EN-50128 certification-oriented design of a safety-critical hard real-time kernel

Ciro Donnarumma
Rete Ferroviaria Italiana S.P.A.
Firenze, Italy
Scuola Superiore Sant'Anna
Pisa, Italy
c.donnarumma@rfi.it

Pietro Fara
Scuola Superiore Sant'Anna
Pisa, Italy
pietro.fara@sssup.it

Gabriele Serra
Scuola Superiore Sant'Anna
Pisa, Italy
gabriele.serra@sssup.it

Sandro Di Leonardi
Scuola Superiore Sant'Anna
Pisa, Italy
sandro.dileonardi@sssup.it

Mauro Marinoni
Scuola Superiore Sant'Anna
Pisa, Italy
m.marinoni@sssup.it

***Abstract*—The growing complexity and the need for high safety standards in railways infrastructures are pushing the infrastructure operators toward the adoption of newer solutions able to exploit modern platforms and state-of-the-art software solutions while guaranteeing safety and timing constraints, and maintaining the compliance with the standards. This paper presents the design guidelines of a novel real-time kernel whose development is based on the Italian use case, highlighting its focus on adherence to the standards.**

***Index Terms*—real-time, kernel, safety-critical, EN-50128**

I. Introduction

In recent years, the need to modernize the railway infrastructure, both in terms of technology and safety, is pushing many infrastructure operators towards the design of ad-hoc real-time systems for railway management. In such systems, high predictability, safety, and security have to be guaranteed in all operating conditions since all monitoring, control, and actuation functions are implemented in software and must be executed under stringent timing constraints. Rete Ferroviaria Italiana S.p.A. (RFI), the Italian railway infrastructure operator, aims to develop the next generation railway management products, built on top of a custom-made safety-critical real-time kernel. The development is underway within a project involving the R&D department of RFI and the ReTiS Lab of Scuola Superiore Sant'Anna to design and implement a reliable, secure, and real-time kernel with the ability to run on different hardware platforms, especially multi-core ones. The design of the kernel requires a substantial effort to evaluate and determine the architecture best suiting the needs of railway applications and then proceed with the implementation, that must be in line with the stringent regulations required to proceed to software certification such as the European standard *EN-50128 - Railway applications* [1]. The aforementioned standard specifies procedures and technical requirements for the design and development of software and programmable electronic systems used in railway control and protection applications. Note that, the standard has a particular focus on the methods which need to be used to provide software meeting the demands for safety integrity. In order to cope with standard regulation and to simplify the certification process, the kernel development focused on the design of an ad-hoc architecture able to provide to critical applications a fine *timing management* mechanism and the *fault isolation* property.

1) Timing management: Timing constraints are part of the mandatory properties that have to be supported by the system, according to the standard (clause 4.1) [2]. As a consequence, the kernel must provide specific mechanisms for handling tasks with explicit timing constraints.

2) Fault-isolation: Faults, especially in safety-critical software, represent sources of interference. According to the standard, it must be demonstrated that software and hardware interact correctly to perform their functions (clause 7.6.1.1) and, obviously, this means stem faults and avoid data corruption [2]. Hence, a correct and robust solution able to provide the desired level of isolation among components is crucial to satisfy the requirements. If the system is well-partitioned, a failure in a given component does not propagate to another one that, maybe, is handling a higher critical job [3].

A great number of the real-time systems used to support control applications are based on modified versions of time-sharing operating systems. As a consequence, isolation and timing features offered at the upper layer are not suited to support safety-critical activities [4]. In conclusion, these important properties, together with other desired features such as code maintainability, require a carefully designed architecture.

The kernel is under development, and a prototype version already runs on Xilinx Zynq Ultrascale+ platform equipped with a quad-core ARM Cortex-A53. However, this paper will not discuss implementation issues or performance measurements; conversely, it presents some choices and guidelines relative to the kernel design with a particular focus on how they address the requirements coming from the EN-50128.

II. Motivations

There is a broad and growing range of safety-critical application fields requiring real-time computing, such as nuclear power plants, automotive and avionics systems, air traffic control, robotics, and military systems. In such environments, where a software crash may cause damage or even loss of life, preventing failures is crucial. Safety norms and regulatory requirements demand the production of a safety case, identifying potential functional and non-functional hazards and demonstrating that the software does not violate the related safety goals [2].

Relevant safety norms in the transport field include DO-178B, DO-178C, IEC-61508, ISO-26262, and EN-50128. The IEC-61508 "Functional safety of electrical / electronic / programmable electronic safety systems" represents the central standard on the functional safety of control systems. Both ISO-26262 and EN-50128 are adaptions of IEC-61508 for the application of electrical/electronic systems in specific fields. The ISO-26262 addresses the automotive domain, and its adoption is not part of the current EU automotive regulatory framework [5]. Instead, the EN-50128 specifies procedures and technical requirements for the development of software and programmable electronic systems used in railway control and protection applications, and the EU required its transposition into national legislation since 2011 [1]. Currently, few RTOS solutions conform to CENELEC EN-50128 safety norm, and many vendors offer commercial products adapting their existing RTOS. For instance, SYSGO recently received the EN-50128 certification for its PikeOS, the commercial RTOS designed for avionic solutions [6]. Another commercial RTOS compliant with the EN-50128 is INTEGRITY [7], developed by Green Hills. PikeOS and INTEGRITY share several common characteristics. For instance, both use a micro-kernel approach and provide device drivers at the user-level.

Instead, the solution proposed in this paper consists of designing from the beginning a safety-critical kernel to be used explicitly for the railway environment that reflects requirements and preferences advised by the infrastructure operator. A proprietary kernel helps to avoid developing and certifying code for unnecessary or legacy features and allows reducing the code-base considerably, bringing evident advantages during testing, certification, and maintenance. The kernel design focused on supporting state-of-the-art real-time solutions and thus dropping support for legacy solutions, to make it easier to verify applications built on top of it. Eventually, having to be explicitly certified only for a single set of standards, any update and adjustment of the entire system at each update of the norm is more accessible than that of other off-the-shelf solutions.

III. Architecture design

This section presents an overview of the main elements composing the design of the developed kernel, showing how they fit the requirements of the EN-50128 standard, with the aim of certification.

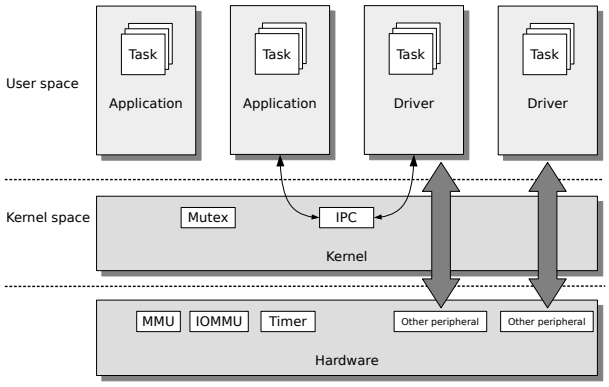

Fig. 1. Kernel's architecture.

Language and coding guidelines. As stated in clause D.54, the chosen programming language should lead to easily verifiable code to facilitate verification and maintenance [1]. The selected one has been the C programming language, due to the high grade of portability across a wide range of hardware and architecture and the intrinsic language flexibility. Furthermore, the norm highly recommends the use of coding guidelines (clause D.15). The Motor Industry Software Reliability Association (MISRA) consortium first published its Development Guidelines for Vehicle Based Software in 1994, which describes the set of rules and measures that should be adopted in software development. The kernel coding style follows the last MISRA set of rules, published in 2012 [8] and recognized as the de-facto standard for writing C code for safety-critical systems. Both MISRA (dir 4.12) and EN-50128 (clause D.54) remarkably discourage features which make verification more complex such as the use of heaps or any type of dynamic variables or objects. Thus, in order to satisfy the standard requirements, the kernel does not make use of dynamic memory. Applications, tasks, channels, and all the other entities of the system are set up in the *configuration phase* through a configuration tool.

Spatial-isolation and fault-containment. As depicted in Fig. 1, enforcement of spatial-isolation and fault-containment have been obtained applying the micro-kernel architecture, memory virtualization techniques, and a programming model based on tasks and applications. A task is the basic schedulable unit of execution, and an application is a group of tasks which share the same virtual addressing space. Tasks are executed concurrently and scheduled using fixed-priorities. In this way, the tasks belonging to the same application can cooperate through shared-memory communication paradigm, while the tasks belonging to different applications can communicate only using the message-passing mechanism offered by the kernel. With such a kind of architectural organization, tasks can be partitioned into applications, so that a fault of a task can propagate only to the other tasks of the same application.

According to the micro-kernel model principles, the drivers are full-fledged applications that execute in user-space, and they have their own virtual memory. The memory virtual-

ization mechanism is provided by the Memory Management Unit (MMU) that is responsible for the virtual to physical address translation of all the memory transactions issued by the CPU. Since the drivers can use DMAs and because memory transactions issued by the DMAs are not subject to MMU's permissions check, the isolation of the addressing spaces of the different applications could be broken. As described in [9], the IOMMU support provided by modern hardware architectures is exploited to controls all DMA's memory accesses and achieve a complete spatial-isolation. Therefore the DMA controlled by a device driver inherits read and write permissions of the associated driver, and the IOMMU checks the DMA's memory accesses according to the page-table of its driver. Therefore it read and write accesses are limited only to the address space of the associated driver.

Temporal-isolation. Although all the tasks should be fully characterized in their temporal features, some error could be done in the evaluation of their WCET and other time-related parameters. This kind of error can induce a task to execute more than expected, particularly in case of a system overload. In such a case, the wrong characterized task could delay the execution of the lower priority tasks which could miss their deadline. Temporal isolation techniques can resolve this issue in a way that only the wrong task will be penalized. We use the Sporadic-Server algorithm [10] to achieve the temporal-isolation of the tasks. By statically assigning a period and a budget (i.e., a maximum execution time) to each task, the kernel can enforce the temporal isolation at run-time. In such a way, it ensures that each task runs in each period at most for its budget and blocks any task that exceeds the assigned budget, without effecting other tasks.

Priority-inversion-free resources and deadlock avoidance. A primitive mechanism for tasks synchronization is one of the fundamental facilities that a kernel must provide. General-purpose operating systems provide generic semaphores, but they do not suite the constraints of hard real-time tasks because they suffer from the well-known problem called priority inversion. The mentioned issue introduces unbounded blocking on the execution of the tasks, making the schedulability analysis infeasible [4]. In order to meet the temporal constraints required by the safety norms, the proposed kernel provides *Resources* (i.e., mutual-exclusion mechanism) according to the *Stack Resource Policy* (SRP) [11]. SRP impacts the scheduling algorithm in a way that a task is not scheduled until all needed resources become free. Consequently, a task cannot be blocked on the access to a resource, and once it has acquired a resource, the higher priority tasks that could use the same resource cannot preempt it. Obviously, it can only be preempted by the higher priority tasks not using the same resource. This behavior permits to analyze the whole system because the waiting time experienced by a task depends only by the known temporal properties of the lower priority tasks. Note that SRP is a protocol that regulates the accesses to local resources (i.e., resources shared by tasks running on the same core). The kernel also supports Multiprocessor SRP (MSRP), described and developed by Gai et al. [12], to provide global resources (i.e., resources shared among tasks on different cores of a multi-core system). According to this protocol, if a task tries to acquire an already locked global resource, it starts spinning in a non-preemptive fashion using a FIFO non-preemptive spinlock. The chosen spinlock implementation is the *Abortable CLH Lock* [13], whose algorithm is designed with a logical FIFO queue that allows the task waiting for the resource to be unblocked in the arrival order. This property allows the multi-core system to be analyzable because the maximum spinning time is equal to the number of cores (i.e., number of the slots in the FIFO) multiplied for the longest resource locking time of all tasks.

Another essential mechanism implemented to prevent the stall of the entire system is a *deadlock avoidance* mechanism based on a total ordering of all resources. In the configuration stage of the system, the system-integrator must specify a total nesting ordering of the resources. With such kind of information, the kernel can detect a violation of this ordering at run-time, and it can throw an exception. This behavior complies to the clause D.14 (Defensive Programming) of [1].

Fault handling. As the clause D.26 of [1] states, the system has to be able to detect faults, which might lead to failure, and provide the basis for countermeasures in order to minimize the consequences of such failures. Examples of possible faults that the kernel can detect include the attempt to violate the isolation, illegal accesses to memory, etc. To provide a primitive mean of fault handling, we rely on the mechanism of notification for the exceptions. In the design of this mechanism, the concept of signals was taken as a model. A signal represents a generic mechanism used to deliver a notification to a task. Each application must register an exception handler callback that is in charge to manage the faults. Furthermore, the entire system must include a unique supervisor application that is executed if, for some reasons, an exception handler is not able to handle the exception. If the application's exception handler or the supervisor application succeed in managing the exception, the system can continue its normal execution; otherwise the supervisor application is in charge to put the system in a fail-safe state.

Non-blocking Inter Task Communication mechanism. The communication mechanism in the proposed kernel relies on the general concept of channels based on the message-passing paradigm. The channel is intended as a logical link used by two different tasks to communicate. The type of channels provided by our solution is *mono-directional*; namely it provides support to information flow in only one direction. The semantics of communication is *asymmetric*, and this means that one task can send information onto the channel while N tasks can receive the information. Channels work in a pure asynchronous way. In fact, establish a communication using the synchronous semantics means waiting until two tasks encounter at a point in time called rendezvous point. However, in a real-time system, this behaviour leads to system unpredictability due to the difficulty of estimating tasks worst-case execution times. In general, the asynchronous paradigm is more suitable for real-time systems, and indeed, if no

unpredictable delays are introduced during the communication, timing constraints can be guaranteed without increasing the complexity of the system. Furthermore, each channel must be associated with a working mode among two available: *sampling* and *queuing*. These modes are similar to those described by ARINC-653 avionic standard [14]. The sampling mode is intended to transport messages with the same payload structure but updated data. A message remains in the channel until a new occurrence of the message overwrites it. In this mode, the channel must be associated with a *validity period* that indicates the maximum acceptable age of a valid message. When the message is read, if the validity period expires, a warning is issued although the message can still be retrieved. Channels working in sampling mode implements a non-reliable policy, suitable for real-time applications in which tasks are interested in receiving fresh data rather than the complete message history. On the contrary, the queuing mode is intended to transport messages with uniquely different data, and therefore, a message is not allowed to overwrite previous messages transmitted on the same channel. Therefore, channels associated with queuing mode have to buffer multiple messages in the message queue. A maximum number of buffered messages must be specified for each channel. Sending a message into a full-queue or reading a message from an empty queue generates an error. However, in no case, a task is blocked reading or sending a message. Channels working modes are compared in Tab. I. Briefly, the sampling mode represents a predictable and best-effort way of communicating, while the queuing mode is thought to enforce delivery correctness.

Tab. I. Comparison between sampling and queuing working modes.

	Sampling	**Queuing**
Number of messages	One	Fixed at config. time
Reliability policy	Best-effort	Always delivered
Consumption policy	Never consumed	When all tasks receive
Timing policy	Expiration field	No timing policy
Arrangement policy	Last win	First In - First Out

Low overhead with tickless activation. Commonly, available commercial RTOS are tick-based; namely, the kernel runs periodically with each timer tick. Timer ticks can be programmed to trigger an interrupt to the normal flow of execution. This type of behavior allows the kernel to monitor the state of the system and make decisions constantly. However, the resulting overhead is quite significant, since the execution flow is interrupted many times per second without a real need. The kernel described in this article is tickless: it does not run periodically but only when an event occurs in the system (i.e., in case of task activation, deadline miss, budget refill) or it is explicitly called using a system call. As also described in [15], the tickless paradigm allows less overhead than the tick-based one since the number of interrupts of the timer, and the cumulative kernel overhead, decreases dramatically.

IV. Conclusions

The demand for real-time safety-critical systems is continuously increasing. This paper presents the design of a kernel compliant to the EN-50128 standard that incorporates the preferences of railways engineers. The kernel design focused on supporting state-of-the-art real-time solutions such as per-task budget control to provide reliable time isolation. Moreover, the development provided memory virtualization techniques and a programming model based on tasks and applications for spatial isolation. An exception handling mechanism has been designed to handle faults to achieve the safety-critical requirements. The communication mechanism allows tasks to easily send messages to each other with the use of two types of channels that works asynchronously. As future works, other kinds of hardware architectures will be supported, like the Intel x86_64. A further step will be the development of a Hypervisor both as additional fault-isolation and spatial-isolation improvement and support for mixed-criticality applications.

References

[1] C. E. 50128, "Railway applications - communication, signalling and processing systems - software for railway control and protection systems," 2011.

[2] D. Kästner and C. Ferdinand, "Applying abstract interpretation to verify en-50128 software safety requirements," in *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, T. Lecomte, R. Pinger, and A. Romanovsky, Eds. Cham: Springer International Publishing, 2016, pp. 191–202.

[3] Y. Zhao, Z. Yang, and D. Ma, "A survey on formal specification and verification of separation kernels," *Frontiers of Computer Science*, vol. 11, no. 4, pp. 585–607, Aug 2017.

[4] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, ser. Real-Time Systems Series. Springer US, 2011.

[5] R. Palin, D. Ward, I. Habli, and R. Rivett, "Iso 26262 safety cases: Compliance and assurance," in *6th IET International Conference on System Safety 2011*, Sep. 2011, pp. 1–6.

[6] SysGo, "Pikeos: En 50128 sil4 certification on multi-core." [Online]. Available: https://www.sysgo.com/solutions/safety-security-certification/en-50128

[7] G. Software, "Integrity rtos receives cenelec en 50128." [Online]. Available: https://www.ghs.com/news/20100302_CENELEC_EN_certification.html

[8] M. I. S. R. Association and M. I. S. R. A. Staff, *MISRA C:2012: Guidelines for the Use of the C Language in Critical Systems*. Motor Industry Research Association, 2013.

[9] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Fault isolation for device drivers," in *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE, 2009.

[10] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Real-Time Systems*, vol. 1, no. 1, pp. 27–60, 1989.

[11] T. P. Baker, "A stack-based resource allocation policy for realtime processes," in *[1990] Proceedings 11th Real-Time Systems Symposium*, Dec 1990, pp. 191–200.

[12] P. Gai, G. Lipari, and M. Di Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*. IEEE, 2001, pp. 73–83.

[13] M. Herlihy and N. Shavit, *The art of multiprocessor programming*. Morgan Kaufmann, 2011.

[14] J. Garrido, J. Zamorano, and J. A. de la Puente, "Arinc-653 interpartition communications and the ravenscar profile," *ACM SIGAda Ada Letters*, vol. 35, pp. 38–45, 12 2015.

[15] S. Siddha, V. Pallipadi, and A. Ven, "Getting maximum mileage out of tickless," in *Proceedings of the Linux Symposium*, vol. 2. Citeseer, 2007, pp. 201–207.