Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 605–609
DOI: 10.15439/2022F163
ISSN 2300-5963 ACSIS, Vol. 30

Resource Partitioning in Phoenix-RTOS for Critical and Noncritical Software for UAV systems

Hubert Buczyński, Krzysztof Cabaj
Warsaw University of Technology
Nowowiejska 15/9, 00-665 Warszawa, Poland
Email: {hubert.buczynski.dokt, krzysztof.cabaj}@pw.edu.pl

Paweł Pisarczyk
Phoenix Systems Sp. z o.o.
Ostrobramska 86, 04-163 Warszawa, Poland
Email: pawel.pisarczyk@phoenix-rtos.com

***Abstract*—Modern embedded systems' increasing complexity and varied safety levels make it hard to coordinate all functionalities within a single run-time environment. Access to more advanced and capacious hardware changes the trend from utilising many separated platforms into one managing the whole compounded airborne system. Providing an appropriate isolation and synchronisation level is achievable only by adapting an operating system with separation mechanisms into UAV systems. This paper introduces Phoenix-RTOS, the microkernel structured real-time operating system designed to be consistent with aerospace standards DO-178C and ARINC 653. The current market offers many counterparts like VxWorks, Integrity 178, PikeOS and many others. These products are well known and used in leading-edge avionics and space projects. However, it is not possible to use them in many low-budget projects due to the high price. The Phoenix-RTOS differs from others and is an open-source project becoming a standard solution for energy, gas meters and UAV systems.**

In this paper, we focus on the currently designed mechanisms of microkernel architecture for providing a mixed-criticality system, particularly for compliance with ARINC 653. Engineers have been identifying time and space partitioning issues to cope with tight worst-case execution bounds of critical tasks.

I. Introduction

THE Unmanned Air Vehicles (UAV) market increased significantly in recent years. However, plans about flying machines carrying out missions of particular importance (e.g. transport of hazardous substances and medical supplies) above our heads in inhabited areas cannot become a reality without ensuring the high safety standards of airborne systems. According to European Union regulations introduced in 2019 [1], the UAV can be allowed to conduct a mission in the urban environment only after a positive certification process conducted by the civil aviation agencies such as European Union Aviation Safety Agency (EASA) or Federal Aviation Agency (FAA). To reduce the time of the certification process, the manufacturers choose certified Real-Time Operating Systems (RTOS) to run their critical and non-critical applications. The current market offers several operating systems compliant with aviation standards, but only large companies can afford them due to high license costs. To overcome this obstacle Phoenix Systems has started working on Phoenix-RTOS 178 version compliant with aviation standards [1]. Making a system as an open source product allows manufacturers with a limited budget to enter the market. The new version of the Phoenix-RTOS is designed to comply with DO-178C and ARINC 653 standards described in the next paragraph.

A. Aviation Standards

Standards play a crucial role in the aerospace industry. One of the most important is DO-178C, the fulfilment of which allows usage software in airborne devices. DO-178C defines five levels of failure caused by software: catastrophic, hazardous, major, minor, and no effect. Moreover, it defines five corresponding Design Assurance Levels (DAL) from the most rigorous level A to level E. Depending on the DAL level, tenants' appropriate process management has to be fulfilled. Another one, Aeronautical Radio Incorporated (ARINC) standard focuses on technical aspects that define a series of detailed rules for a dedicated area, like data transmission protocols, cockpit user interfaces, and many others. The aerospace projects are always compliant with DO-178C, however meeting the ARINC standards depends on project's requirements. The best policy for operating system development for critical applications is described in standard ARINC 653. Running different critically level software on the same hardware platform is the primary domain of Integrated Modular Avionics (IMA). To realize an IMA approach in RTOS, the Partitioning Operating System (POS) concept should be introduced to support spatial and time partitioning [10]. In the avionics industry, the Avionics Application Standard Software Interface, called APEX provided by ARINC 653, has been introduced as a set of guidelines to provide a standardized interface between POS and avionics applications [10]. The leading role of the ARINC 653 is to improve the safety and certification process of safety-critical systems and outline the architectures approach for POS's engineers [10]. More details about APEX are presented in the fourth section.

B. History of Phoenix-RTOS

The idea for writing a new RTOS originates from the Warsaw University of Technology where the prototype of Phoenix-RTOS was developed from scratch as a master thesis [1] in 1998. The rapidly growing Internet of Things (IoT) market resulted in an industry need where a new operating system with efficient implementation and rich functionalities to be required.

Due to this fact, the second version of Phoenix-RTOS was developed by Phoenix Systems and widely implemented in data concentrators for the smart grid, smart energy meters and smart gas meters. Phoenix-RTOS 2 is recognized on the market as a real-time system for the smart grid and software-defined solutions.

The third version of the system based on a microkernel architecture has been developed to be easily used in microcontroller-based low power devices and more advanced processors. The new approach provided in version three allows the use of Phoenix-RTOS on a broad range of processors architecture from the smallest ones like ARM Cortex-M, to more those complex like ARM Cortex-A, ia32 or RISC-V [1]. Employing the Phoenix-RTOS in version three to the energy meters sector has proven its applicability for high complexity systems which work in harsh environments. The experience obtained in the industry moved the company forward in achieving the next milestone to make Phoenix-RTOS consistent with DO-178C standard in a design assurance level A.

Phoenix-RTOS compliant with DO-178C is developed as an open-source project under the BSD license. From the time of writing this paper, only several open-source competitors supporting ARINC 653 have been found, which are described in the third chapter. The rest of the systems for critical purposes are commercial. It is strongly believed that by providing the system as an open-source product, the gap in the market is filled for projects with a limited budget, and the topic of critical systems will be covered.

This paper presents our consideration for partitioned based microkernel development in Phoenix-RTOS. A current trends analysis in the industry has been conducted and there are visible alternatives that can be offered by the open-source product. The research focuses on implementing microkernel mechanisms to support spatial and time partitioning to fill ARINC 653 requirements.

II. Mixed Criticality System

Operating systems for critical purposes should be highly reliable. As is considered in *Tanenbaum et al.* [5] they should not be interrupted entirely or halted to recover from a failure that occurs in a module that is not in the critical execution path of a service or an internal operation [4]. To achieve this goal, operating systems should provide safety and security functionalities not to allow the non-critical zone's fault to propagate to a critical one. Safety and security functions seem to be similar, and although these terms have common elements, they differ. This chapter presents the concept of safety-critical and security-critical systems in more detail. The next one presents examples of systems divided into appropriate categories.

A. Safety-critical

The main challenge tackled by a safety-critical system is to provide a clear border between different critical zones called partitions. Due to this fact, these kinds of operating systems are often called partitions kernels. Partitioning mechanisms should provide spatial and temporal separation [2]. Moreover, the partition kernel is in charge of resources management such as I/O devices to permit access to only assigned parts of the system. To enforce spatial separation, each individual partition runs in a hardware-protected address space. For this purpose, the Memory Management Unit (MMU) performs mapping of virtual to physical addresses. To enforce time separation, each partition and thread have a dedicated time slot of the CPU in which the actions have to be completed. The static analysis is obligatory to define the Worst-Case Execution Time (WCET) of each running partition and process. Based on the WCET the best scheduling algorithm is selected to take control of the CPU when attempts of time overrun occur. The central concept of a safety system requires static resource allocation for partitions and static analysis of system performance. Safety-critical systems are compliant with DO-178C and ARINC 653. However, standards do not impose requirements describing how spatial and time separation should be performed. The chosen solutions can differ between individual systems.

B. Security-critical

The security-critical systems originate from the concept of Multiple Independent Levels of Security (MILS) specification, which is a high-assurance architecture based on separation and information flow security [3]. The foundation of MILS is a Separation Kernel (SK), which is responsible for adherence of data isolation, damage limitation and resource partitioning. SK extends partitioning kernels of sets of specific functionalities to enforce security separation, and information flow [2]. The security requirements are known as Common Criteria (CC) [3] establishing seven Evaluation Assurance Levels (EAL) from 1 to 7, which is the most rigorous. Moreover, CC defines robust requirements dedicated to an operating system called Separation Kernel Protection Profile (SKPP). To obtain satisfying certification for EAL7, the partitioning mechanisms have to be rigorously verified using formal methods [3]. Compared to partitioning kernels, SK provides a more dynamic environment that does not have to be known ahead of time. The typical approach to realizing separation kernel is based on embedded hypervisor and software virtualization mechanisms. The hypervisor layer manages the system resources and virtual partitions, which are capable of running guest operating systems with different levels of critically [2].

III. Background and Related Work

Operating systems for critical applications is a complex topic that demands a considerable workloads and financial resources. Due to this fact, only several companies can deliver reliable products. The most known and widely used systems on the market are presented in the following subsections. The last paragraph is devoted to open source projects.

A. Lynx Software Technologies

The Lynx Software Technologies has on offer two operating systems for critical application: LynxOS-178 and LynxSecure.

The first version of the system, LynxOS-178, is a safety-critical system. It meets the DO-178C DAL A regulations and supports POSIX, and APEX standards [6]. However, this version of OS would not meet the highest EAL7 criteria, which require formal mathematical methods to prove the security of SK. For this reason, LynxSecure is introduced to provide a separation kernel based on hypervisor virtualization. The second version OS design includes safety and security domain isolation, trusted execution environments and reference monitor plugins such as firewalls and encryption [6]. A popular solution for complex systems is to use LynxSecure to provide secure partitioning and LynxOS-178 to launch critical applications compliant with APEX.

B. GreenHills

INTEGRITY 178 is a world-leading RTOS for safety and security applications certified both to the highest level of DO-178C DAL A and SKPP/EAL6+ [7]. The producer claims tasks protection in the guaranteed time window domain. The space domain is arranged by static memory allocation and hardware enforcement of MMU for each partition. The system isolates applications and data into different security domains to ensure the MILS environment. As a separation kernel, INTEGRITY 178 provides a virtualization layer in the userspace instead of in the kernel [7].

The GreenHills also offers a version for multicore architectures - INTEGRITY 178 tuMP. As one of the few systems that provide full flexibility in choosing the software multi-processing architecture, ranging from simple Asymmetric Multi-Processing (AMP) to modern Symmetric Multi-Processing (SMP) to Bound Multi-Processing (BMP) for the highest combination of determinism and utilization [7].

C. WindRiver

The WindRiver company takes a similar approach as Lynx Software Technologies. In the offer can be found two versions of software for critical purposes. The first one VxWorks 653 is compliant with DO-178B/C DAL A and ARINC 653 [8]. The second one VxWorks MILS provides compliance to SKPP using a hypervisor. To combine safety and security mechanisms, two versions of WindRiver products are used side by side.

D. SYSGO

The flag product of SYSGO company is a PikeOS compliant with DO-178C DAL B and ARINC 653 [2]. Although the system provides safety and security-critical mechanism, it is not compliant with SKPP. PikeOS offers a separation kernel-based hypervisor with multiple partitions for many other operating systems and applications [9]. The engineers from SYSGO developed a similar concept as the GreenHills company for INTEGRITY-178. The PikeOS does not need an external hypervisor, its internal layer provides security and virtualization between partitions.

E. Open Source Systems

According to a survey presented by researchers in [2], the majority of open-source projects are academic implementations. Special consideration should be given to POK and XtratuM operating systems due to compliance with ARINC 653. However, none of them is compliant with DO-178C. Furthermore, there are other systems worth mentioning like Quest-V or Muen. Although they provide some mechanism of spatial and temporal partitioning of resources, they do not comply with any of the considered standards.

IV. ARINC 653

The primary objective of ARINC 653 is to fulfil the IMA requirements to provide an environment for the independent execution of avionics applications utilizing different critical levels. The majority scope of the regulation defines the Application Programing Interface (API) between the operating systems and avionics applications. One of the benefits of standard usage is to provide high portability of avionics software to run on different operating systems.

The standard consists of five parts [11]. Part 0 describes a general overview about ARINC 653. Part 1 summarises the required services to be supported by APEX and part 2 reports extended services. Section B in chapter fourth, precisely presents the services concepts. Part 3 of ARINC 653 is divided into two parts: 3A and 3B, and provides a guideline for conformity tests for both required and extended services. Part 4 defines a subset of API specified by part 1. Finally, part 5 describes recommended capabilities for multicore applications. The conformance of the APEX API to the ARINC 653 standard is validated by passing all of test suites defined by part 3 of the standard.

A. General architecture

Fig. 1 shows the example architecture of a system compliant with ARINC 653. The integrated module represents a complete RTOS which consists of a core module and applications. The Core Software (CSW), referred to as the operating system and hardware layer applying many processors, each consisting of one or more processor cores [11]. CSW should provide the ARINC 653 interface, health monitoring system, two level scheduler and suitable time and space separation based on configuration data. The highest layer of the IMA architecture presents either application software partitions and system partitions. The first ones are restricted to using only ARINC 653 calls to interface to the system [11]. The second ones are partitions specific for the operating system requiring interfaces outside of the APEX scope. They manages system specific task such as device drivers or fault management schemes [11] which are not defined by ARINC 653. Both of them are subject to robust space and time partitioning.

B. Static System Configuration

According to the specification, a partition is a program in an application environment that consists of text, data section, stack, own context, and configuration attributes [11]. In this

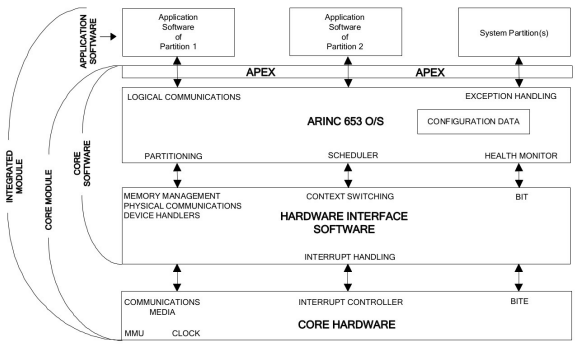

Fig. 1. Example of Integrated Module Architecture [11].

case, a process is a programming unit contained within a partition that executes concurrently with other processes within the same partition [11]. In other words, the process in ARINC 653 glossary corresponds to a thread running inside a process, and the partition refers to a process in UNIX like systems.

Each of the partitions and processes are described by system configuration files. They contain detailed information about the demand for system resources, communication ports, execution windows and scenarios for health monitor systems. The most popular approach for APEX environment description is an Extensible Markup Language (XML) or AADL language.

V. PHOENIX-RTOS 178

Phoenix-RTOS 178 is a new version of the system compliant with DO-178C and ARINC 653, based on the Phoenix-RTOS version three. The new approach follows the rules for mixed safety-critical systems to provide a reliable and robust run-time environment for the avionics industry.

This chapter focuses on the system architecture to explain how spatial and time separation is resolved in Phoenix-RTOS 178. The presented solutions have been developed and deployed on the Zedboard platform, equipped with a Xilinx Zynq-7000 dual-core ARM Cortex-A9 processor family.

A. System Architecture

The system architecture consists of a bootloader and a kernel. The first one, Phoenix-RTOS loader (PLO) can be treated as a first-stage and a second-stage bootloader. Acting as the first-stage bootloader, PLO configures the memory controllers and various supported devices on a dedicated platform. It is also responsible for preparing the board for the kernel and performing built-in tests (BITs) to check correctness of the operation of the critical peripherals. The second-stage bootloader loads the operating system and selected partitions from storage devices to the memory. The fig. 2 shows PLO Command-Line Interface (CLI). Configuration data about the system setup is passed to the kernel by a structure called syspage. The syspage contains all information about board configuration and partition descriptions compliant with ARINC 653 standard. The second system's component is a kernel responsible for providing spatial and time separation by memory and threads

```
Phoenix-RTOS loader v. 1.21 rev: 253ed19
hal: Cortex-A9 Zynq 7000
dev/uart: Initializing uart(0.0)
dev/uart: Initializing uart(0.1)
dev/usb: Initializing usb-cdc(1.2)
dev/flash: Initializing flash(2.0)
cmd: Executing pre-init script
console: Setting console to 0.1
Waiting for input,   200 [ms]
(plo)%
```

Fig. 2. Phoenix-RTOS loader running on Zynq-7000.

```
Phoenix-RTOS microkernel v. 2.97 rev: aa8e57e
hal: Xilinx Zynq-7000 ARMv7 Cortex-A9 r3p0
hal: ThumbEE, Jazelle, Thumb, ARM, Security
hal: Using gic interrupt controller
vm: Initializing page allocator (876+0)/131072KB, page_t=16
vm: [256x][24K][6P]H[17K][36A][127H]PPPP[805.]PPPS[31744.]
vm: Initializing memory mapper: (8105*64) 518720
vm: Initializing kernel memory allocator: (64*48) 3072
vm: Initializing memory objects
proc: Initializing thread scheduler, priorities=8
syscalls: Initializing syscall table [101]
main: Starting syspage programs: 'dummyfs', 'zynq7000-uart', 'psh'
(psh)%
```

Fig. 3. Phoenix-RTOS shell running on Zynq-7000.

management modules. The system is startup by the initial thread being in charge of launching the health monitor, system and user partitions with resources assigned to them by the setup data. After the successful initialization phase, the chosen scheduler algorithm is launched. The fig. 3 shows Phoenix-RTOS providing interaction with user via Phoenix-RTOS shell. The essential kernel's safety-critical mechanisms are described in the next two chapters.

B. Spatial Separation

The crucial element of the system for critical purposes is providing spatial separation for partitions. In Phoenix-RTOS 178, this mechanism is implemented using multi maps and an MMU controller. The multi maps approach allows the simultaneous use of different memory devices such as on-chip RAM (OCRAM) or DDR SDRAM. This solution provides additional physical separation. User is able to use a specific memory for dedicated purposes. If there is a requirement to use the memory with a fast access for critical application, the OCRAM should be chosen. The memory virtualization is provided by the MMU controller and configured based on the information about accessible volatile memory described by the PLO in the syspage.

Another essential element embedded into the memory management module is a caching policy. Phoenix-RTOS 178 invalidates and cleans executable pages and allows the user to define uncacheable memory regions independently.

The memory virtualization provides the same linear memory map for each partition. Switching between them provides an additional delay in changing the MMU controller's translation table. To avoid an overhead, the Address Space Identifier (ASID) mechanism is added to the memory management module to assign a memory map to the partition. Furthermore, the spatial separation is in charge of controlling of the mapping

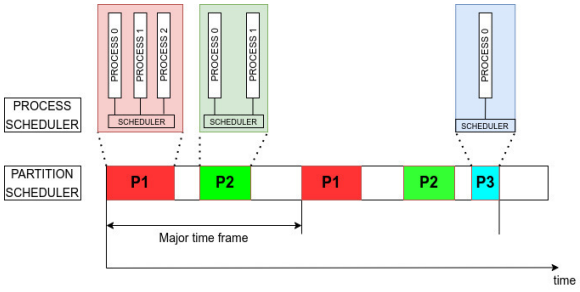

Fig. 4. Example of partition and processes scheduling.

I/O devices' memory to only one partition to not allowing sharing resources.

C. Time Separation

A robust and reliable scheduling policy is an essential element of the operating system to perform critical tasks on time. In an environment compliant with ARINC 653, a two-level scheduler should be introduced. The fig. 4 shows an illustrative scheduling policy for mixed-criticality systems.

The first scheduler supervises the partition work. The most common algorithm for hard real time embedded systems is the Rate Monotonic (RM) scheduling with a fixed priority [12]. In the presented example, each partition has: an assigned priority, WCET, Average Case Execution Time (ACET) and period. The critical partitions have the highest priority allowing to preempt the other ones and to meet deadlines. Making a static analysis using the integration tool, it is possible to verify whether the system is feasible and all partitions can meet their deadlines with the assumed values.

The second scheduler is responsible for scheduling processes within a partition. The ARINC 653 allows to choose a dedicated scheduling policy or the default one is selected.

Keeping WCET bounds of partitions depend on the internal mechanism of the operating system. The common problem for RM scheduling policy is a priority inversion, causing deadlock between partitions or processes. To avoid this situation, the synchronization mechanisms owning dynamic changing priority to the highest one sharing amongst other partitions or threads.

D. APEX Interface

Realization of the APEX services in Phoenix-RTOS 178 is performed using systems calls to the kernel. The interface is implemented in the form of a static library linked to the partition executive file. The library interface checks the correctness of the given arguments and passes them via the Application Binary Interface (ABI) to the kernel running in a privileged mode.

The microkernel architecture provides a message passing interface between servers. The communication services like inter-partition communication use the exact mechanism. The other ones use a dedicated system calls to perform the action defined in ARINC 653.

VI. Conclusion

In this paper, we presented the research work on the mechanisms for integrating critical and noncritical software management by safety operating systems. The growing UAV market and its application will need to use reliable RTOS compliant with DO-178C and ARINC 653. Phoenix-RTOS 178, as an open source project, will be the best choice for manufacturers entering the market with a limited budget.

The presented work shows working Phoenix-RTOS on the Zynq-7000 platform commonly used in airborne systems. Our goal was to highlight the basic concepts of mixed criticality systems and explain the general architecture of Phoenix-RTOS 178. Future work includes completing time and space separation mechanisms as well as the APEX interface in Phoenix-RTOS 178.

VII. Acknowledgements

The presented concepts are the part of the project - *Phoenix-RTOS 178 - new version of the Phoenix-RTOS operating system dedicated for implementation of systems compliant with the guidelines of the DO-178C safety standard* developed by the Phoenix Systems Sp. z o.o.. The project POIR.01.01.01-00-1412/19-00 is co-financed as a grant agreement by European Union represented by the National Centre for Research and Development (NCBiR) in Poland.

References

[1] Phoenix Systems Sp. z.o.o. - Homepage, https://phoenix-rtos.com/documentation. Last accessed 4.12.2021
[2] Y. Zhao, D. Sanan, F. Zhang, and Y. Liu, "High-Assurance Separation Kernels: A Survey on Formal Methods." arXiv, 2017. doi: https://doi.org/10.48550/arXiv.1701.01535.
[3] J. A. Foss, P. W. Oman, C. Taylor, and W. S. Harrison, "The MILS architecture for high-assurance embedded systems," International Journal of Embedded Systems, vol. 2, no. 3/4. Inderscience Publishers, p. 239, 2006. doi: http://dx.doi.org/10.1504/IJES.2006.014859.
[4] M. Ahmed and S. Gokhale, "Reliable Operating Systems: Overview and Techniques," IETE Technical Review, vol. 26, no. 6. Medknow, p. 461, 2009. doi: http://dx.doi.org/10.4103/0256-4602.57831
[5] A. S. Tanenbaum, J. N. Herder, and H. Bos, "Can We Make Operating Systems Reliable and Secure?," Computer, vol. 39, no. 5. Institute of Electrical and Electronics Engineers (IEEE), pp. 44–51, May 2006. doi: https://doi.org/10.1109/MC.2006.156.
[6] Lynxks - Homepage, http://www.lynuxworks.com/. Accessed 4.12.2021
[7] GreenHills - Homepage, https://www.ghs.com/. Accessed 4.12.2021
[8] WindRiver - Homepage, https://www.windriver.com/products/vxworks. Accessed 4.12.2021
[9] PikeOS Homepage, https://www.sysgo.com/pikeos. Accessed 17.12.2021
[10] Delange J and Lec L. Pok, An ARINC653-compliant operating system released under the BSD license. In: Proc. of the 13th Real-Time Linux Workshop, Prague (Czech Republic) 2011
[11] Aeronautical Radio, Inc., 2010, Avionics Application Software Standard Interface: ARINC Specification 653P0-1, 653P1-3
[12] S. Siewert and J. Pratt. Real-Time Embedded Components and Systems with LINUX and RTOS. ISBN: 978-1-942270-04-1