. The approach decouples the lifecycle of the open-source Linux kernel, from that of other software elements, including the Supervisor, the Hypervisor, userland libraries, and the application software itself, all of which must comply with applicable safety standards.

But safety does not come without Security. And safe solutions need an adequate security concept along with its technical implementation. This mandates i.e., a modular lifecycle-management, a secure boot chain, or continuous integrity checks. CVE security monitoring as well as the supplementary Maintenance Monitoring need to accompany the embedded Linux solution.

II. CORE REQUIREMENTS/ REQUESTS, THAT MOTIVATE LINUX

Modern control systems, and more generally cyber-physical systems, must address many challenges originating from the ever increasing complexity and diversity of the functionalities to be implemented and from the technological advancement of the hardware. And this comes with a lot of challenges, that call for an adequate operating system, and Linux seems to tick all the boxes (acc. to [1]):

- open source with no vendor lock-in and the flexibility in utilization

  (e.g. in context of export control as outlined by the Linux Foundation [2], rule on „Securing the Information and Communications Technology and Services Supply Chain: Connected Vehicles“ [3] as published by the US Bureau of Industry and Security or product liability as reported in „ProdHaftRL“ resp. Directive (EU) 2024/2853 [4])
- broad support for diverse hardware
- rapid evolution and continuously maintained
- built for performance and Security
- broad API and a huge variety of features that support application development
- established tools and build infrastructures
- broad utilization in the most diverse and demanding contexts providing an absolutely non-episodical evidence of its reliability
- huge and extremely skilled community-base

While in the past connectivity of all embedded devices and machines has been rarely established in some industries, digital elements now typically do have a type of connectivity such as Bluetooth, WiFi, LAN, USB or other. But even those devices, that lack an intended interface to a networked system can be subject to interference via physical connector on PCB level.

This leads to the omnipresent risk of so called Cybersecurity attacks/ cyberattacks, Acc. to [11] this is understood as “any intentional effort to steal, expose, alter, disable, or destroy data, applications, or other assets through unauthorized access to a network, computer system or digital device.” And as awareness on this grows, several regulations, standards or guidelines arise or former ones are refined and extended. As a reference only and without listing all regulations and guidelines across all industries, it is worth to mention here IEC62443 [5] but also the so called Cyber Resilience ACT [7] with its national interpretations and technical regulations as provided by BSI ([8], [9]).

A major obligation that comes with this is vulnerability handling, which aims on freedom from known vulnerabilities for digital elements throughout the element’s support period acc. to [9]. Vulnerability handling further needs to include some basic requirements. Selected ones which motivate the discussion herein, are (simplified and derived from [9]):

- The manufacturer[1] needs a process to identify vulnerabilities affecting the TOE[2].
- The manufacturer needs a documentation for vulnerabilities, its impact and how they can be mitigated.
- The manufacturer needs to mitigate vulnerabilities in a timely feasible manner .

This responsibility along with its obligations is not new or even unknown when working in regulated context. Those developing and producing safety-related components and systems are used to this. And a lot of embedded systems are used in safety-related context. Acc. to the prescriptions of the safety standard in use, many measures have to be put in place to show, that the component or system does exactly what it is intended to do but nothing else. The rigor to which those measures are selected and implemented depend on the risk the corresponding system function can cause commonalities and differences in between Functional Safety and Security which are relevant to this article, can be summarized as follows:

- Both, Functional Safety and Security as a system property need to be managed throughout the lifecycle of the component or product from engineering to disposal.
- Both, safety-engineering and security-engineering work with assumptions, model the real-world complexity and validate both throughout the development lifecycle. An assessment is typically done before putting a product onto the market and a a certification body is involved [3].
- Update and modification typically needs re-assessment done by an assessment body.
- Continuous modification is not intended to be done for functionally safe elements. But for secure elements it is required: assumptions made on an element’s context will change continuously especially in context of Security. E.g., the model on how attackers behave and the tools they have will continuously evolve. Vulnerabilities will exploit and as such, risk needs to be re-assessed and modifications be made throughout an elements lifetime. As pointed out in [10], that in the following years AI will further impact the threat landscape. It will change as „malicious actors are launching attacks at machine speed and scale“.[10] further emphasizes, that security measures need to „evolve in tandem with the the sophistication of attackers“.

1 “The manufacturer” in particular and further roles in general are defined in [6], book 1, section 3.3.1.

2 TOE: Target of Evaluation as defined in [6], book 1, section 3.2.

3 Standards and regulations need to be considered with care as there are different level of rigor also defined for assessments and/ or audits (e.g. as defined in [6], book 1-1, section 3.2.10.).

As an essence from what has been described before and for what is relevant throughout this paper, three key-properties need to be balanced, and are relevant to an OS-solution based on embedded Linux:

- Functional Safety,
- Cybersecurity, and
- long-term Maintainability

And while Linux is known for being well long-term maintained and designed for Cybersecurity, the lack of a safety-centric focus throughout its design and development as well as the area of conflict in between continuous maintenance and Functional Safety need to be addressed.

III. Areas of conflict when using Linux in highly regulated context

A. *Functional safety and Linux*

Linux has not been developed with regulatory requirements on Functional Safety in mind. While domain-specific solutions are available, there has not been a product solution available that has received a positive and very far-reaching assessment from a certification body which applies to both EN 61508 (SIL 2) and ISO 26262 (ASIL B, SEooC).

Whether it is machinery, railway signaling, automotive, biomedical, or even home appliances, any domain is subject to the prescriptions of standards, norms, regulations, and also laws, with which the Linux development cycles did not comply. There has been no approach to maintain any claim on compliance with each update issued, which happens quite frequently and which is one of the advantages of Linux (acc. to [1]). Two key issues we want to point out herein are:

- argument and evidence on process compliance
- argument and evidence on freedom from interference in between Linux and safety-relevant userland applications

  note: this implies also freedom from interference in between DMA [12] devices and userland applications.

Unfortunately, there has been no generic solution available at present that shows process compliance of Linux to any standard or regulation on functional safety with all its lifecycles, especially architecture, design and modification although various companies and consortia look at different aspects (e.g., ELISA [14] and [13], opentech [15], RedHat [36], Distributed and Embedded System Lab [16])[4].

Various attempts made or being made to show correctness and freedom from interference often share a common approach that can be summarized as one or the other of the many flavors of “reverse engineering” (as mentioned in [1]).

Following the explanations as given in [1], the common idea mostly is, although with different levels of sophistication, is to demonstrate by analysis and testing that Linux “does what it says on the tin”, with the added difficulty that:

- Linux does not have the “tin” where to read its intended functionalities, meaning that the design information is sparse and distributed and does not guarantee its completeness and correctness and
- Linux is designed for performance and Security. It is a huge blob of software. The Linux (its kernel) is monolithic. Its services run in a single address space and as such they do have full access to the memory. This implies kernel-memory, device-memory but also memory assigned to userland processes. This access is immanent to the nature of Linux. While such a design comes with the advantage of performance (amongst others), it is close to impossible (not in theory but practically) to show, that the Linux kernel does not adversely effect userland processes without any further measures. awhile maintaining its main properties: performance and Security.

Such an approach is labor-intensive and time-consuming: just to name the most obvious issue, the almost infinite number of internal states of Linux requires an extremely extensive testing activity and an even more extensive analysis to demonstrate that such a testing activity is sufficiently exhaustive.

Even assuming that such an approach is successful, it would need to be repeated each time an update to Linux is issued, which happens quite frequently (and which is one of the advantages of Linux, as stated above); it is true that an appropriate impact analysis could reduce the effort required, but even such an impact analysis would take time and would be expensive.

On this basis, **area of conflict 1** can be derived**:** Showing process compliance, correctness and freedom from interference for a huge blob of SW as Linux is extremely labor-intensive, time-consuming and hardly maintainable. Due to the nature of Linux, the kernel does have full access to userland applications.

B. *Cybersecurity and long-term maintenance while maintaining claims on functional safety*

One of the core advantages besides the broad support of hardware and performance is, that Linux is developed in the community and continuously maintained. There is no vendor lock-in. Everybody can look at the sources. And everybody can contribute. This trait of open source software is one of the motivators why it is treated differently from various rules/ laws ([2][3][4]).

The Linux kernel has a new version every 8 to 12 weeks. And developers and researchers all over the world find bugs and exploits and send fixing patches almost every day to the Linux source code (see [37] or [38]) which lead to updates, that are available to all users. Management of these fixing patches is an established process done by system administrators typically and it serves the need to maintain Cybersecurity.

4 Design alternatives based e.g. coded processing are not considered here as those approaches differ from the main idea followed herein: use Linux as is.

While this is of great value for all those selling devices, which have to consider security requirements given in industry-specific standards, this comes with the following needs:

- continuous re ensuring Functional Safety
- continuous re ensuring Cybersecurity acc. to standards, regulations and guidelines

This is because any change can adversely effect the safety argument and the related supporting evidence on Functional Safety or Cybersecurity[5] and comes with the risk to add unintended functionality within the usage profile/ assumed context for the element under consideration.

It is essential, that the software in use is owned and considered trustable. It is fit for its purpose/ for use in its assumed context and asset owners (for role-definition see e.g., [18]) can claim this with sufficient rigor, qualitatively or quantitatively. Ownership can be interpreted from a practical and a formal point of view. Practically, ownership is understood herein as:

- it is known, from where the software comes and which systematic capability and as such, we know whether the software fits to its purpose.
- it can be build in a systematic way from source
- it can be modified in a controlled way without breaking functionality, regressions and w/o adding unintended features.
- it can be used in a controlled way.
  - its intended functionality is known by adequate documentation.
  - its intended functionality is provided and can be tested
  - no unintended functionality is implemented within the assumed context of use with its tests.

Ownership also comes with the obligation on open source license compliance. OpenChain 2.1 which is now ISO/IEC 5230:2020 [19], the International Standard for open source compliance defines processes one has to act upon.

As a consequence, the larger the SW is, that needs continuous maintenance to ensure Cybersecurity, the more labor-intensive and time-consuming it is to do the maintenance. Evidences needed by arguments on Functional Safety, Security and further standards need to be updated, the compliance argument re-evaluated and eventually even re-assessed by a certification body.

Area of conflict 2: Continuously maintaining arguments, analysis and supporting evidence on Functional Safety along all patches for a huge blob of SW as Linux throughout the lifetime of the corresponding product or component, is extremely labor-intensive, time-consuming and hardly doable.

IV. PROPOSED TECHNICAL SOLUTION

We are looking for an OS-solution based on embedded Linux, that implements the need for Functional Safety, Cybersecurity, and long-term Maintenance with reasonable effort. Not to forget the feature-richness that comes with Linux along with the aspects mentioned in section II.

A. Basic strategies and derived principle

Rather than following the attempt to show that Linux " does what it says on the tin" [1], the innovation is based on a completely different approach compared to established „safety" approaches. The "burden of the proof" is shifted from Linux to a "supervision software layer" detecting when Linux does not behave dependably.

In other words, rather than trying to demonstrate that Linux is dependable, it is detect when it isn't.

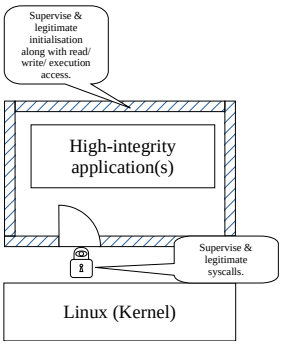

Fig. 1. *The basic principle used to guide the design of the OS solution for Safety Applications based on Linux*

The solution follows the following strategies (acc. to [20]):

- Change the paradigm: use the strength of Linux and detect when things go wrong rather than trying to prevent faults. Let the kernel run as usual. Do not attempt to change it. Use it. And instead, "put it into a box".
- Just indicate once integrity cannot be ensured and allow fault-reactions to be added as needed by the project.
- Focus on an application's data space. Ensure correctness. Make it a dependable [21] data space.
- Separate lifecycles of all building blocks (Hypervisor, Supervisor, kernel, userland applications with its libraries).

The basic principle is depicted in the following Fig. 1.

B. Basic functional and architectural concept

The solution implemented leverages upon the features offered by advanced hardware to supervise the behavior of Linux, namely its access to memory and processing resources.

Two main software elements implement this solution ([1]):

- a Hypervisor provides Linux with virtual memory and computation resources, hence the hypervisor has full control over the access to those resources by Linux
- a Supervisor software analyses any attempt made by Linux to access memory or computation resources and detects when such an attempt is able to adversely

5 The argument with its reference to supporting evidence is typically summarized in assurance cases [17].

affect the dependability of the safety function (with the terms used as in EN 61508).

The Hypervisor with its Supervisor software implement duplication of the control structures of virtual memory and computation resources. The representation of the memory map and the relevant hardware registers, which we call "state of the System on Chip [31]", is available twice: high-integrity and low-integrity. Fig. 2 depicts this duplication in the middle using the symbol of a database. One is used for non and the other for safety-related intermediate physical memory with its mapping to physical memory. This duplication is done in between intermediate physical and virtual addresses. While the OS maintains the virtualization on the level of virtual addresses, the hardware with the Supervisor maintain the translation of intermediate physical to physical addresses.

The terms "high-integrity" and "low-integrity" are used for convenience throughout this text. "high-integrity" is used to characterize hardware resources, which are supervised and as such, for which the Supervisor is used to detect adverse events. An adverse event happens if any software element other than high-integrity application(s) itself modifies the data generated, used or managed by the high-integrity Application(s). "low-integrity" is used to characterize unsupervised hardware resources.

Once the safety-related application is executed, the high-integrity representation is used. If not, the low-integrity representation is used. Access to the high-integrity domain is strictly controlled. Therefore, the two main SW elements [20]:

- supervise and legitimate user-space initialization with its processes, for the start user space applications
- separate read/ write/ execute-rights onto memory pages for kernel and user space applications
- supervise and legitimate any read/ write/ execute attempt on the high-integrity application memory
- supervise and legitimate updates on registers holding the processor state throughout context switches caused by switching in between userland applications and kernel
- indicate any fault detected

C. *Main interaction in between HW and SW: multi-level address virtualization*

As outlined at [20] the concept does heavily rely on virtual address spaces with its settings and tables for all the translations which are often called „translation regimes" [32].

And while the OS controls the set of translations from virtual memory to what the Software sees as physical memory, a hypervisor can control another set of translations. Those map the addresses with its settings as seen by the OS to the real physical address space. With this approach, a so called intermediate physical address space is placed in between the virtual one and the physical one. This layer is under full control of a Hypervisor. A much more complete description on this is given at Arm's AArch64 memory management Guide [25]. As virtualization does not only focus on memory but on HW in general, the aforementioned description is also applied to CPU registers. The supervision of DMA-devices works similarly and demands the corresponding HW-support.

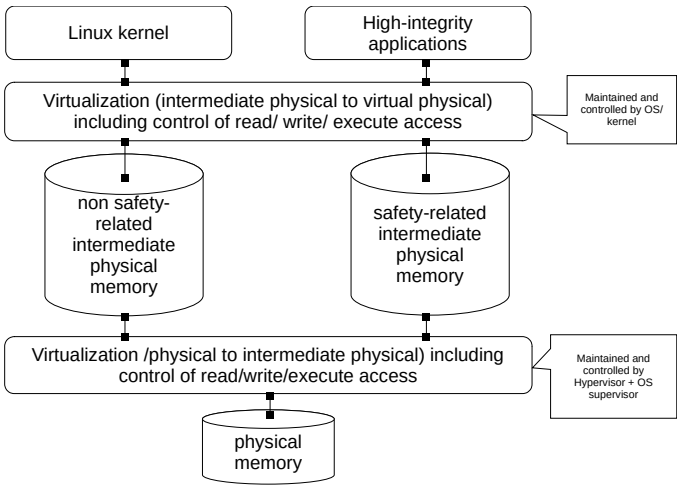

Fig. 2. *Two stages of virtualization used as one of the core HW-features*

D. *Applicability of the solution*

As outlined in [1] the immediate advantage of this is that the dependability of the safety function performed by a cyber-physical system does not rely on Linux itself, but on the above-mentioned two software elements, while at the same time it allows to exploit all the features of Linux.

As a consequence, the effort required to build the safety argument and the related supporting evidence is reduced, because only the two above-mentioned software elements are directly involved, but also the update to a new version of Linux requires a moderate effort and, above all, even if performed incorrectly it would not affect the safety but only the availability.

Another remarkable advantage of this solution is that it allows to execute both safety-related (supervised) and non-safety-related (unsupervised) applications at the same time on the same Linux; the non-safety-related application is like any other application running on Linux and is not affected or functionally limited by the presence of the safety-related application[6].

As such, EB corbos Linux for Safety Applications is the first and only Linux OS-solution to comply with SIL 2/ASIL B safety requirements [33]. It comes with at least one execution environment based on Linux, that supports mixed-criticaly: low-integrity and high-integrity applications can be executed next to each other while using the same kernel. And as it is based on Linux, it opens up the utilization of established tools and libraries, that fit to the customer's projects.

6 The ability to execute two software elements that are associated with different levels of criticality on the same HW system is also called "Mixed Criticality" [34]. The term is typically used on the level of systems. On the level of operating systems, also the term "Mixed-Criticality OS Environments" is used [35].

[1] stated that a minimum viable product (which can be considered a technological demonstrator) has been built and is functional while an independent assessor has confirmed not only the dependability of the software, but also that a cyber-physical system implemented using this solution is able to:

- perform safety functions up to SIL2 according to EN 61508
- fulfill safety requirements up to ASILB according to ISO 26262.

This makes this solution suitable for almost any regulated industry, including automotive; a fully-featured version is currently being developed.

The user effort required to positively assess a system developed using the OS solution is reduced to the minimum in terms of analysis, documentation and testing.

And even for the achievement of higher SILs or ASILs, it offers a broad range of possibilities that can be exploited on a project-specific basis.

For more advanced versions, multiple different independent virtual domains are supported. Also a low level of abstraction is supported with a virtual domain directly hosted by the Hypervisor. Further virtual domains can be added with a reasonable effort because their independence is largely covered by the already achieved positive independent safety assessment.

The above mentioned independent domains further allow to apply techniques (like, for instance, diversity, redundancy or cross-check) able to achieve the required level of safety integrity for the most demanding projects.

Architectural patterns that can be built with this solution described are briefly described online within a blog-post [22]. The core misconceptions that prevent stakeholders to consider OS solutions based on Linux for safety related systems are also analyzed therein.

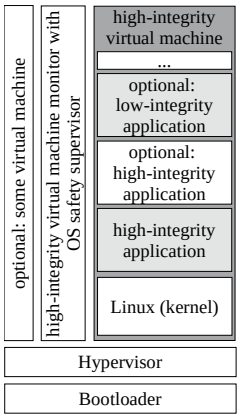

Fig. 3. *Basic software architecture*

Fig. 3 depicts the basic architectural setup with the core building blocks such as Bootloader, Hypervisor and multiple virtual machines. The virtual machine monitor for the high-integrity virtual machine is extended with the Supervisor. As mentioned before, this Supervisor is only used for high-integrity applications while low-integrity applications maintain unsupervised.

E. Interplay of lifecycle elements and SW maintenance

Not only the SW-architecture does reflect a high degree of modularization. It goes hand in hand with the modularization of the safety argument and the related evidence with the corresponding lifecycles of core building blocks such as Hypervisor, Supervisor, kernel and userland applications with its libraries.

And as Functional Safety mainly relies on the two SW elements Hypervisor and Supervisor as described in subsection B, the effort required to build the safety argument and the related supporting evidence is reduced, because only the two software elements Hyper- and supervisor are directly involved.

Asset owners and product suppliers[7] are enabled to efficiently and effectively implement vulnerability analysis along with update management. The corresponding processes can largely be implemented and operated independently.

Security features known in Linux are available in the optional virtual machine which can operate as gateway. But security features can also be enabled within high-integrity virtual machine with its mixed-criticality user-space domain: non-safety-related and safety-related features can seamlessly be orchestrated and used.

V. SUMMARY ON WHY THE SOLUTION CAN ENABLE CUSTOMERS, TO USE LINUX IN HIGHLY REGULATED CONTEXT

In December 2024 emlix listed five misconceptions on Linux and Functional Safety that may deter system and software architects from considering Linux and hypervisors as core building blocks in context of safety related cyber-physical systems [22]:

- open-source processes and software cannot be used in context of safety-related systems.
- Linux needs to be assessed as a monolithic software-blob.
- Virtualization is a source of additional problems, requiring extra efforts for qualification and maintenance, and demands expertise that is rarely available on the market.
- Each single element of an execution environment including its libraries need to be “safe”
- There is only one architectural approach to meet all functional and non-functional requirements using Linux

In context of Security it is worth to bring in and emphasize one more, which is [23]:

- An OS-solution based on Linux requires continuous re assessment with each and every update/ patch .

7 There are a lot more stakeholders with different names given in different standards and regulations. Two of them are used here as defined in the series of IEC standards on Industrial communication networks – Network and system security, IEC62443. The limitation used here has been done for convenience only and does not exclude other roles and stakeholders.

EB corbos Linux for Safety Applications [33] shows that all the previously mentioned misconceptions are no more true. The OS-solution pairs the nature of Linux that perfectly utilizes the technological advancements of the hardware alongside with compliance with the mandatory functional safety prescriptions required in various domains to perform safety functions to regulatory standards such as e.g. those on functional safety.

Acc. to [24], EB corbos Linux for Safety Applications:

- is based on Linux,
- complies with the mandatory functional safety prescriptions required in various domains to perform safety functions up to SIL2 according to EN 61508 and up to ASILB according to ISO 26262,
- supports mixed-criticality which means, that safety-related and non-safety-related applications can run on the same kernel,
- supports different architectures with or without multiple domains and with or without containers,
- supports long-term maintenance (up to 15 years) and security support,
- is largely compatible with the features and interface of any Linux while considering best practices to support Security,
- comes with a safety-certified tool-chain and libraries, warranty and liability and
- it is available for free and ad hoc for use in demonstrators.

The free edition available at [24] ensures, that anyone can start right away and today. One just has to follow the link provided above. It comes with a reference setup consisting of two virtual machines, pre-configured userland setup, and demo applications. It supports logging and warns if system calls [29] are invoked that are not allowed in a high integrity context.

The OS-solution enables system and software architects to consider Linux and hypervisors as core building blocks in context of safety related systems: instantly.

The team currently works on the evolution of the MVP into a fully featured, industrialized, commercially available product supporting a growing variety of Linux functionalities and system calls. Moreover, an infrastructure enabling customers to build their systems is being developed.

Integrators and application developers can also use any other embedded Linux SDK in the first instance; porting the developed high integrity applications to the solution described herein will be easy to handle as long as some basic conditions are considered:

- use musl libc [26]
- use libc++ [27]
- use a proper and qualified startup process
- use specific system calls rather than those, that can hardly be analyzed out of a customer's functional context (e.g. ioctl [28])

It is worth noting that typically similar principles apply also to secure and hardened embedded Linux solutions anyway. It is assumed, that the aforementioned conditions are most probably part of a customer's engineering guideline that does consider domain specific security standards.

The OS-solution is currently designed for aarch64 v8 architectures. Support for further aarch64 versions with its SOCs is planned.

Acknowledgment

As part of the innovative development project together with Elektrobit Automotive GmbH in Erlangen, emlix is deeply involved in the development of the Supervisor software.

The authors do like to express their deepest appreciation to the whole development team at emlix and Elektrobit Automotive GmbH. The support and engagement is inimitable. And the deep expertise of all the contributors in all the different aspects of technology and processes has been a key enabler to transfer the initial thoughts to a real product that can be presented to the market. Many thanks also to all the sponsors and decision makers for the continuous support and the unconditional trust given to the technologists. This development project continuously proved the relevance to take decisions following objective decision records and technical facts. We feel proud and grateful for the opportunity to contribute and to accompany the journey we went through and are looking forward to enable customers to use this secure OS-solution for Safety-Applications based on embedded Linux.

References

[1] F. Arrighetti, M. Armbruster, U. Kirchmaier, M. v. Czettritz, D. Glöckner, M. A. J. Butt, U. Hildebrand, "Linux für sicherheitsrelevante Anwendungen - Ermöglicht den Aufbau sicherer und geschützter Systeme mit Linux", unpublished abstract for safe.Tech 2025.

[2] S. Winslow, M. Dolan, J. Perlow. Understanding US export controls with open source projects. Linux Foundation, [online] https://www.linuxfoundation.org/resources/publications/understanding-us-export-controls-with-open-source-projects

[3] Bureau of Industry and Security, Department of Commerce .Securing the Information and Communications Technology and Services Supply Chain: Connected Vehicles.. [online] https://www.federalregister.gov/documents/2025/01/16/2025-00592/securing-the-information-and-communications-technology-and-services-supply-chain-connected-vehicles#p-130

[4] European Parliament and of the council (2024, Oct. 18). DIRECTIVE (EU) 2024/2853 on liability for defective products and repealing Council Directive 85/374/EEC. Accessed on: Feb. 7, 2025. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32024L2853&qid=1738051140257

[5] IEC 62443 series of standards "Security for industrial automation and control system".

[6] "Understanding IEC 62443" (2021, Feb. 26). Accessed on: Feb. 7, 2025. [ONLINE]. Available: https://www.iec.ch/blog/understanding-iec-62443

[7] European Parliament and of the council (2024, Oct. 20). REGULATION (EU) 2024/2847 on horizontal cybersecurity requirements for products with digital elements and amending Regulations (EU) No 168/2013 and (EU) No 2019/1020 and Directive (EU) 2020/1828 (Cyber Resilience Act). Accessed on: Feb. 7, 2025 . [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=OJ:L_202402847

[8] Federal Office for Information Security. Cyber Resilience Act - Cybersecurity in the EU. . Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Cyber_Resilience_Act/cyber_resilience_act_node.html

[9] Federal Office for Information Security. BSI TR-03183: Cyber Resilience Requirements for Manufacturers and Products. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03183/TR-03183_node.html

[10] Darktrace, (2024). “State of AI Cybersecurity” Accessed on: Feb. 7, 2025 . [Online]. Available: URL https://darktrace.com/resources/state-of-ai-cyber-security-2024

[11] IBM. What is a cyberattack?. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.ibm.com/think/topics/cyber-attack

[12] J. Corbet, A. Rubini, G. Kroah-Hartman (2005). Chapter 15. Memory Mapping and DMA. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch15.html

[13] I. Stoppa, Using Linux in Safety Scenarios. Accessed on: Feb. 7, 2025 . [Online]. Available: https://elisa.tech/blog/2024/03/04/using-linux-in-safety-scenarios/

[14] S. Khan (2021). White Paper: Advancing Open Source Safety-Critical Systems. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.elisa.tech/wp-content/uploads/sites/75/2021/05/ELISA_Advancing-Open-Source-Safety-Critical-Systems_Whitepaper_050521.pdf

[15] A. Platschek (2016, Dec. 1). DB4SIL2 - Kernel assurance data for SIL2LinuxMP. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.sambuz.com/doc/db4sil2-kernel-assurance-data-for-sil2linuxmp-ppt-presentation-1001163

[16] Distributed and Embedded System Lab, Information Science and Engineering college, SIL4Linux Web Interface. Accessed on: Feb. 7, 2025. Lanzhou University. [Online]. Available: DSL http://sil4linux.dslab.lzu.edu.cn/

[17] M. Klessascheck (2020, Nov. 10). Safety Assurance Cases: Leidvolle Diskussionen mit Auditoren abkürzen. Accessed on: Feb. 7, 2025. [Online]. Available: https://www.johner-institut.de/blog/iso-14971-risikomanagement/safety-assurance-cases/

[18] ISA Global Cybersecurity Alliance (2020, Oct.). Security of Industrial Automation and Control Systems. Accessed on: Feb. 7, 2025 . [Online]. Available: https://21577316.fs1.hubspotusercontent-na1.net/hubfs/21577316/2022%20ISA%20Website%20Redesigns/ISASecure/PDFs/Miscellaneous%20PDFs/Documents-Articles-and-Technical-Papers/ISAGCA-Security-Lifecycles-whitepaper.pdf

[19] S. Coughlan (2020, Dec. 15). OpenChain 2.1 is ISO/IEC 5230:2020, the International Standard for open source compliance.. Accessed on: Feb. 7, 2025 . [Online]. Available: https://openchainproject.org/featured/2020/12/15/openchain-2-1-is-iso5230

[20] M. Armbruster, F. Arrighetti (2024), “Linux for safety-related applications”. exida Automotive Symposium, Spitzing, 2024.

[21] “dependability,” International Electrotechnical Vocabulary. IEC 60050. section 192-0122. Accessed on: Feb. 7, 2025 . [Online]. https://electropedia.org/iev/iev.nsf/display?openform&ievref=192-01-22

[22] M. Armbruster (2024). A discussion on misconceptions with Linux and functional safety. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.emlix.com/de/about/tech-trends/blog/linux-safety-news-a-discussion-on-misconceptions-with-linux-and-functional-safety/

[23] M. Armbruster (2025, Jan. 08). Tomorrow is now. Accessed on: Feb. 11, 2025 . [Online]. https://emlix.com/en/about/tech-trends/blog/linux-safety-news-tomorrow-is-now/

[24] Elektrobit Automotive GmbH (2024). Try EB corbos Linux for Safety Applications for free. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.elektrobit.com/products/ecu/eb-corbos/linux-for-safety-applications/free/

[25] arm. Learn the architecture - AArch64 memory management Guide - Address spaces. Accessed on: Feb. 7, 2025 . [Online]. Available: https://developer.arm.com/documentation/101811/0104/Address-spaces

[26] musl libc. Accessed on: Feb. 7, 2025 . [Online]. Available: https://musl.libc.org/

[27] “libc++” C++ Standard Library. Accessed on: Feb. 7, 2025 . [Online]. Available: https://libcxx.llvm.org/

[28] ioctl(2) — Linux manual page. Accssed on: Feb. 7, 2025 . [Online]. Available: https://www.man7.org/linux/man-pages/man2/ioctl.2.html

[29] syscalls(2) — Linux manual page. Accessed on: Feb. 7, 2025 . [Online]. Available: system calls https://www.man7.org/linux/man-pages/man2/syscalls.2.html

[30] patch(1) — Linux manual page. Accessed on: Feb. 7, 2025 . [Online]. Available: patch https://www.man7.org/linux/man-pages/man1/patch.1.html

[31] arm. ARM Cortex-A Series Programmer's Guide for ARMv7-A - System-on-Chip (SoC). Accessed on: Feb. 7, 2025 . [Online]. Available: https://developer.arm.com/documentation/den0013/d/Introduction/System-on-Chip--SoC-?lang=en

[32] arm. Learn the architecture - AArch64 virtualization Guide – Stage 2 Translation. Accessed on: Feb. 7, 2025 . [Online]. Available: https://developer.arm.com/documentation/102142/0100/Stage-2-translation?lang=en

[33] Elektrobit Automotive GmbH (2024). Open-source operating solution for safety. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.elektrobit.com/products/ecu/eb-corbos/linux-for-safety-applications/

[34] R. Martin (2022, Sep. 14). Why Mixed-Criticality Is the Future of Automotive Architectures. Accessed on: Feb. 7, 2025 . [Online]. Available: https://blogs.blackberry.com/en/2022/09/why-mixed-criticality-is-the-future-of-automotive-architectures

[35] Windriver . What Are Mixed-Criticality OS Environments?. Accessed on: Feb. 7, 2025 . [Online]. Available: https://www.windriver.com/solutions/learning/what-are-mixed-criticality-os-environments

[36] RedHat (2021, Jul. 2). Functional safety and continuous certification on Linux. Accessed on: Feb. 10, 2025 . [Online]. Available: https://www.redhat.com/en/topics/open-source/functional-safety-and-continuous-certification-on-linux

[37] (2025, Feb. 10). Kernel Evolvement. Accessed on: Feb. 10, 2025 . [Online]. Available:https://remword.com/kps_result/evolvement.php

[38] The Linux Foundation (2020, Aug.). 2020 Linux Kernel History Report. Accessed on: Feb. 10, 2025 . [Online]. Available: https://project.linuxfoundation.org/hubfs/Reports/2020_kernel_history_report_082720.pdf?hsLang=en