Linux for safety-related applications - Making it possible to build safe and secure systems with 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.

A good example is the automotive industry: the almost romantic mechanical car has evolved into the software-defined vehicle where electronic control systems are used for the most diverse purposes, from engine control to infotainment, from driving assistance to air conditioning, from connected automated driving to accidents prevention.

The above mentioned challenges call for an adequate operating system, and Linux seems to tick all the boxes:

- its open-source nature offers a rapid evolution cycle, continuous maintenance, and quick bug-fixing, while at the same time eliminating development costs, without any vendor lock-in
- its hardware support already covers the greatest majority of applications
- its security features provide protection against malicious attacks, especially for interconnected and always-on embedded devices
- its scalability allows to select only the features or functionalities (“packages”) actually required for the specific needs of the intended application
- its extremely broad use in the most diverse and demanding contexts provides an absolutely non-episodical evidence of its reliability
- highly skilled and very experienced developers are broadly available on the market.

Unfortunately, at least until yesterday, Linux did not tick the most critical box: safety.

Whether it is machinery, railway signalling, automotive, biomedical (but also home appliances) any domain is subject to the prescriptions of standards, norms, regulations, and also laws, with which Linux did not comply.

The attempts made or being made to address this issue share a common approach that can be summarised as one or the other of the many flavours of “reverse engineering”.

The common idea behind these attempts, 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 any “tin” where to read its intended functionalities, meaning that the design information is sparse and does not guarantee its completeness and correctness.

Such an approach is labour-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.

This presentation shows a completely different approach where 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, the choice has been to detect when it is not.

The solution implemented leverages upon the features offered by advanced hardware (whether it is a microprocessor or a system on chip) to supervise the behaviour of Linux, namely its access to memory and processing resources.

Two main software elements implement this solution:

- a hypervisor provides Linux with virtualised 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 affect the dependability of the safety function (in the acception given by EN 61508).

The most immediate advantage of this solution 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 and advantages 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 only moderate effort and, above all, even if performed incorrectly it would not affect the safety but only the reliability.

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.

This solution has proven successful: 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
- fulfil 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.

Although the independent assessment has been performed for SIL2/ASILB, this solution offers features and functionalities that can be exploited to implement the most appropriate fault detection and mitigation techniques achieving even higher levels of integrity.

Presenter and co-authors:

- Federico Arrighetti (Federico.Arrighetti@elektrobit.com, presenter),
- Michael Armbruster (Michael.Armbruster@emlix.com),
- Ulrich Kirchmaier (Ulrich.Kirchmaier@elektrobit.com),
- Michel von Czettritz und Neuhaus (mvc@emlix.com),
- Daniel Gloeckner (Daniel.Gloeckner@emlix.com),
- Muhammad Aqib Javaid Butt (Aqib.Javaid@elektrobit.com),
- Uwe Hildebrand (Uwe.Hildebrand@elektrobit.com)